PROJECT REPORT

on

MODELLING OF ARTIFICAL NEURAL NETWORK FOR BEALE FUNCTION

Submitted by

Deepak Lanke

PB23MTECH11003

COURSE NAME: Basics & Applications of AI/ML for Process Systems Engineering (PB5230)

CONTENTS

1. INTRODUCTION

This function is used as a test function in order to evaluate the performance of optimization algorithms.

$$f(x,y) = (1.5 - x - xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x - xy^3)^2$$

where -4.5 < *x,y* <4.5. The number of independent variables is 2 *(x,y).* ANN modelling was performed to develop a functional relationship between the function value and these 2-inputs. The data for the model was generated using the function expression and it was used to build the model.

2. DATA GENERATION

1. Generation of Input and Target Data

The required packages such as ***Numpy*** and *pandas* were imported into the code. The input data was generated in the range of -4.5 < *x,y* <4.5 with a step size of 0.2. These generated data were given as input to the test function and its function value was evaluated.

```
# Defining Beale function
def beale_function(x,y):
    """
    Equation of beale function

    Inputs:
    x and y values

    Outputs:
    Value of beale function at x and y
    """
    return (1.5-x+x*y)**2 + (2.25-x+x*y**2)**2 + (2.625-x+x*y**3)**2
# Generating data for beale function in range in the range -4.5 < x,y < 4.5
with step size 0.2
import numpy as np
X,Y,Z = [],[],[] # Empty lists to store x,y and output values
for i in np.arange(-4.5,4.6,0.2): # -4.5 < x < 4.5
    for j in np.arange(-4.5,4.6,0.2): # -4.5 < y < 4.5
        X.append(i) # Adding x value to list X
        Y.append(j) # Adding y value to list Y
        Z.append(beale_function(i,j)) # Adding beale function value for
respective x and y to list Z
data1 = zip(X,Y,Z) # Concatenation of inputs and outputs
```

2. Saving and reading the file from excel file

The generated data were then concatenated, and it was assigned to a data frame variable using ***DataFrame***. These were then stored in an excel file locally.

```
import pandas as pd
# Creating a dataframe containing inputs and outputs
data2 = pd.DataFrame(data1,columns=['x','y','z'])
# data2 now has 3 columns containing x, y, f(x,y) values for beale function in
the range of -4.5 < x,y < 4.5 with step size 0.2
data2.to_excel('beale_function_data_step=0.2.xlsx',index=False)
```

The data from excel was read using read_excel and store it in an array for further processing.

```
# Reading data from excel sheetdata3 =
pd.read_excel('beale_function_data_step=0.2.xlsx')
```

3. Normalization of Data

The input and output data were normalized between 0 – 1. To normalize the data, ***MinMaxScaler*** tool was imported ***skl.Preprocessing*** package. Then the normalized data was stored for further usage.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# Normalizing data between 0 and 1 using MinMaxScaler
scaled_dataset = scaler.fit_transform(data3.values)
```

4. Data for Training, Validation, and Testing

The normalized data was split into input data and target data required for the development of the ANN model. 70% of the data was taken as the input data, 15% as validation and the rest 15 % as testing data.

No.	Process	Data Percentage
1	Training	70
2	Validation	15
3	Testing	15

```
inputs = scaled_dataset[:,0:2] # Splitting inputs from normalized data
output = scaled_dataset[:,-1] # Splitting output from normalized data
from sklearn.model_selection import train_test_split
# Splitting the data into training data and testing data using
train_test_split
inputs_train,inputs_test,output_train,output_test =
train_test_split(inputs,output,train_size=0.7,random_state=0) # train_size=0.7
```

3. ARTIFICIAL NEURAL NETWORK MODELLING

ANN modelling was implemented for the normalized data using ***TensorFlow*** package in python. ***Sequential and Dense*** functions were imported from TensorFlow and it was used to create the neural network.

```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

1. Neural network using *adam* optimizer

The network structure is as follows

Layer	Activation Function	No. of nodes
Hidden Layers 1	relu	30
Hidden Layers 2	relu	20
Output Layer	linear	1

```python
# Creating ANN model
model = Sequential()
# First layer with 18 nodes and two inputs and "relu" as activation function
model.add(Dense(30,input_dim=2,activation='relu'))
# Second layer with 12 nodes and "relu" as activation function
model.add(Dense(20,activation='relu'))
# Output layer with one output and "linear" as activation function
model.add(Dense(1,activation='linear'))
```

The' ***adam'*** algorithm was taken as the optimizer and '***Mean Square Error (MSE)'*** was taken as the loss function.

```python
# Compiling using optimizer "adam" and loss function "MSE"
model.compile(optimizer ='adam',loss='MSE')
```

The number of iterations (epoch) was fixed as 100 and the validation data percentage was assigned as 15% of the dataset.

```python
# Training data with 100 epochs and validation split 0.15
history= model.fit(inputs_train,output_train,epochs=100,validation_split=0.15)
```

To validate the predicted results, statistical parameter like Mean Square Error (MSE), R_2 was imported from ***sklearn.metrics***.

```python
pred_train = model.predict(inputs_train) # Prediction of training data
pred_test = model.predict(inputs_test) # Prediction of testing data
from sklearn.metrics import r2_score
r2_train = r2_score(output_train,pred_train) # R2 score of training data
print(f"R2 Score for training data: {r2_train:.4f}")
r2_test = r2_score(output_test,pred_test) # R2 score of testing data
print(f"R2 Score for testing data: {r2_test:.4f}")
from sklearn.metrics import mean_squared_error
mse_train = mean_squared_error(output_train,pred_train) # MSE of training data
print(f"Mean Squared Error for training data: {mse_train:.4f}")
mse_test = mean_squared_error(output_test,pred_test) # MSE of Testing data
print(f"Mean Squared Error for testing data: {mse_test:.4f}")
```

No.	**Predicted Data**	R^2	**MSE**

1	Training	0.9743	0.0005
2	Testing	0.9707	0.0004

The scatter plot of Predicted data vs True data was plotted using the scatter plot function imported from ***matplotlib***.

```
import matplotlib.pyplot as plt
fig,plot = plt.subplots(1,2,figsize=(10,5))
plot[0].plot(output_train,pred_train,'y.')
plot[0].plot(output_train,output_train,'-')
plot[0].set_xlabel('True output')
plot[0].set_ylabel('Predicted output')
plot[0].set_ylim(-0.05,1.1)
plot[0].set_xlim(-0.05,1.1)
plot[0].set_title(f'Training Data (adam)')
plot[0].text(0,1,f'R2 = {r2_train:.4f}',fontsize=12,color='magenta')
plot[1].plot(output_test,pred_test,'r*')
plot[1].plot(output_test,output_test,'-')
plot[1].set_ylim(-0.05,1)
plot[1].set_xlim(-0.05,1)
plot[1].set_xlabel('True output')
plot[1].set_ylabel('Predicted output')
plot[1].set_title(f'Testing Data (adam)')
plot[1].text(0,0.9,f'R2 = {r2_test:.4f}',fontsize=12,color='magenta')
plt.tight_layout()
plt.show()
```

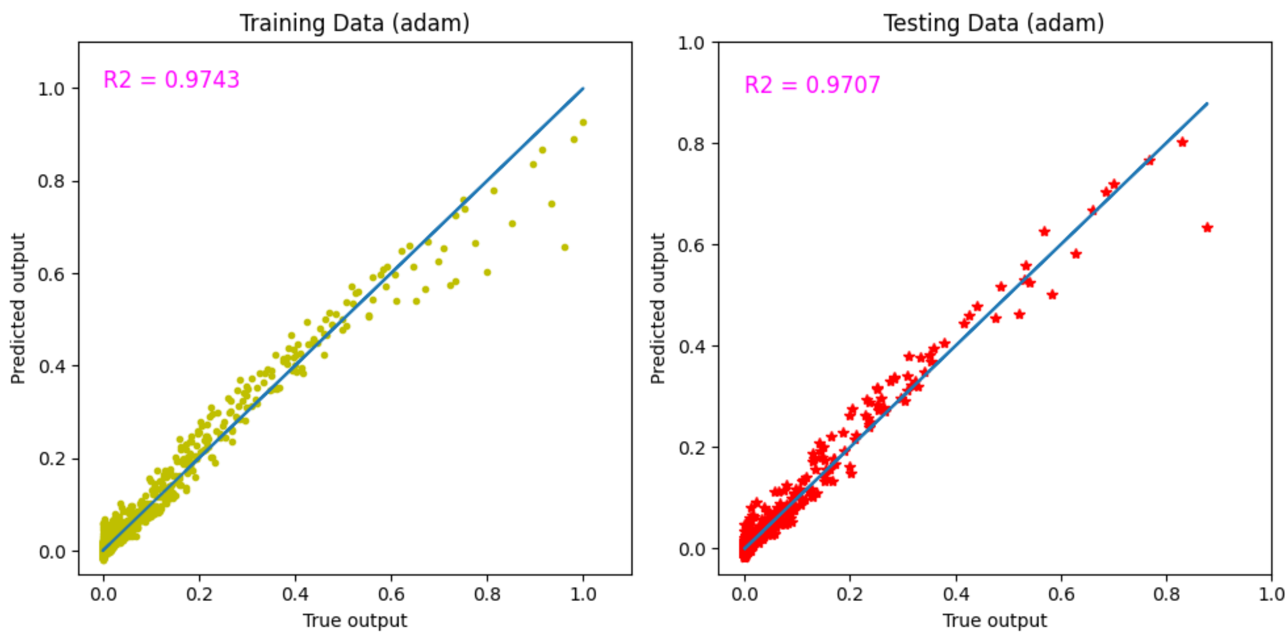


Figure 1: Traning Data Vs Testing Data using adam optimizer

2. Neural network using RMSProp

The same network architecture was now trained with ‘***RMSProp***’ as the optimizer and ‘***MSE’*** as the loss function.

```
# Compiling using optimizer "RMSProp" and loss function "MSE"
model.compile(optimizer ='RMSProp',loss='MSE')
```

No.	Predicted Data	R^2	MSE
1	Training Data	0.9635	0.0005
2	Testing Data	0.9568	0.0004

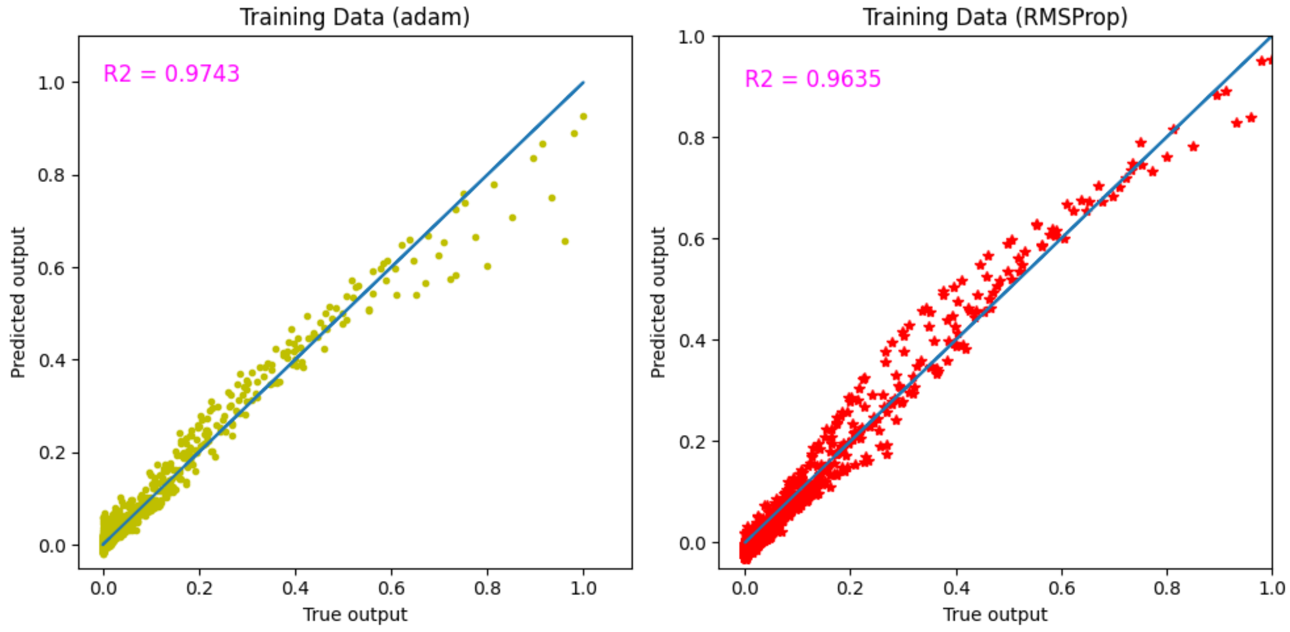


Figure 2: Comparision between adam and RMSProp for Training Data

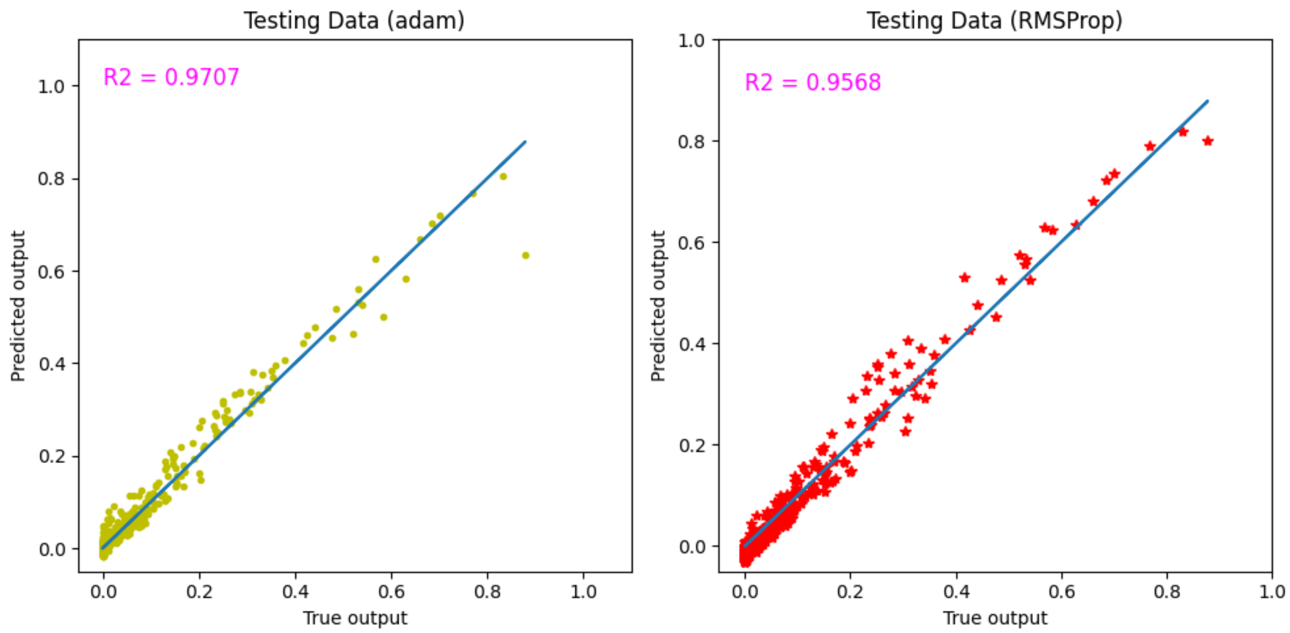


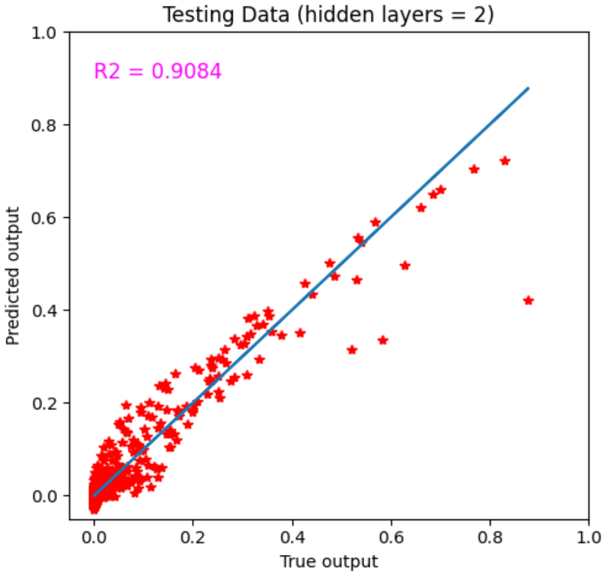
Figure 3: Comparision between adam and RMSProp for Testing Data

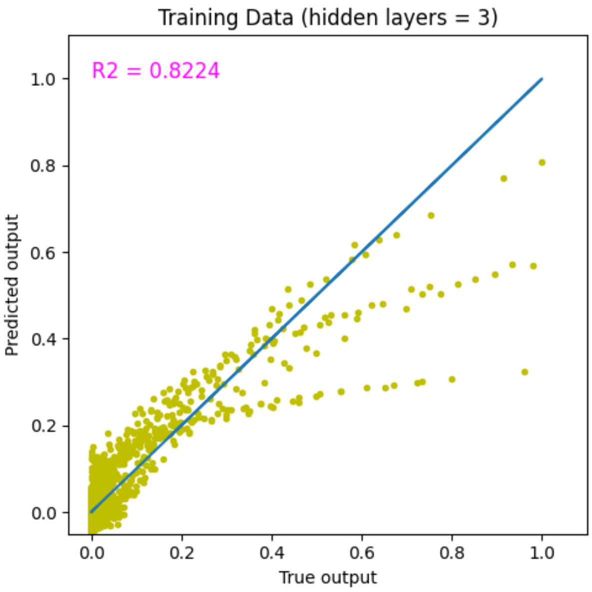
4. RESULTS AND DISCUSSIONS

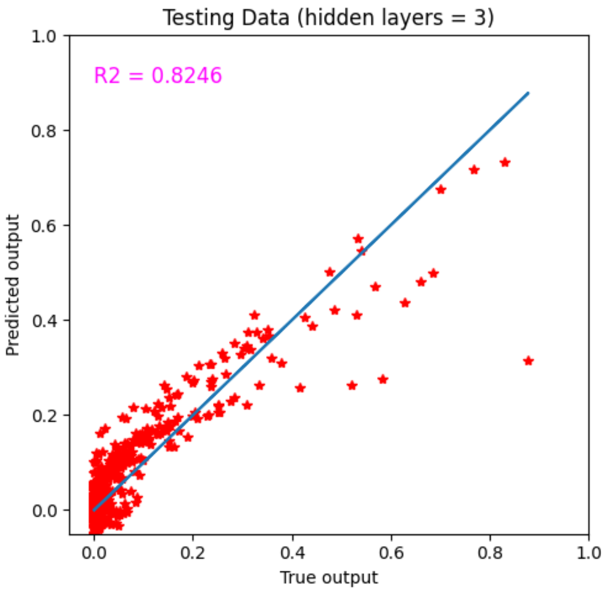
1. Effect of layers

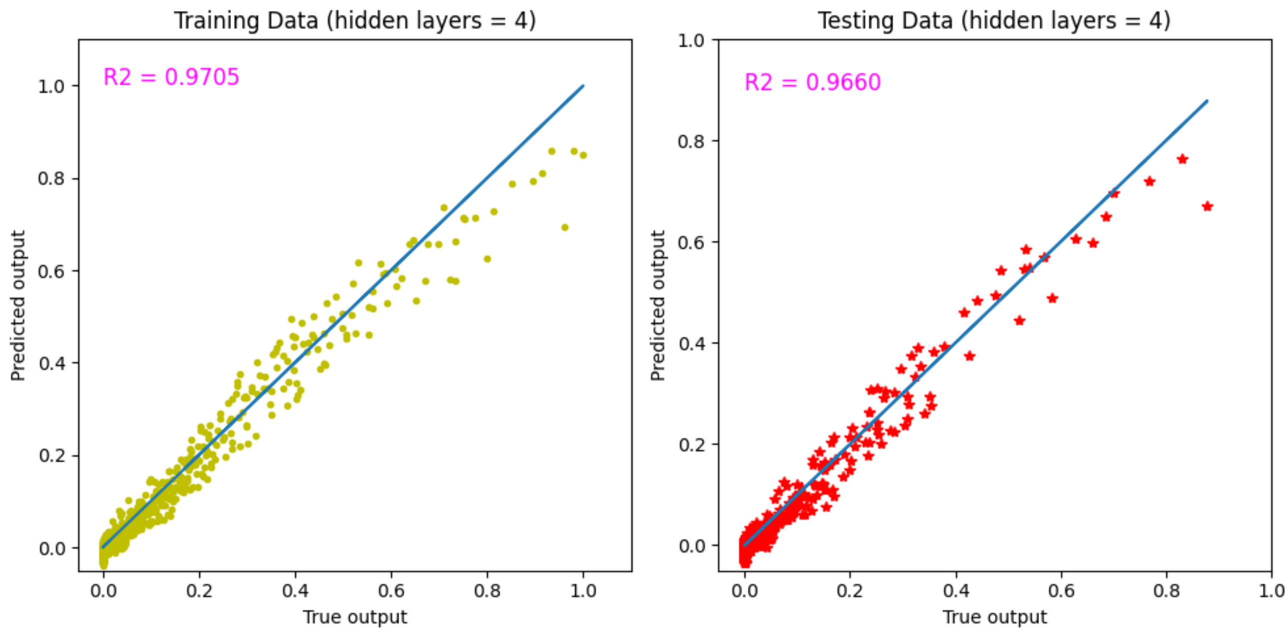
The number of hidden layers of an ANN model plays an important role in the architecture of the neural network. Additional layers were added to the given layer. Study was conducted using 2,3,4 hidden layers and the results were shown in the table below. Each hidden layer has *'relu'* as the activation function.

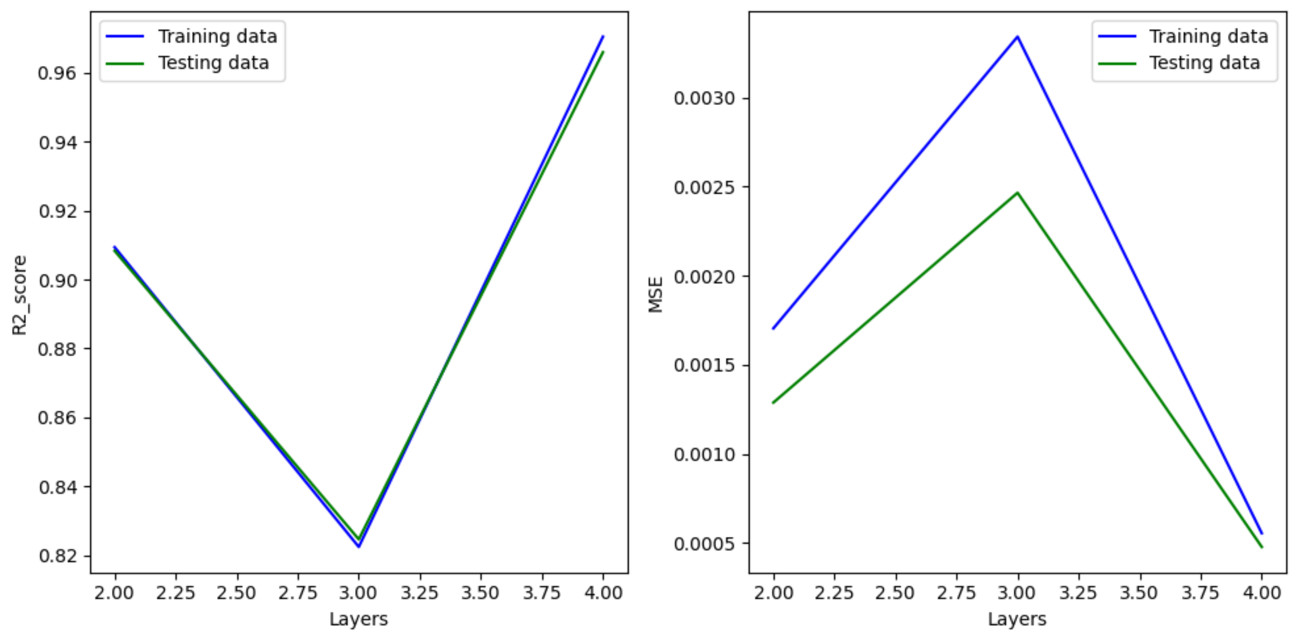
Number of Hidden Layers	Training		Testing	
	R^2	MSE	R^2	MSE
2	0.9094	0.0017	0.9084	0.0013
3	0.8224	0.0033	0.8246	0.0025
4	0.9705	0.0006	0.9660	0.0005







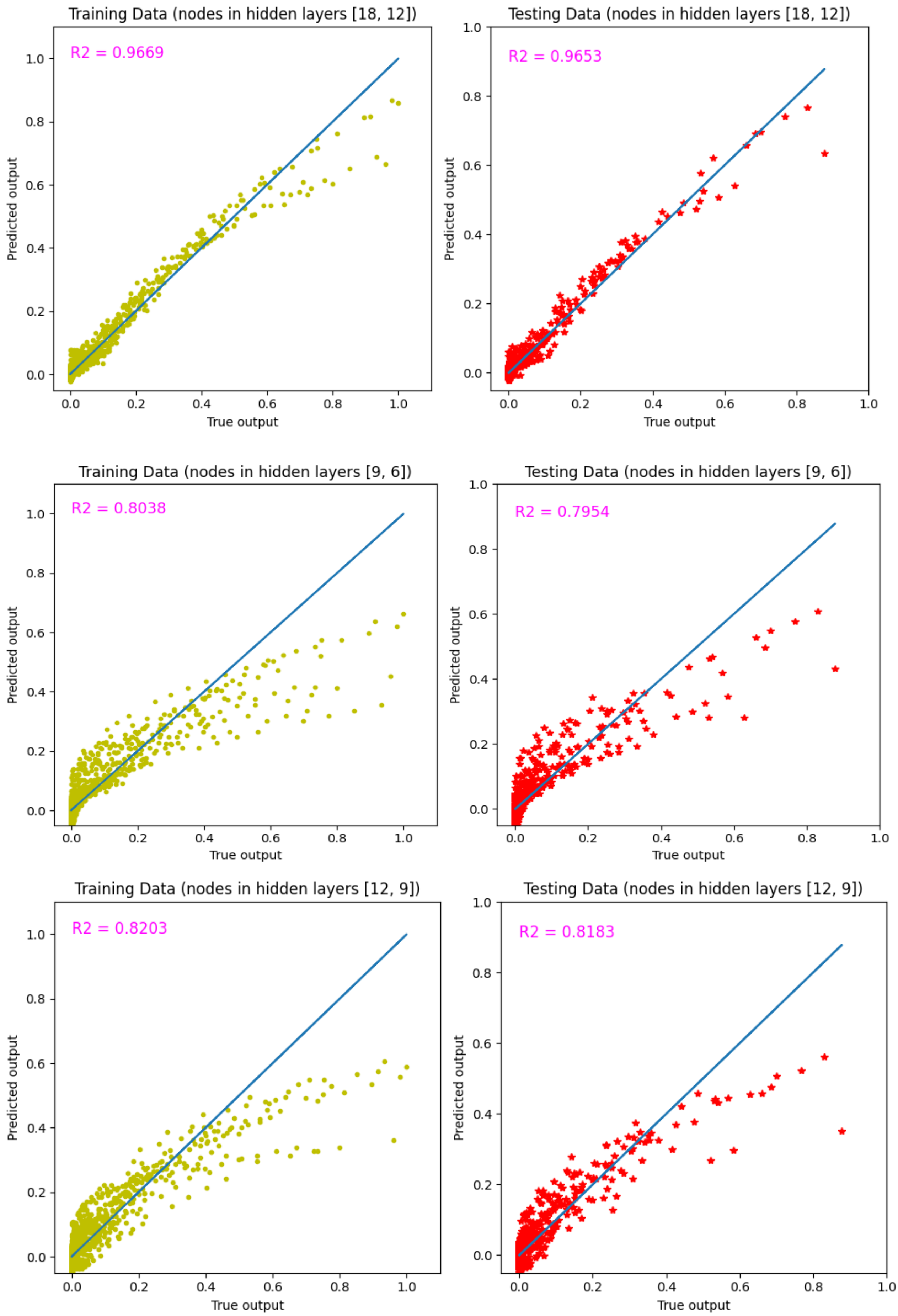


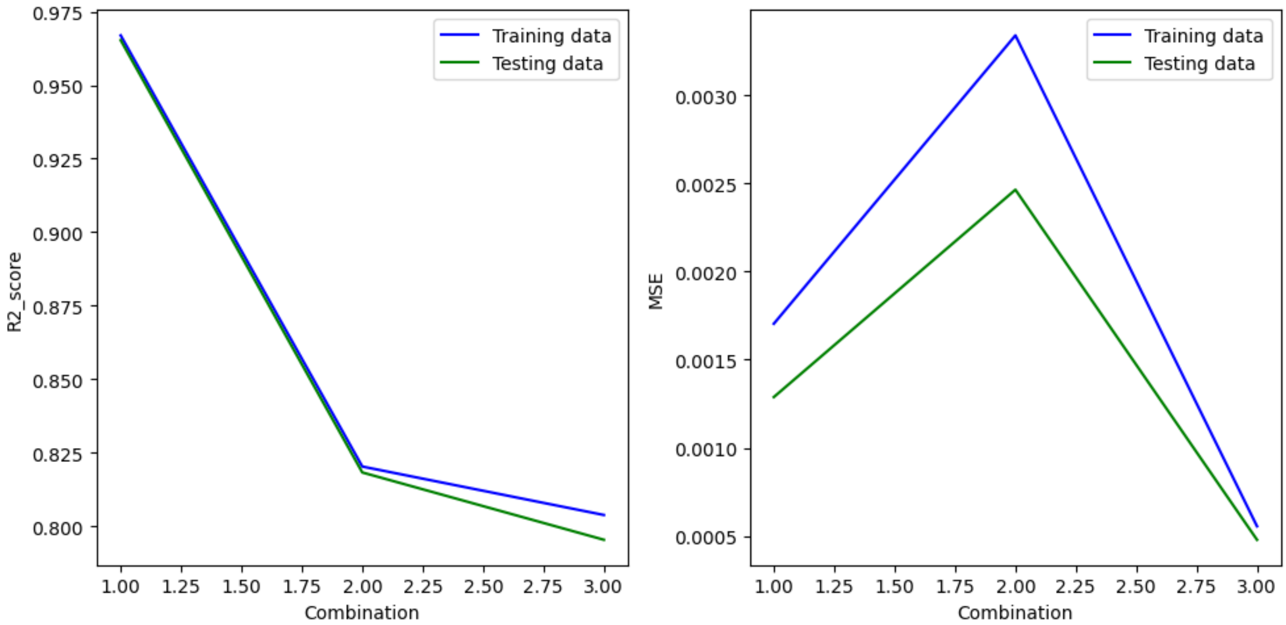


2. Effect of Nodes

The number of Nodes in the hidden layers of the neural network is the important parameter. The number of nodes plays an important role in the performance and predicting capability of the model. The number of nodes in the two hidden layers was varied and the topology which gave the best prediction was found.

Combination	Number of nodes	Training		Testing	
		R^2	MSE	R^2	MSE
1	2-18-12-1	0.9669	0.0006	0.9653	0.0005
2	2-12-9-1	0.8203	0.0034	0.8183	0.0026
3	2-9-6-1	0.8038	0.0037	0.7954	0.0029

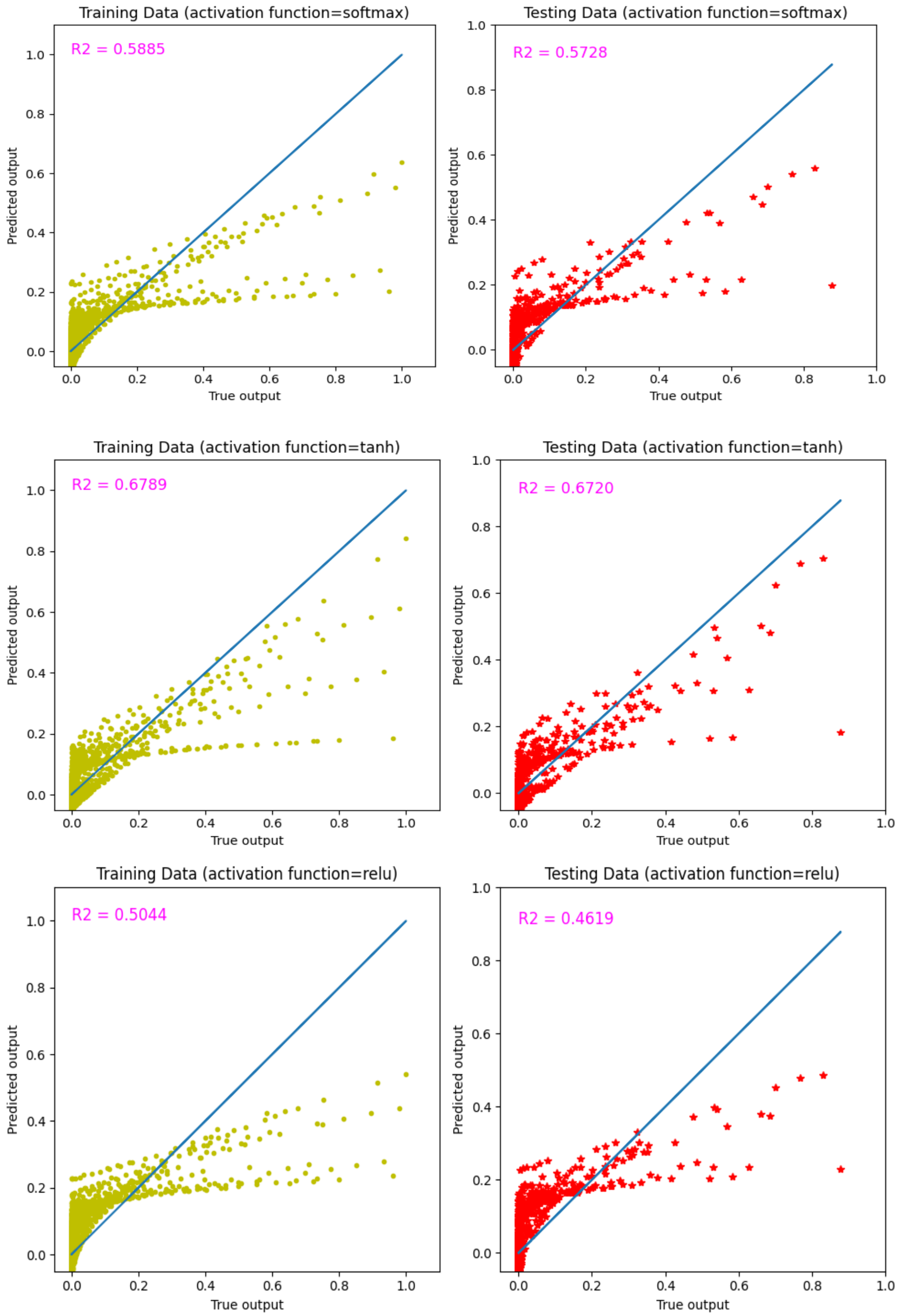

Training Data (nodes in hidden layers [18, 12])
R2 = 0.9669
Predicted output
True output
Testing Data (nodes in hidden layers [18, 12])
R2 = 0.9653
Predicted output
True output
Training Data (nodes in hidden layers [9, 6])
R2 = 0.8038
Predicted output
True output
Testing Data (nodes in hidden layers [9, 6])
R2 = 0.7954
Predicted output
True output
Training Data (nodes in hidden layers [12, 9])
R2 = 0.8203
Predicted output
True output
Testing Data (nodes in hidden layers [12, 9])
R2 = 0.8183
Predicted output
True output

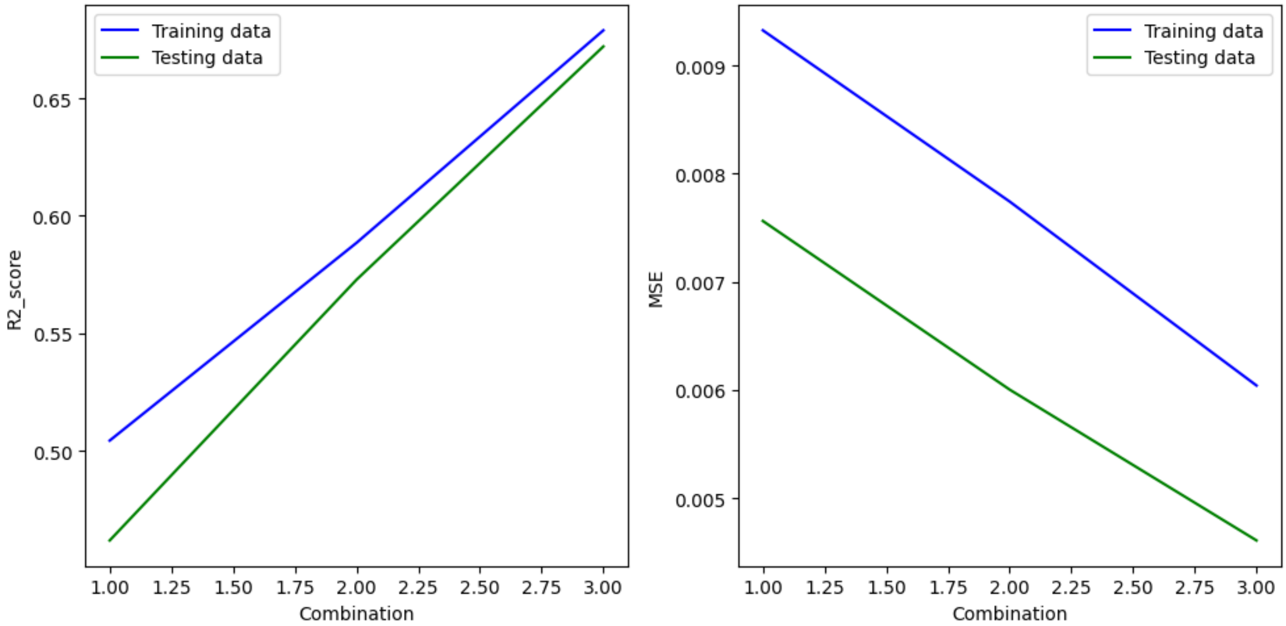


3. Effect of Activation functions

All the other parameter were fixed to study the effect of activation functions. 2 hidden layers were taken. Activation function for both the layers were taken as *'relu','softmax','tanh'* respectively. The results observed are given below

Combination	Activation function	Training		Testing	
		R^2	MSE	R^2	MSE
1	relu-relu-linear	0.5044	0.0093	0.4613	0.0076
2	softmax-softmax-linear	0.5885	0.0077	0.5728	0.0060
3	tanh-tanh-linear	0.6789	0.0060	0.6720	0.0046

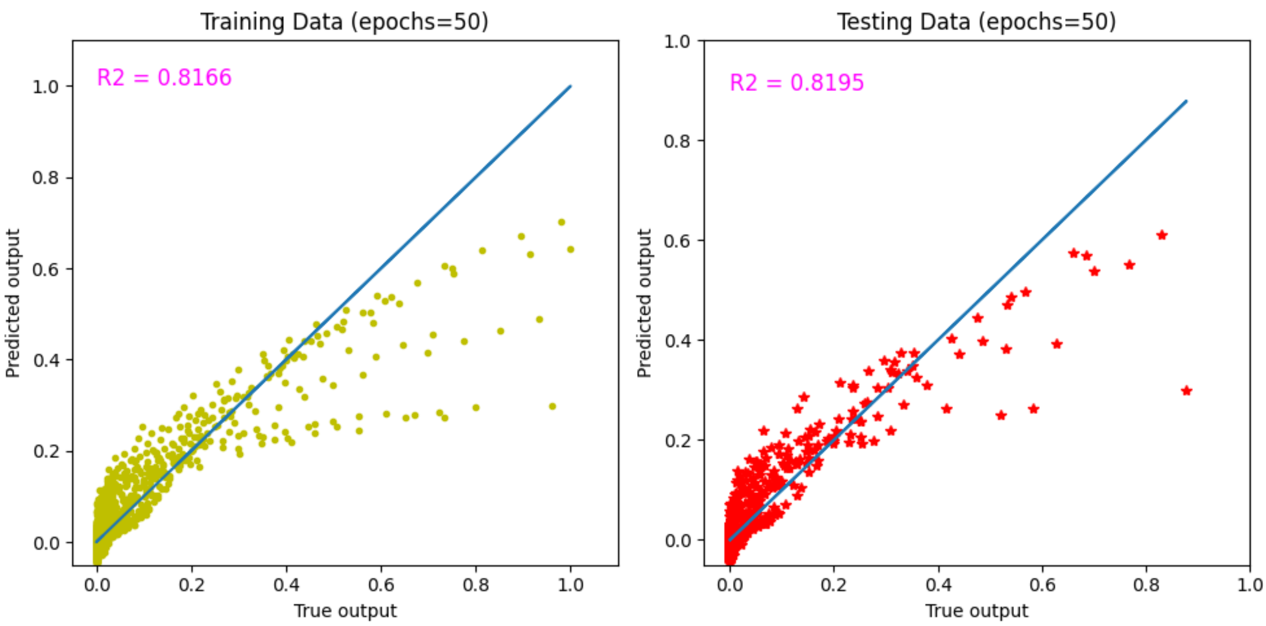

Training Data (activation function=softmax)
R2 = 0.5885
Testing Data (activation function=softmax)
R2 = 0.5728
Training Data (activation function=tanh)
R2 = 0.6789
Testing Data (activation function=tanh)
R2 = 0.6720
Training Data (activation function=relu)
R2 = 0.5044
Testing Data (activation function=relu)
R2 = 0.4619
Predicted output
True output

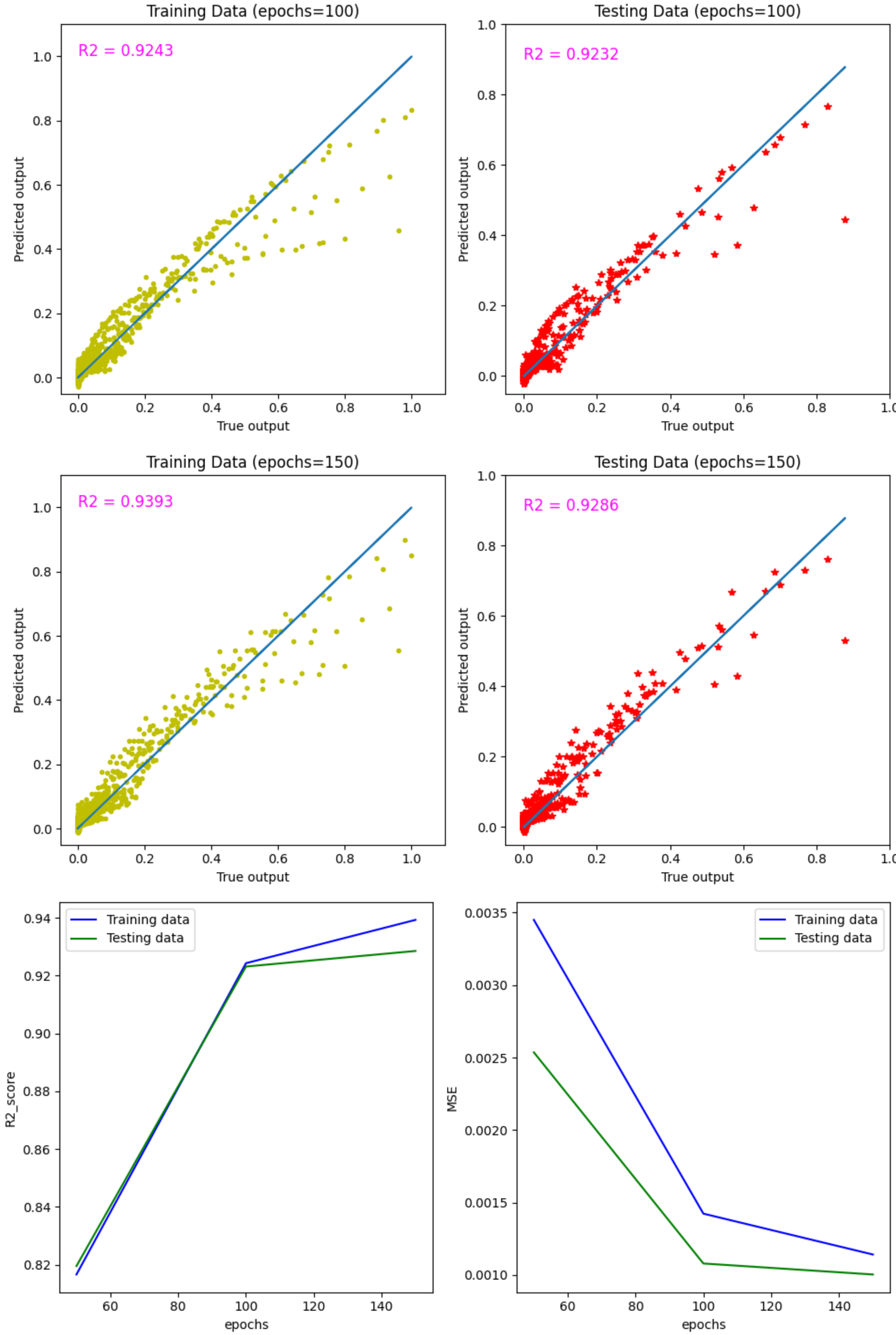


4. Effect of epochs

To study the effect of the Number of Iteration (Epoch) on the prediction performance of the ANN model, the epoch was varied keeping the other parameters fixed as given in the network structure. The results were tabulated below.

Epochs	Training		Testing	
	R^2	MSE	R^2	MSE
50	0.8166	0.0035	0.8195	0.0025
100	0.9243	0.0014	0.9232	0.0011
150	0.9393	0.0011	0.9286	0.0010



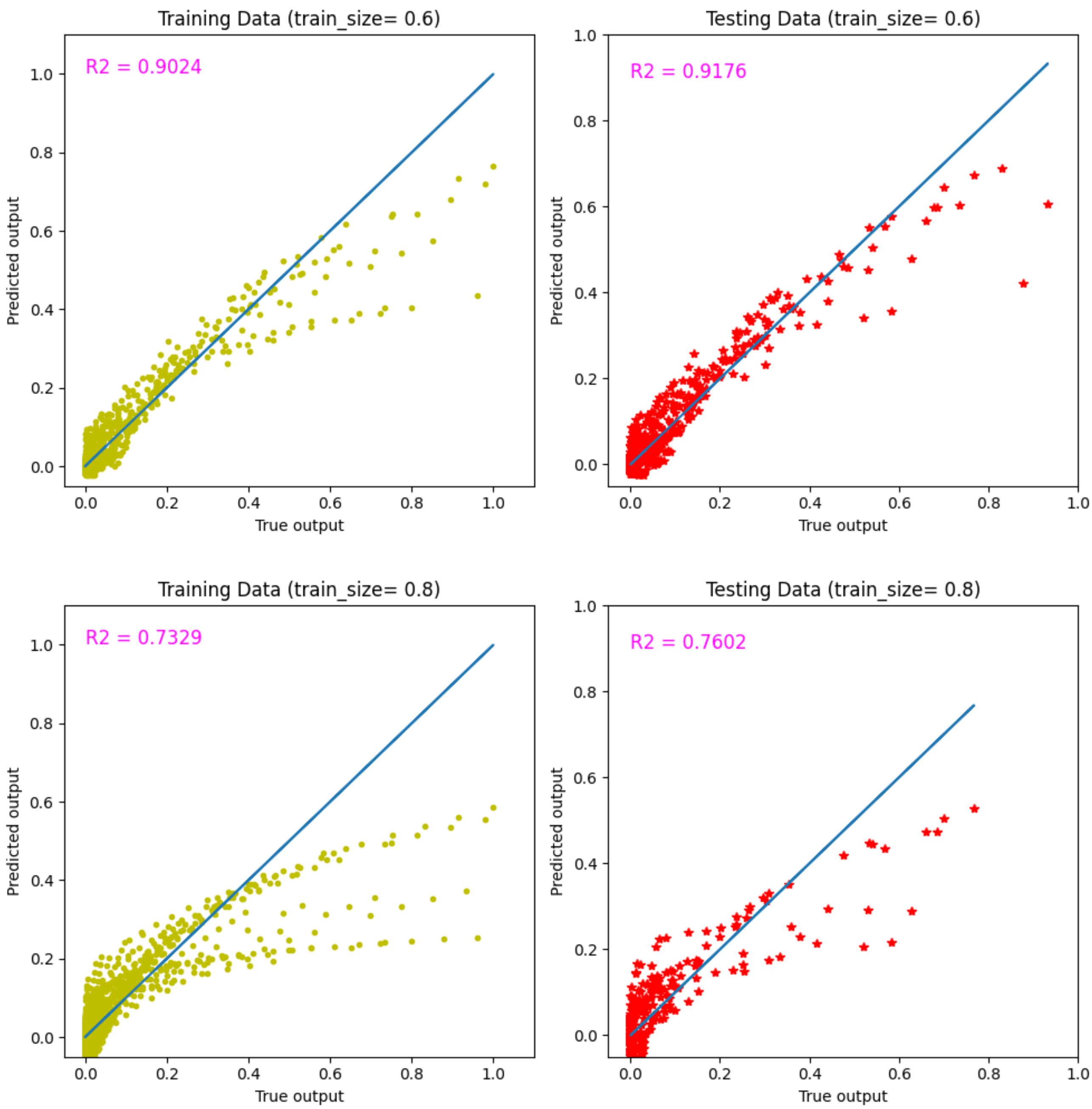


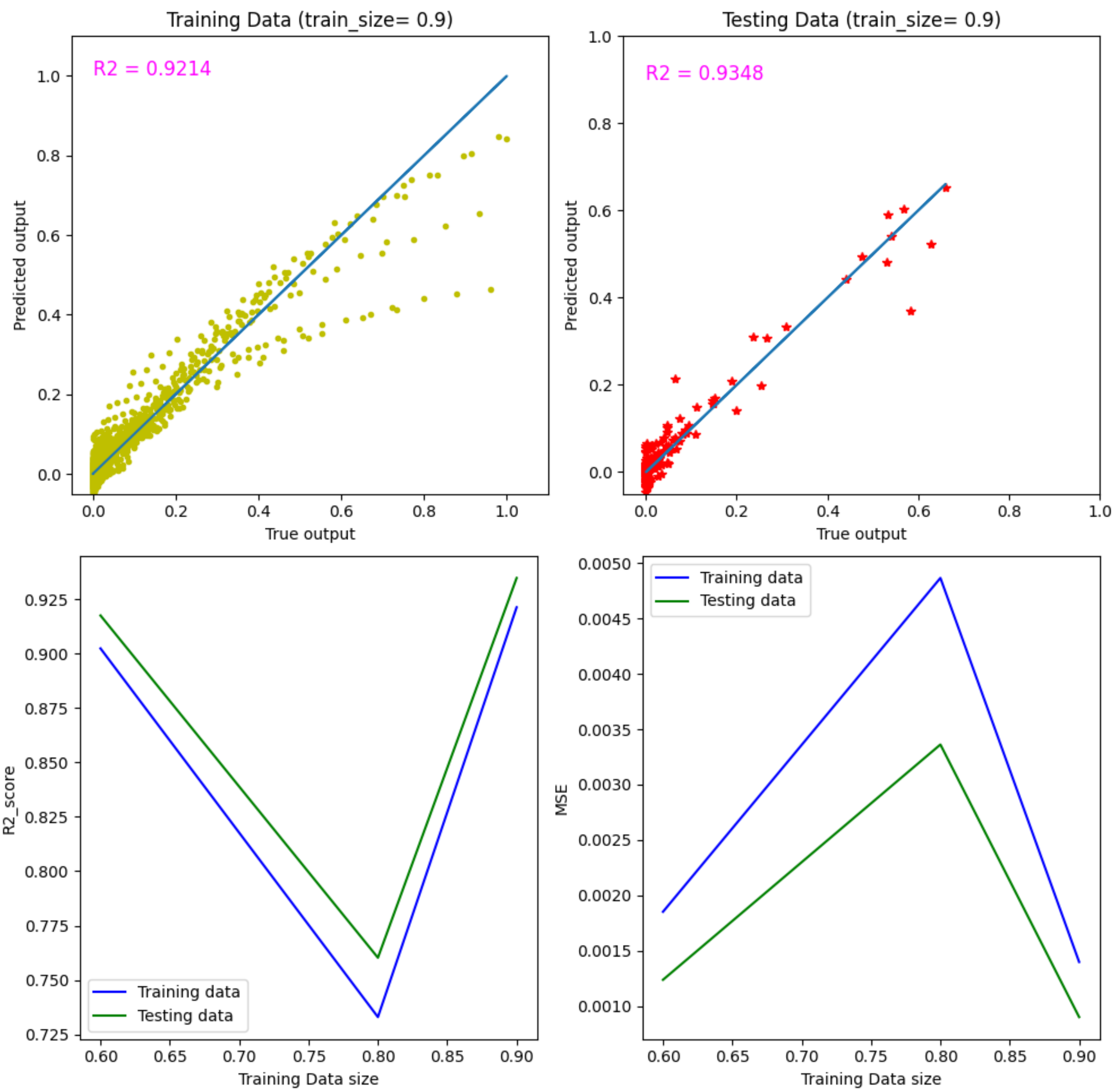
Training Data (epochs=100)
R2 = 0.9243
Predicted output
True output
Testing Data (epochs=100)
R2 = 0.9232
Predicted output
True output
Training Data (epochs=150)
R2 = 0.9393
Predicted output
True output
Testing Data (epochs=150)
R2 = 0.9286
Predicted output
True output
Training data
Testing data
R2_score
epochs
Training data
Testing data
MSE
epochs

5. Effect of Sample size for training

To study the effect of the sample size taken for training on the prediction performance of the ANN model, the sample size taken for training data was varied keeping the other parameters fixed as given in the network structure. The results were tabulated below.

Size of Training Data	Size of Testing Data	Size of Validation data	Training		Testing	
			R^2	MSE	R^2	MSE
51%	40%	9%	0.9024	0.9176	0.9176	0.0012
68%	20%	12%	0.7329	0.0049	0.7062	0.0034
76%	10%	14%	0.9214	0.0014	0.9348	0.0009




Training Data (train_size= 0.9)
R2 = 0.9214
Predicted output
True output
Testing Data (train_size= 0.9)
R2 = 0.9348
Predicted output
True output
R2_score
Training data
Testing data
Training Data size
MSE
Training data
Testing data
Training Data size

5. Conclusion

In our study, we found the best neural network setup for modeling the Beale function. A network with 2 hidden layers, having 64 and 16 nodes respectively, and using ReLU activation functions, performed exceptionally well. We employed the Adam optimizer with a Mean Squared Error (MSE) loss function. Training over 100 epochs, and splitting the data into 70% for training and 15% for both validation and testing, ensured accurate predictions without overfitting. This configuration strikes a balance between complexity and performance, making it the optimal choice for modeling the Beale function.

ANNEXURE

Data Generation

In [ ]:
```
# Defining Beale function
def beale_function(x,y):
    """
    Equation of beale function

    Inputs:
    x and y values

    Outputs:
    Value of beale function at x and y
    """
    return (1.5-x+x*y)**2 + (2.25-x+x*y**2)**2 + (2.625-x+x*y**3)**2
# Generating data for beale function in range in the range -4.5 < x,y < 4.5 with step size 0.2
import numpy as np
X,Y,Z = [],[],[] # Empty lists to store x,y and output values
for i in np.arange(-4.5,4.6,0.2): # -4.5 < x < 4.5
    for j in np.arange(-4.5,4.6,0.2): # -4.5 < y < 4.5
        X.append(i) # Adding x value to list X
        Y.append(j) # Adding y value to list Y
        Z.append(beale_function(i,j)) # Adding beale function value for respective x and y to list Z
data1 = zip(X,Y,Z) # Concatenation of inputs and outputs
import pandas as pd
data2 = pd.DataFrame(data1,columns=['x','y','z']) # Creating a dataframe containing inputs and outputs
# data2 now has 3 columns containing x, y, f(x,y) values for beale function in the range of -4.5 < x,y < 4.5 with step size 0.2
data2.head()
# Saving data to a excel file
data2.to_excel('beale_function_data_step=0.2.xlsx',index=False)
# Reading data from excel sheet
data3 = pd.read_excel('beale_function_data_step=0.2.xlsx')
```

Normalizing the data - preprocessing

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_dataset = scaler.fit_transform(data3.values) # Normalizing data between 0 and 1 using MinMaxScaler
# Splitting the data into inputs and outputs
inputs = scaled_dataset[:,0:2] # Splitting inputs from normalized data
output = scaled_dataset[:,-1] # Splitting output from normalized data
# Splitting the data into training data and testing data using train_test_split
```

```
from sklearn.model_selection import train_test_split
inputs_train,inputs_test,output_train,output_test = train_test_split(inputs,output,train_size=0.7,random_state=0) # train_size=0.7
```

Creating ANN model

In [ ]:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential() # Creating ANN model
model.add(Dense(64,input_dim=2,activation='relu')) # First layer with 18 nodes and two inputs and "relu" as activation function
model.add(Dense(16,activation='relu')) # Second layer with 12 nodes and "relu" as activation function
model.add(Dense(1,activation='linear')) # Output layer with one output and "linear" as activation function
```

Training the model using "adam"

In [ ]:

```
model.compile(optimizer ='adam',loss='MSE') # Compiling using optimizer "adam" and loss function "MSE"
history = model.fit(inputs_train,output_train,epochs=100,validation_split=0.15) # Training data with 100 epochs and validation split 0.15
```

```
Epoch 1/100
40/40 [==============================] - 1s 4ms/step - loss: 0.0219 - val_loss: 0.0211
Epoch 2/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0174 - val_loss: 0.0189
Epoch 3/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0159 - val_loss: 0.0177
Epoch 4/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0149 - val_loss: 0.0168
Epoch 5/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0140 - val_loss: 0.0159
Epoch 6/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0131 - val_loss: 0.0145
Epoch 7/100
40/40 [==============================] - 0s 4ms/step - loss: 0.0118 - val_loss: 0.0134
Epoch 8/100
40/40 [==============================] - 0s 5ms/step - loss: 0.0110 - val_loss: 0.0117
Epoch 9/100
40/40 [==============================] - 0s 4ms/step - loss: 0.0099 - val_loss: 0.0112
Epoch 10/100
40/40 [==============================] - 0s 5ms/step - loss: 0.0092 - val_loss: 0.0103
Epoch 11/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0087 - val_loss: 0.0097
Epoch 12/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0082 - val_loss: 0.0092
Epoch 13/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0077 - val_loss: 0.0089
Epoch 14/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0073 - val_loss: 0.0084
Epoch 15/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0070 - val_loss: 0.0080
Epoch 16/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0066 - val_loss: 0.0079
Epoch 17/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0064 - val_loss: 0.0074
Epoch 18/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0061 - val_loss: 0.0072
Epoch 19/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0061 - val_loss: 0.0066
Epoch 20/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0054 - val_loss: 0.0064
Epoch 21/100
40/40 [==============================] - 0s 5ms/step - loss: 0.0052 - val_loss: 0.0063
Epoch 22/100
40/40 [==============================] - 0s 6ms/step - loss: 0.0050 - val_loss: 0.0071
```

```
Epoch 23/100
40/40 [==============================] - 0s 4ms/step - loss: 0.0049 - val_loss: 0.0057
Epoch 24/100
40/40 [==============================] - 0s 4ms/step - loss: 0.0046 - val_loss: 0.0060
Epoch 25/100
40/40 [==============================] - 0s 5ms/step - loss: 0.0042 - val_loss: 0.0056
Epoch 26/100
40/40 [==============================] - 0s 5ms/step - loss: 0.0043 - val_loss: 0.0052
Epoch 27/100
40/40 [==============================] - 0s 5ms/step - loss: 0.0041 - val_loss: 0.0048
Epoch 28/100
40/40 [==============================] - 0s 5ms/step - loss: 0.0039 - val_loss: 0.0046
Epoch 29/100
40/40 [==============================] - 0s 4ms/step - loss: 0.0037 - val_loss: 0.0045
Epoch 30/100
40/40 [==============================] - 0s 3ms/step - loss: 0.0037 - val_loss: 0.0043
Epoch 31/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0038 - val_loss: 0.0043
Epoch 32/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0035 - val_loss: 0.0041
Epoch 33/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0032 - val_loss: 0.0041
Epoch 34/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0031 - val_loss: 0.0037
Epoch 35/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0029 - val_loss: 0.0037
Epoch 36/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0029 - val_loss: 0.0036
Epoch 37/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0028 - val_loss: 0.0036
Epoch 38/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0029 - val_loss: 0.0035
Epoch 39/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0028 - val_loss: 0.0033
Epoch 40/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0026 - val_loss: 0.0030
Epoch 41/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0023 - val_loss: 0.0030
Epoch 42/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0025 - val_loss: 0.0032
Epoch 43/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0025 - val_loss: 0.0028
Epoch 44/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0023 - val_loss: 0.0029
```

```
Epoch 45/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0021 - val_loss: 0.0030
Epoch 46/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0022 - val_loss: 0.0026
Epoch 47/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0020 - val_loss: 0.0029
Epoch 48/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0020 - val_loss: 0.0028
Epoch 49/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0019 - val_loss: 0.0026
Epoch 50/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0018 - val_loss: 0.0024
Epoch 51/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0018 - val_loss: 0.0025
Epoch 52/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0018 - val_loss: 0.0021
Epoch 53/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0018 - val_loss: 0.0020
Epoch 54/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0017 - val_loss: 0.0020
Epoch 55/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0016 - val_loss: 0.0020
Epoch 56/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0022 - val_loss: 0.0020
Epoch 57/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0016 - val_loss: 0.0019
Epoch 58/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0015 - val_loss: 0.0019
Epoch 59/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0015 - val_loss: 0.0018
Epoch 60/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0014 - val_loss: 0.0017
Epoch 61/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0014 - val_loss: 0.0016
Epoch 62/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0013 - val_loss: 0.0018
Epoch 63/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0013 - val_loss: 0.0015
Epoch 64/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0012 - val_loss: 0.0018
Epoch 65/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0014 - val_loss: 0.0017
Epoch 66/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0011 - val_loss: 0.0014
```

```
Epoch 67/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0012 - val_loss: 0.0016
Epoch 68/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0011 - val_loss: 0.0018
Epoch 69/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0016 - val_loss: 0.0013
Epoch 70/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0011 - val_loss: 0.0014
Epoch 71/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0011 - val_loss: 0.0012
Epoch 72/100
40/40 [==============================] - 0s 2ms/step - loss: 0.0010 - val_loss: 0.0014
Epoch 73/100
40/40 [==============================] - 0s 6ms/step - loss: 0.0010 - val_loss: 0.0013
Epoch 74/100
40/40 [==============================] - 0s 5ms/step - loss: 0.0011 - val_loss: 0.0011
Epoch 75/100
40/40 [==============================] - 0s 6ms/step - loss: 9.1982e-04 - val_loss: 0.0011
Epoch 76/100
40/40 [==============================] - 0s 5ms/step - loss: 8.8811e-04 - val_loss: 0.0011
Epoch 77/100
40/40 [==============================] - 0s 2ms/step - loss: 8.8763e-04 - val_loss: 0.0012
Epoch 78/100
40/40 [==============================] - 0s 2ms/step - loss: 9.0482e-04 - val_loss: 9.4236e-04
Epoch 79/100
40/40 [==============================] - 0s 4ms/step - loss: 9.2058e-04 - val_loss: 0.0012
Epoch 80/100
40/40 [==============================] - 0s 6ms/step - loss: 8.2582e-04 - val_loss: 0.0010
Epoch 81/100
40/40 [==============================] - 0s 3ms/step - loss: 8.0485e-04 - val_loss: 9.4995e-04
Epoch 82/100
40/40 [==============================] - 0s 3ms/step - loss: 8.1076e-04 - val_loss: 9.1897e-04
Epoch 83/100
40/40 [==============================] - 0s 5ms/step - loss: 8.0194e-04 - val_loss: 8.4098e-04
Epoch 84/100
40/40 [==============================] - 0s 5ms/step - loss: 7.9463e-04 - val_loss: 0.0012
Epoch 85/100
40/40 [==============================] - 0s 3ms/step - loss: 7.5274e-04 - val_loss: 8.9879e-04
Epoch 86/100
40/40 [==============================] - 0s 3ms/step - loss: 7.9996e-04 - val_loss: 9.3851e-04
Epoch 87/100
40/40 [==============================] - 0s 3ms/step - loss: 8.4681e-04 - val_loss: 9.0067e-04
Epoch 88/100
40/40 [==============================] - 0s 2ms/step - loss: 8.6534e-04 - val_loss: 0.0012
```

```
Epoch 89/100
40/40 [==============================] - 0s 2ms/step - loss: 8.3141e-04 - val_loss: 8.7885e-04
Epoch 90/100
40/40 [==============================] - 0s 2ms/step - loss: 9.2267e-04 - val_loss: 9.9132e-04
Epoch 91/100
40/40 [==============================] - 0s 2ms/step - loss: 6.5781e-04 - val_loss: 8.9673e-04
Epoch 92/100
40/40 [==============================] - 0s 2ms/step - loss: 7.7507e-04 - val_loss: 7.8268e-04
Epoch 93/100
40/40 [==============================] - 0s 2ms/step - loss: 6.7644e-04 - val_loss: 6.4784e-04
Epoch 94/100
40/40 [==============================] - 0s 2ms/step - loss: 6.7133e-04 - val_loss: 0.0012
Epoch 95/100
40/40 [==============================] - 0s 2ms/step - loss: 7.9216e-04 - val_loss: 8.2156e-04
Epoch 96/100
40/40 [==============================] - 0s 2ms/step - loss: 5.8391e-04 - val_loss: 6.8428e-04
Epoch 97/100
40/40 [==============================] - 0s 2ms/step - loss: 5.8680e-04 - val_loss: 6.6805e-04
Epoch 98/100
40/40 [==============================] - 0s 2ms/step - loss: 5.7190e-04 - val_loss: 7.3684e-04
Epoch 99/100
40/40 [==============================] - 0s 2ms/step - loss: 6.5567e-04 - val_loss: 7.5395e-04
Epoch 100/100
40/40 [==============================] - 0s 2ms/step - loss: 6.0951e-04 - val_loss: 5.5630e-04
```

Prediction and analysis

In [ ]:

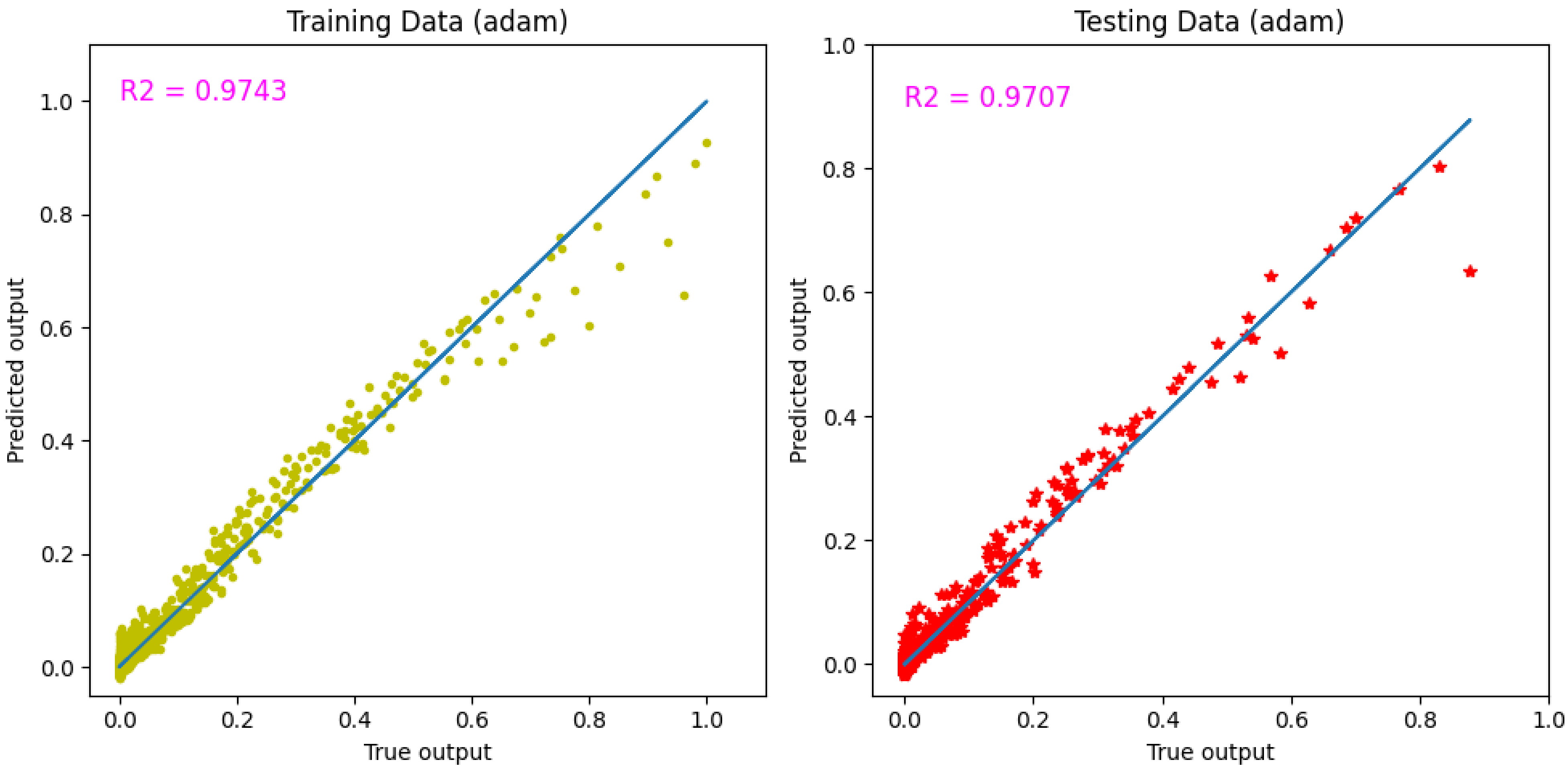
```
pred_train = model.predict(inputs_train) # Prediction of training data
pred_test = model.predict(inputs_test) # Prediction of testing data
from sklearn.metrics import r2_score
r2_train = r2_score(output_train,pred_train) # R2 score of training data
print(f"R2 Score for training data: {r2_train:.4f}")
r2_test = r2_score(output_test,pred_test) # R2 score of testing data
print(f"R2 Score for testing data: {r2_test:.4f}")
from sklearn.metrics import mean_squared_error
mse_train = mean_squared_error(output_train,pred_train) # MSE of training data
print(f"Mean Squared Error for training data: {mse_train:.4f}")
mse_test = mean_squared_error(output_test,pred_test) # MSE of Testing data
print(f"Mean Squared Error for testing data: {mse_test:.4f}")
```

```
47/47 [==============================] - 0s 953us/step
20/20 [==============================] - 0s 1ms/step
R2 Score for training data: 0.9743
R2 Score for testing data: 0.9707
Mean Squared Error for training data: 0.0005
Mean Squared Error for testing data: 0.0004
```

Plots - Training Data Vs Testing Data

In [ ]:

```
import matplotlib.pyplot as plt
fig,plot = plt.subplots(1,2,figsize=(10,5))
plot[0].plot(output_train,pred_train,'y.')
plot[0].plot(output_train,output_train,'-')
plot[0].set_xlabel('True output')
plot[0].set_ylabel('Predicted output')
plot[0].set_ylim(-0.05,1.1)
plot[0].set_xlim(-0.05,1.1)
plot[0].set_title(f'Training Data (adam)')
plot[0].text(0,1,f'R2 = {r2_train:.4f}',fontsize=12,color='magenta')
plot[1].plot(output_test,pred_test,'r*')
plot[1].plot(output_test,output_test,'-')
plot[1].set_ylim(-0.05,1)
plot[1].set_xlim(-0.05,1)
plot[1].set_xlabel('True output')
plot[1].set_ylabel('Predicted output')
plot[1].set_title(f'Testing Data (adam)')
plot[1].text(0,0.9,f'R2 = {r2_test:.4f}',fontsize=12,color='magenta')
plt.tight_layout()
plt.show()
```



Training the model using "RMSProp"

In [ ]:
```
model.compile(optimizer ='RMSProp',loss='MSE') # Compiling using optimizer "RMSProp" and loss function "MSE"
history = model.fit(inputs_train,output_train,epochs=100,validation_split=0.15) # Training data with 100 epochs and validation split 0.15
```

```
Epoch 1/100
40/40 [==============================] - 1s 5ms/step - loss: 9.8247e-04 - val_loss: 7.2633e-04
Epoch 2/100
40/40 [==============================] - 0s 3ms/step - loss: 7.3786e-04 - val_loss: 6.0421e-04
Epoch 3/100
40/40 [==============================] - 0s 3ms/step - loss: 8.0547e-04 - val_loss: 6.8151e-04
Epoch 4/100
40/40 [==============================] - 0s 2ms/step - loss: 8.5381e-04 - val_loss: 7.8488e-04
Epoch 5/100
40/40 [==============================] - 0s 2ms/step - loss: 7.1486e-04 - val_loss: 7.4644e-04
Epoch 6/100
40/40 [==============================] - 0s 3ms/step - loss: 9.3126e-04 - val_loss: 5.8971e-04
Epoch 7/100
40/40 [==============================] - 0s 3ms/step - loss: 6.7033e-04 - val_loss: 0.0015
Epoch 8/100
40/40 [==============================] - 0s 3ms/step - loss: 7.4424e-04 - val_loss: 5.7427e-04
Epoch 9/100
40/40 [==============================] - 0s 3ms/step - loss: 7.5734e-04 - val_loss: 0.0012
Epoch 10/100
40/40 [==============================] - 0s 3ms/step - loss: 7.7890e-04 - val_loss: 8.7224e-04
Epoch 11/100
40/40 [==============================] - 0s 3ms/step - loss: 7.8344e-04 - val_loss: 7.3564e-04
Epoch 12/100
40/40 [==============================] - 0s 3ms/step - loss: 6.7759e-04 - val_loss: 7.0305e-04
Epoch 13/100
40/40 [==============================] - 0s 3ms/step - loss: 6.9668e-04 - val_loss: 5.1650e-04
Epoch 14/100
40/40 [==============================] - 0s 2ms/step - loss: 7.8510e-04 - val_loss: 6.1793e-04
Epoch 15/100
40/40 [==============================] - 0s 3ms/step - loss: 7.0310e-04 - val_loss: 5.6028e-04
Epoch 16/100
40/40 [==============================] - 0s 3ms/step - loss: 6.6520e-04 - val_loss: 6.0850e-04
Epoch 17/100
40/40 [==============================] - 0s 3ms/step - loss: 8.3390e-04 - val_loss: 6.0670e-04
Epoch 18/100
40/40 [==============================] - 0s 3ms/step - loss: 6.4078e-04 - val_loss: 6.2199e-04
Epoch 19/100
40/40 [==============================] - 0s 2ms/step - loss: 8.1247e-04 - val_loss: 9.6612e-04
Epoch 20/100
40/40 [==============================] - 0s 2ms/step - loss: 6.9862e-04 - val_loss: 6.1816e-04
Epoch 21/100
40/40 [==============================] - 0s 3ms/step - loss: 6.1649e-04 - val_loss: 9.5602e-04
Epoch 22/100
40/40 [==============================] - 0s 2ms/step - loss: 7.2240e-04 - val_loss: 7.5206e-04
```

```
Epoch 23/100
40/40 [==============================] - 0s 2ms/step - loss: 6.8571e-04 - val_loss: 4.8839e-04
Epoch 24/100
40/40 [==============================] - 0s 6ms/step - loss: 7.0508e-04 - val_loss: 0.0010
Epoch 25/100
40/40 [==============================] - 0s 7ms/step - loss: 7.3157e-04 - val_loss: 9.4220e-04
Epoch 26/100
40/40 [==============================] - 0s 6ms/step - loss: 6.5926e-04 - val_loss: 0.0019
Epoch 27/100
40/40 [==============================] - 0s 5ms/step - loss: 7.3397e-04 - val_loss: 6.1705e-04
Epoch 28/100
40/40 [==============================] - 0s 5ms/step - loss: 6.5159e-04 - val_loss: 6.2865e-04
Epoch 29/100
40/40 [==============================] - 0s 4ms/step - loss: 5.7743e-04 - val_loss: 0.0012
Epoch 30/100
40/40 [==============================] - 0s 3ms/step - loss: 7.1798e-04 - val_loss: 5.9112e-04
Epoch 31/100
40/40 [==============================] - 0s 2ms/step - loss: 5.5622e-04 - val_loss: 4.5787e-04
Epoch 32/100
40/40 [==============================] - 0s 3ms/step - loss: 5.9665e-04 - val_loss: 7.4530e-04
Epoch 33/100
40/40 [==============================] - 0s 2ms/step - loss: 6.3976e-04 - val_loss: 4.8001e-04
Epoch 34/100
40/40 [==============================] - 0s 2ms/step - loss: 7.1547e-04 - val_loss: 5.3833e-04
Epoch 35/100
40/40 [==============================] - 0s 2ms/step - loss: 7.2383e-04 - val_loss: 4.4626e-04
Epoch 36/100
40/40 [==============================] - 0s 3ms/step - loss: 6.6559e-04 - val_loss: 5.7386e-04
Epoch 37/100
40/40 [==============================] - 0s 3ms/step - loss: 5.8595e-04 - val_loss: 8.1571e-04
Epoch 38/100
40/40 [==============================] - 0s 3ms/step - loss: 7.2085e-04 - val_loss: 6.1580e-04
Epoch 39/100
40/40 [==============================] - 0s 3ms/step - loss: 6.0390e-04 - val_loss: 8.6091e-04
Epoch 40/100
40/40 [==============================] - 0s 3ms/step - loss: 5.4755e-04 - val_loss: 0.0014
Epoch 41/100
40/40 [==============================] - 0s 2ms/step - loss: 6.8436e-04 - val_loss: 4.4381e-04
Epoch 42/100
40/40 [==============================] - 0s 2ms/step - loss: 5.5805e-04 - val_loss: 3.6942e-04
Epoch 43/100
40/40 [==============================] - 0s 2ms/step - loss: 5.5330e-04 - val_loss: 0.0016
Epoch 44/100
40/40 [==============================] - 0s 2ms/step - loss: 7.1708e-04 - val_loss: 4.3638e-04
```

```
Epoch 45/100
40/40 [==============================] - 0s 2ms/step - loss: 5.5806e-04 - val_loss: 3.7942e-04
Epoch 46/100
40/40 [==============================] - 0s 2ms/step - loss: 5.8232e-04 - val_loss: 5.0220e-04
Epoch 47/100
40/40 [==============================] - 0s 2ms/step - loss: 7.4984e-04 - val_loss: 4.9273e-04
Epoch 48/100
40/40 [==============================] - 0s 3ms/step - loss: 5.0059e-04 - val_loss: 0.0015
Epoch 49/100
40/40 [==============================] - 0s 2ms/step - loss: 5.6793e-04 - val_loss: 3.4716e-04
Epoch 50/100
40/40 [==============================] - 0s 2ms/step - loss: 6.2797e-04 - val_loss: 9.4271e-04
Epoch 51/100
40/40 [==============================] - 0s 2ms/step - loss: 5.7364e-04 - val_loss: 3.4806e-04
Epoch 52/100
40/40 [==============================] - 0s 3ms/step - loss: 5.9524e-04 - val_loss: 5.4416e-04
Epoch 53/100
40/40 [==============================] - 0s 2ms/step - loss: 6.7287e-04 - val_loss: 3.5731e-04
Epoch 54/100
40/40 [==============================] - 0s 2ms/step - loss: 5.0404e-04 - val_loss: 4.6464e-04
Epoch 55/100
40/40 [==============================] - 0s 2ms/step - loss: 5.4682e-04 - val_loss: 3.1691e-04
Epoch 56/100
40/40 [==============================] - 0s 2ms/step - loss: 6.4455e-04 - val_loss: 3.5907e-04
Epoch 57/100
40/40 [==============================] - 0s 2ms/step - loss: 5.5048e-04 - val_loss: 0.0020
Epoch 58/100
40/40 [==============================] - 0s 2ms/step - loss: 6.0459e-04 - val_loss: 5.8487e-04
Epoch 59/100
40/40 [==============================] - 0s 2ms/step - loss: 6.5632e-04 - val_loss: 3.3193e-04
Epoch 60/100
40/40 [==============================] - 0s 3ms/step - loss: 5.1303e-04 - val_loss: 3.9312e-04
Epoch 61/100
40/40 [==============================] - 0s 3ms/step - loss: 6.2424e-04 - val_loss: 3.7338e-04
Epoch 62/100
40/40 [==============================] - 0s 2ms/step - loss: 4.4642e-04 - val_loss: 9.9579e-04
Epoch 63/100
40/40 [==============================] - 0s 3ms/step - loss: 4.8279e-04 - val_loss: 0.0014
Epoch 64/100
40/40 [==============================] - 0s 2ms/step - loss: 6.3462e-04 - val_loss: 4.3421e-04
Epoch 65/100
40/40 [==============================] - 0s 3ms/step - loss: 6.4278e-04 - val_loss: 8.8773e-04
Epoch 66/100
40/40 [==============================] - 0s 3ms/step - loss: 4.4862e-04 - val_loss: 6.7756e-04
```

```
Epoch 67/100
40/40 [==============================] - 0s 3ms/step - loss: 5.9932e-04 - val_loss: 0.0014
Epoch 68/100
40/40 [==============================] - 0s 3ms/step - loss: 5.1777e-04 - val_loss: 7.8456e-04
Epoch 69/100
40/40 [==============================] - 0s 3ms/step - loss: 6.5426e-04 - val_loss: 3.3369e-04
Epoch 70/100
40/40 [==============================] - 0s 3ms/step - loss: 4.4523e-04 - val_loss: 0.0030
Epoch 71/100
40/40 [==============================] - 0s 2ms/step - loss: 5.7540e-04 - val_loss: 5.6186e-04
Epoch 72/100
40/40 [==============================] - 0s 2ms/step - loss: 5.5314e-04 - val_loss: 0.0018
Epoch 73/100
40/40 [==============================] - 0s 2ms/step - loss: 6.5414e-04 - val_loss: 3.4378e-04
Epoch 74/100
40/40 [==============================] - 0s 4ms/step - loss: 5.0991e-04 - val_loss: 2.9459e-04
Epoch 75/100
40/40 [==============================] - 0s 5ms/step - loss: 5.3810e-04 - val_loss: 0.0022
Epoch 76/100
40/40 [==============================] - 0s 4ms/step - loss: 5.4391e-04 - val_loss: 9.1766e-04
Epoch 77/100
40/40 [==============================] - 0s 3ms/step - loss: 5.6862e-04 - val_loss: 4.5353e-04
Epoch 78/100
40/40 [==============================] - 0s 5ms/step - loss: 5.7171e-04 - val_loss: 3.3533e-04
Epoch 79/100
40/40 [==============================] - 0s 2ms/step - loss: 4.7167e-04 - val_loss: 3.4093e-04
Epoch 80/100
40/40 [==============================] - 0s 2ms/step - loss: 5.5266e-04 - val_loss: 8.6667e-04
Epoch 81/100
40/40 [==============================] - 0s 2ms/step - loss: 5.1395e-04 - val_loss: 2.6468e-04
Epoch 82/100
40/40 [==============================] - 0s 2ms/step - loss: 5.0624e-04 - val_loss: 0.0031
Epoch 83/100
40/40 [==============================] - 0s 2ms/step - loss: 5.2743e-04 - val_loss: 0.0023
Epoch 84/100
40/40 [==============================] - 0s 2ms/step - loss: 5.2633e-04 - val_loss: 5.7535e-04
Epoch 85/100
40/40 [==============================] - 0s 1ms/step - loss: 4.8765e-04 - val_loss: 4.9348e-04
Epoch 86/100
40/40 [==============================] - 0s 2ms/step - loss: 5.3023e-04 - val_loss: 5.5799e-04
Epoch 87/100
40/40 [==============================] - 0s 2ms/step - loss: 6.1872e-04 - val_loss: 4.2876e-04
Epoch 88/100
40/40 [==============================] - 0s 2ms/step - loss: 5.2590e-04 - val_loss: 3.4512e-04
```

```
Epoch 89/100
40/40 [==============================] - 0s 2ms/step - loss: 5.1216e-04 - val_loss: 2.9249e-04
Epoch 90/100
40/40 [==============================] - 0s 3ms/step - loss: 4.3112e-04 - val_loss: 4.7897e-04
Epoch 91/100
40/40 [==============================] - 0s 5ms/step - loss: 5.9990e-04 - val_loss: 5.4168e-04
Epoch 92/100
40/40 [==============================] - 0s 4ms/step - loss: 4.1397e-04 - val_loss: 0.0024
Epoch 93/100
40/40 [==============================] - 0s 2ms/step - loss: 5.2461e-04 - val_loss: 5.6938e-04
Epoch 94/100
40/40 [==============================] - 0s 2ms/step - loss: 4.3974e-04 - val_loss: 3.2692e-04
Epoch 95/100
40/40 [==============================] - 0s 2ms/step - loss: 5.0344e-04 - val_loss: 5.2320e-04
Epoch 96/100
40/40 [==============================] - 0s 2ms/step - loss: 4.5833e-04 - val_loss: 8.4537e-04
Epoch 97/100
40/40 [==============================] - 0s 2ms/step - loss: 5.0425e-04 - val_loss: 2.7114e-04
Epoch 98/100
40/40 [==============================] - 0s 5ms/step - loss: 4.6044e-04 - val_loss: 8.0048e-04
Epoch 99/100
40/40 [==============================] - 0s 2ms/step - loss: 4.5768e-04 - val_loss: 5.5124e-04
Epoch 100/100
40/40 [==============================] - 0s 2ms/step - loss: 4.5367e-04 - val_loss: 5.9429e-04
```

Prediction and analysis

In [ ]:

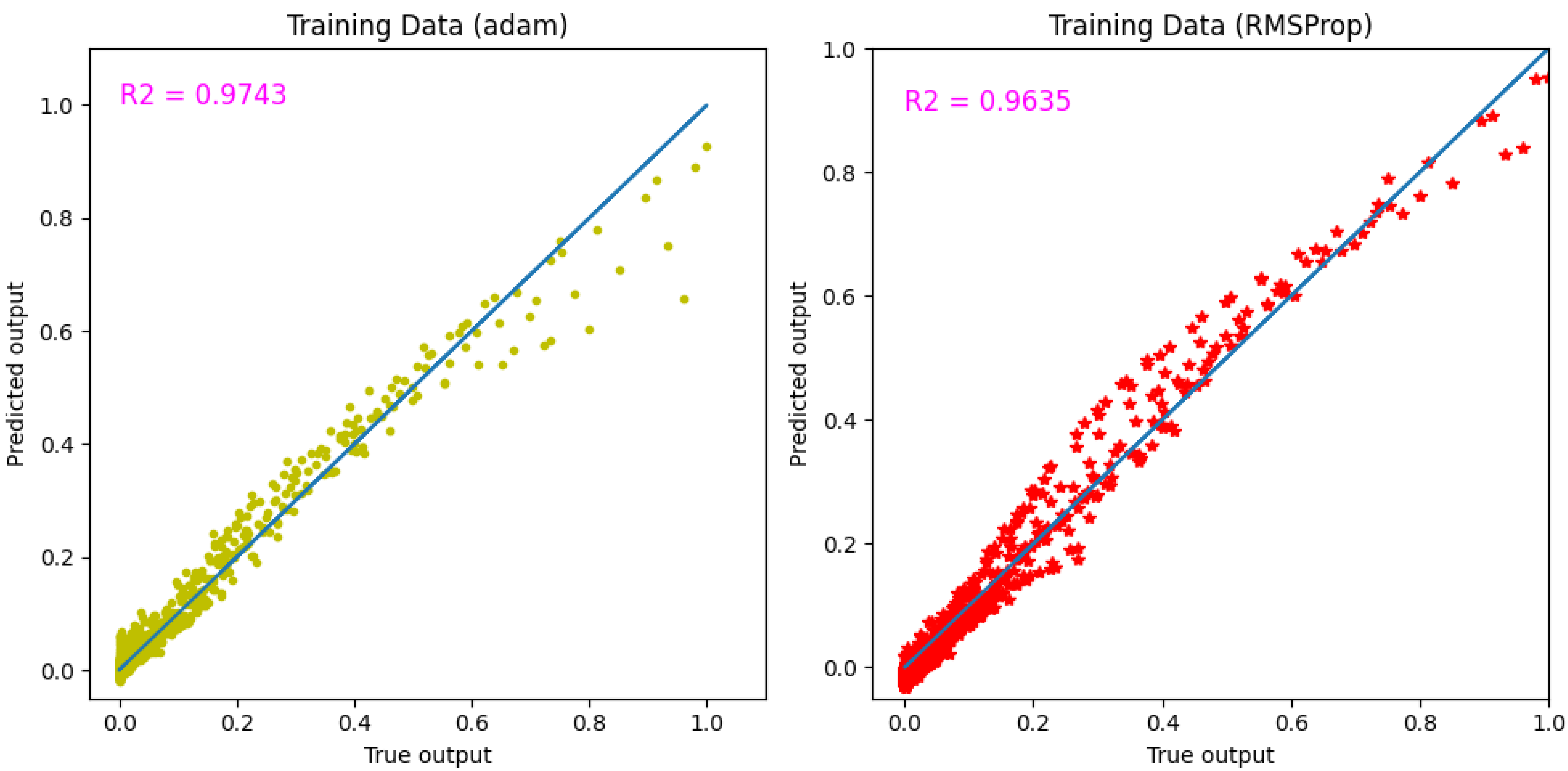
```
pred_train2 = model.predict(inputs_train) # Prediction of training data
pred_test2 = model.predict(inputs_test) # Prediction of testing data
from sklearn.metrics import r2_score
r2_train2 = r2_score(output_train,pred_train2) # R2 score of training data
print(f"R2 Score for training data: {r2_train2:.4f}")
r2_test2 = r2_score(output_test,pred_test2) # R2 score of testing data
print(f"R2 Score for testing data: {r2_test2:.4f}")
from sklearn.metrics import mean_squared_error
mse_train2 = mean_squared_error(output_train,pred_train2) # MSE of training data
print(f"Mean Squared Error for training data: {mse_train:.4f}")
mse_test2 = mean_squared_error(output_test,pred_test2) # MSE of testing data
print(f"Mean Squared Error for testing data: {mse_test:.4f}")
```

```
47/47 [==============================] - 0s 739us/step
20/20 [==============================] - 0s 1ms/step
R2 Score for training data: 0.9635
R2 Score for testing data: 0.9568
Mean Squared Error for training data: 0.0005
Mean Squared Error for testing data: 0.0004
```

Plot for Training data "adam" Vs "RMSProp"

In [ ]:

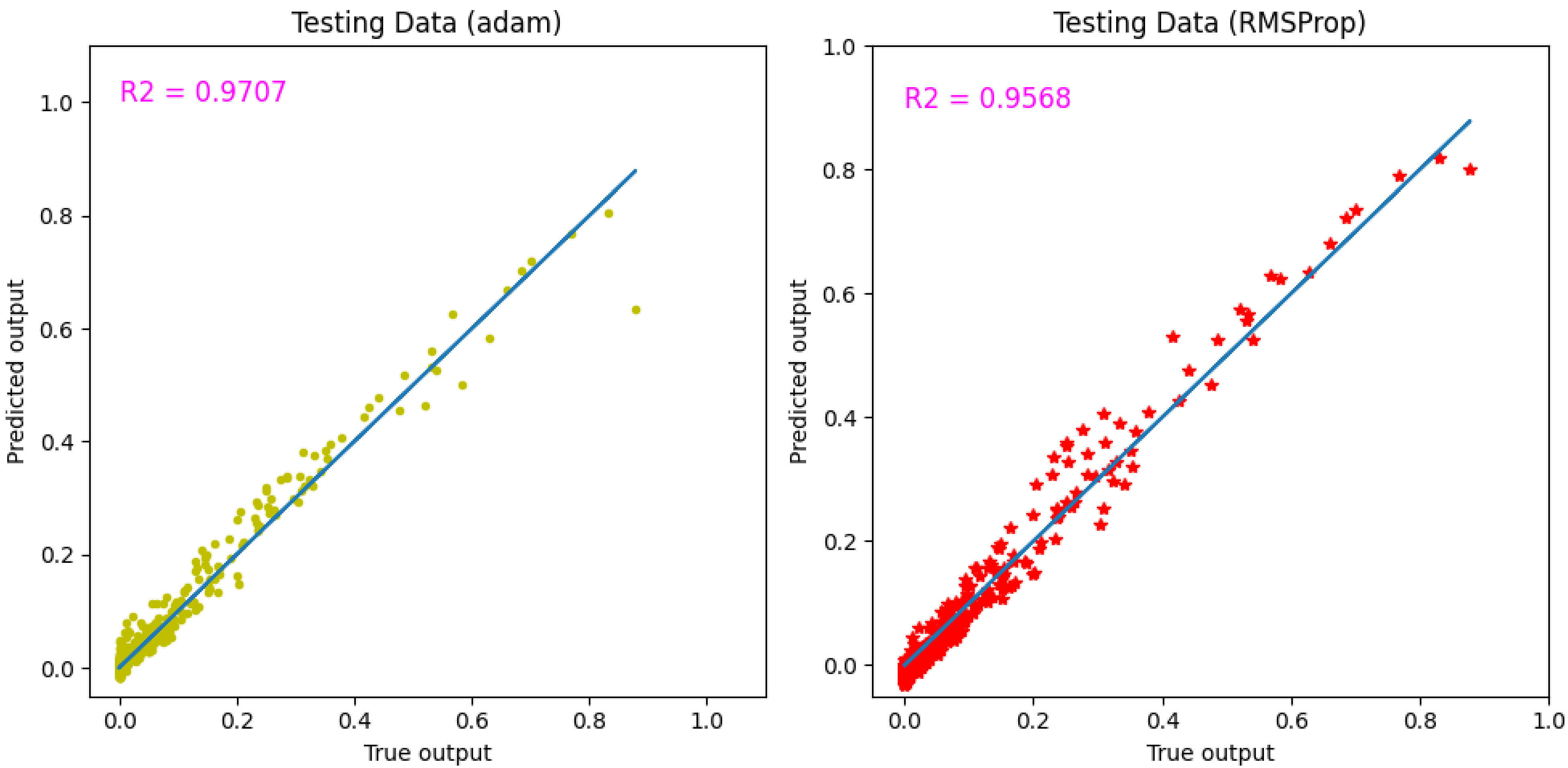
```
# Plot for Training data "adam" Vs "RMSProp"
import matplotlib.pyplot as plt
fig,plot2 = plt.subplots(1,2,figsize=(10,5))
plot2[0].plot(output_train,pred_train,'y.')
plot2[0].plot(output_train,output_train,'-')
plot2[0].set_xlabel('True output')
plot2[0].set_ylabel('Predicted output')
plot2[0].set_ylim(-0.05,1.1)
plot2[0].set_xlim(-0.05,1.1)
plot2[0].set_title(f'Training Data (adam)')
plot2[0].text(0,1,f'R2 = {r2_train:.4f}',fontsize=12,color='magenta')
plot2[1].plot(output_train,pred_train2,'r*')
plot2[1].plot(output_train,output_train,'-')
plot2[1].set_ylim(-0.05,1)
plot2[1].set_xlim(-0.05,1)
plot2[1].set_xlabel('True output')
plot2[1].set_ylabel('Predicted output')
plot2[1].set_title(f'Training Data (RMSProp)')
plot2[1].text(0,0.9,f'R2 = {r2_train2:.4f}',fontsize=12,color='magenta')
plt.tight_layout()
plt.show()
```



Plot for Testing data "adam" Vs "RMSProp"

In [ ]:
```
# Plot for Testing data "adam" Vs "RMSProp"
import matplotlib.pyplot as plt
fig,plot3 = plt.subplots(1,2,figsize=(10,5))
plot3[0].plot(output_test,pred_test,'y.')
plot3[0].plot(output_test,output_test,'-')
plot3[0].set_xlabel('True output')
plot3[0].set_ylabel('Predicted output')
plot3[0].set_ylim(-0.05,1.1)
plot3[0].set_xlim(-0.05,1.1)
plot3[0].set_title(f'Testing Data (adam)')
plot3[0].text(0,1,f'R2 = {r2_test:.4f}',fontsize=12,color='magenta')
plot3[1].plot(output_test,pred_test2,'r*')
```

```python
plot3[1].plot(output_test,output_test,'-')
plot3[1].set_ylim(-0.05,1)
plot3[1].set_xlim(-0.05,1)
plot3[1].set_xlabel('True output')
plot3[1].set_ylabel('Predicted output')
plot3[1].set_title(f'Testing Data (RMSProp)')
plot3[1].text(0,0.9,f'R2 = {r2_test2:.4f}',fontsize=12,color='magenta')
plt.tight_layout()
plt.show()
```



Effect of variations

In [ ]:
```python
# Define Beale function
def beale_function(x,y):
    """
    Equation of beale function

    Inputs:
    x and y values

    Outputs:
    Value of beale function at x and y
    """
    return (1.5-x+x*y)**2 + (2.25-x+x*y**2)**2 + (2.625-x+x*y**3)**2

# Generating data for beale function in range and saving to excel
def generating_data(step,low=-4.5,high=4.6):
    """
    Generates data points on the beale function for all values of x and y in the range -4.5 < x,y < 4.5

    Inputs:
    1) step

    Output:
    1) DataFrame containing values of (x,y,z)
    """
    X,Y,Z = [],[],[]
    for i in np.arange(low,high,step):
        for j in np.arange(low,high,step):
            X.append(i)
            Y.append(j)
            Z.append(beale_function(i,j))
    # Creating a dataframe for the data generated above
    data1 = zip(X,Y,Z)
    data2 = pd.DataFrame(data1,columns=['x','y','z'])
    return data2

# Normarlizing, splitting into inputs and outputs, test splitting
def split(data,train_size:float):
    """
    This function takes the data, normalize it and splits into training dataset and testing dataset

    Input:
    1) data (DataFrame): DataFrame containing labelled data
    2) train_size (float): Fraction of train sample to be split
```

```python
    Output:
    1) inputs_train: input datasets for training
    2) inputs_test: input datasets for testing
    2) output_test: output datasets for testing
    4) output_test: output datasets for training
    """
    # Normallizing the data
    scaler = MinMaxScaler()
    data3 = scaler.fit_transform(data.values)
    # Splitting the data
    inputs = data3[:,0:2]
    output = data3[:,-1]
    # Splitting the data
    inputs_train,inputs_test,output_train,output_test = train_test_split(inputs,output,train_size=train_size,random_state=0)
    return inputs_train,inputs_test,output_train,output_test

# Structuring ANN with number of hidden layers, nodes and activation functions
def network(layers:int,nodes:list,activation_function:str):
    """"
    Creates a Artificial Neural Network with linear output layer with one node

    Inputs:
    1) layers (int): Number of layers in the neural network
    2) nodes (list): Number of nodes present in each hidden layer starting from first hidden layer
    3) activation_function (str): Optimizer for the hidden layers

    Output:
    1) model: Gives you the nodel as output
    """
    model = Sequential()
    # Add hidden layers
    for i in range(layers):
        if i == 0:
            # For the first hidden layer
            model.add(Dense(nodes[i], input_dim=2, activation=activation_function))
        else:
            model.add(Dense(nodes[i], activation=activation_function))
    # Output layer
    model.add(Dense(1,activation='linear'))
    # model.summary()
    return model

# Training
```

```python
def training(model,optimizer:str,loss_function:str,inputs,output,epoch=100,):
    """
    Trains the data given to the model with given optimizer and loss function with default epochs=100

    Inputs:
    1) model: A model to be trained
    2) optimizer (str): A optimizer for training
    3) loss_function (str): Loss function
    4) inputs: All inputs of function
    5) output: All output of function
    6) epoch (int): Number of epochs

    Output:
    1) model: Gives you trained model

    """
    model.compile(optimizer =optimizer,loss=loss_function)
    history = model.fit(inputs,output,epochs=epoch,validation_split=0.15,verbose=0)
    return model

# Results
def result(predicted,output):
    """
    Takes actual values and predicted values and gives you R2_score and MSE

    Inputs:
    1) predicted: Output that is predicted by model
    2) output: Actual output

    Output:
    1) r2 (float): R2_score of the given data
    2) mse (float): MSE of the given data
    """
    mse = mean_squared_error(output,predicted)
    print(f"Mean Squared Error: {mse:.4f}")
    # Calculating R2 score
    r2 = r2_score(output,predicted)
    print(f"R2 Score: {r2:.4f}")
    return r2,mse

# Plotting Training data vs Testing data
def plotting(splitted_data,train_prediction,test_prediction,tr,te,variation):
    """
    Plots graphs between Predicted output and Actual output for Training and Testing data
```

```python
    Inputs:
    1) splitted_data: Actual data
    2) train_prediction: Output predicted by model from training data
    3) test_prediction: Output predicted by model from  testing data
    4) tr: r2_score of Training data
    4) te: r2_score of Testing data
    5) variation (str): If plotting for multiple variations
    """
    fig,plot = plt.subplots(1,2,figsize=(10,5))
    plot[0].plot(splitted_data[2],train_prediction,'y.')
    plot[0].plot(splitted_data[2],splitted_data[2],'-')
    plot[0].set_xlabel('True output')
    plot[0].set_ylabel('Predicted output')
    plot[0].set_ylim(-0.05,1.1)
    plot[0].set_xlim(-0.05,1.1)
    plot[0].set_title(f'Training Data ({variation})')
    plot[0].text(0,1,f'R2 = {tr:.4f}',fontsize=12,color='magenta')
    plot[1].plot(splitted_data[3],test_prediction,'r*')
    plot[1].plot(splitted_data[3],splitted_data[3],'-')
    plot[1].set_ylim(-0.05,1)
    plot[1].set_xlim(-0.05,1)
    plot[1].set_xlabel('True output')
    plot[1].set_ylabel('Predicted output')
    plot[1].set_title(f'Testing Data ({variation})')
    plot[1].text(0,0.9,f'R2 = {te:.4f}',fontsize=12,color='magenta')
    plt.tight_layout()
    plt.show()

# Plotting variation
def plotting2(tr_r2,tr_mse,te_r2,te_mse,var=[1,2,3],iter='Combination'):
    """
    Plots two graphs for R2_score and MSE for different variations

    Inputs:
    1) tr_r2 (list): A list of all r2_score for training data for all variations
    2) tr_mse (list): A list of all MSE for training data for all variations
    3) te_r2 (list): A list of all r2_score for testing data for all variations
    4) te_mse (list): A list of all MSE for testing data for all variations
    5) var (list): A list of number of variations for x-axis
    6) iter (str): Name of the variable that is being studied
    """
    fig,plot = plt.subplots(1,2,figsize=(10,5))
    plot[0].plot(var,tr_r2,'b-',label='Training data')
```

```
    plot[0].plot(var,te_r2,'g-',label='Testing data')
    plot[0].set_xlabel(iter)
    plot[0].set_ylabel('R2_score')
    # plot[0].grid(True)
    plot[0].legend()
    plot[1].plot(var,tr_mse,'b-',label='Training data')
    plot[1].plot(var,te_mse,'g-',label='Testing data')
    plot[1].set_xlabel(iter)
    plot[1].set_ylabel('MSE')
    # plot[1].grid(True)
    plot[1].legend()
    plt.tight_layout()
    plt.show()
```

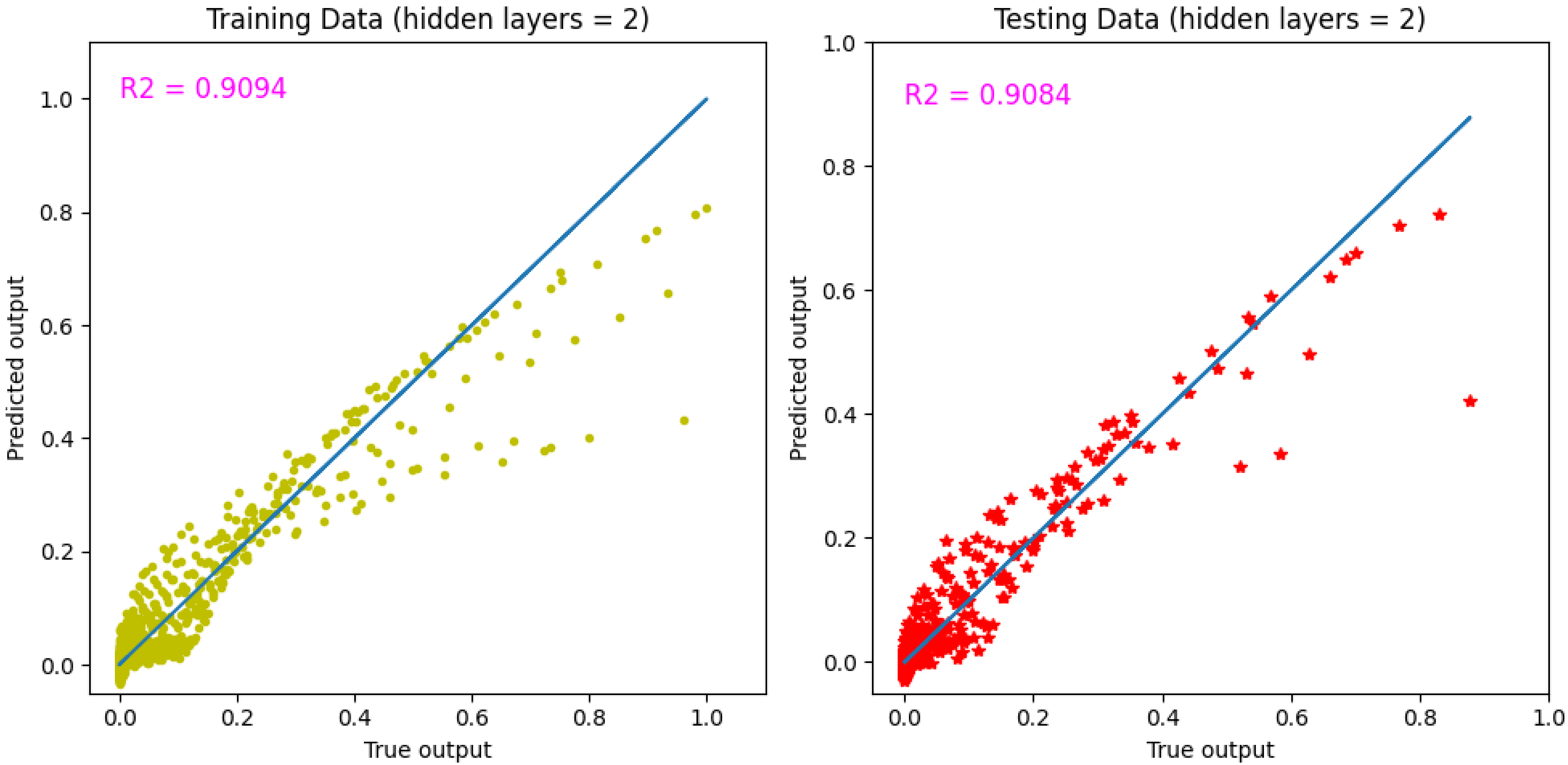
In [ ]:
```
splitted_data = split(data3,0.7) # Data for variations
```

In [ ]:
```
# Variation of parameters
ex_layers = [2,3,4] # Variation in nodes
ex_nodes = [18,12,9,6] # Variation in layers
ex_activation_functions = ['relu','softmax','tanh'] # Variations in activation functions of all hidden layers
ex_epochs = [50,100,150] # Variation in epochs
ex_train_size = [0.6,0.8,0.9] # Variation in Training size
```

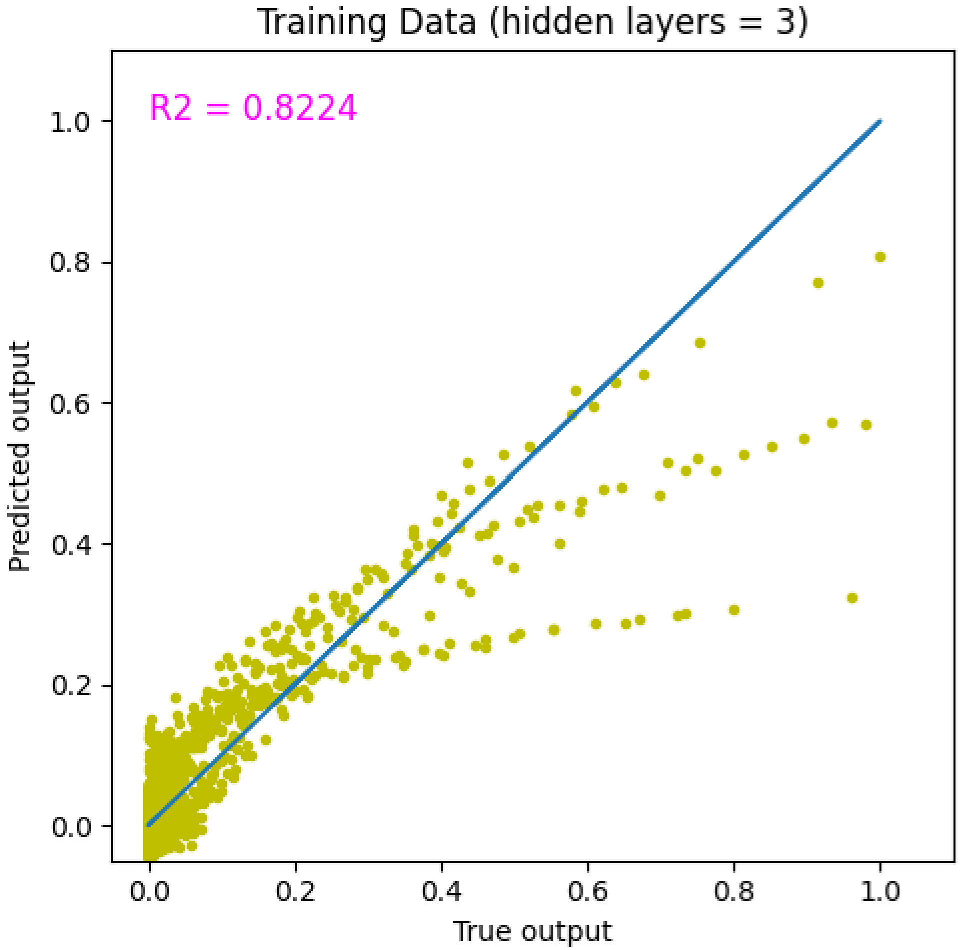
In [ ]:
```
# Varying number of hidden layers
tr_r71,tr_mse71,te_r71,te_mse71 = [],[],[],[]
for a in range(0,3,1):
    iter71 = f'hidden layers = {ex_layers[a]}'
    # Creating a network
    ann711 = network(ex_layers[a],ex_nodes,ex_activation_functions[0])
    # Training
    ann712 = training(ann711,'adam','MSE',splitted_data[0],splitted_data[2],ex_epochs[1])
    # Train data
    z71_train = ann712.predict(splitted_data[0],verbose=0)
    # Test data
    z71_test = ann712.predict(splitted_data[1],verbose=0)
    # Results
    r71,mse71 = result(z71_train,splitted_data[2])
    r711,mse711 = result(z71_test,splitted_data[3])
    tr_r71.append(r71)
    tr_mse71.append(mse71)
    te_r71.append(r711)
    te_mse71.append(mse711)
    #Plotting
```

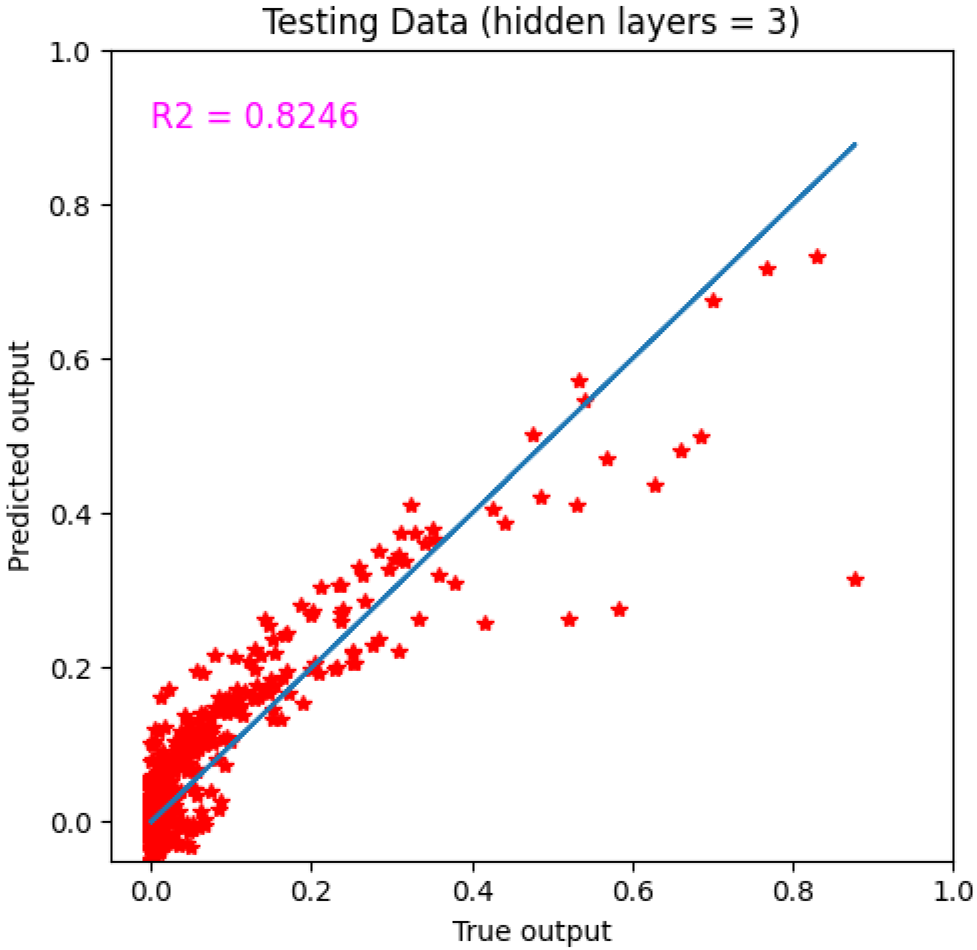
```
    plotting(splitted_data,z71_train,z71_test,r71,r711,iter71)
plotting2(tr_r71,tr_mse71,te_r71,te_mse71,ex_layers,'Layers')
```

```
Mean Squared Error: 0.0017
R2 Score: 0.9094
Mean Squared Error: 0.0013
R2 Score: 0.9084
```

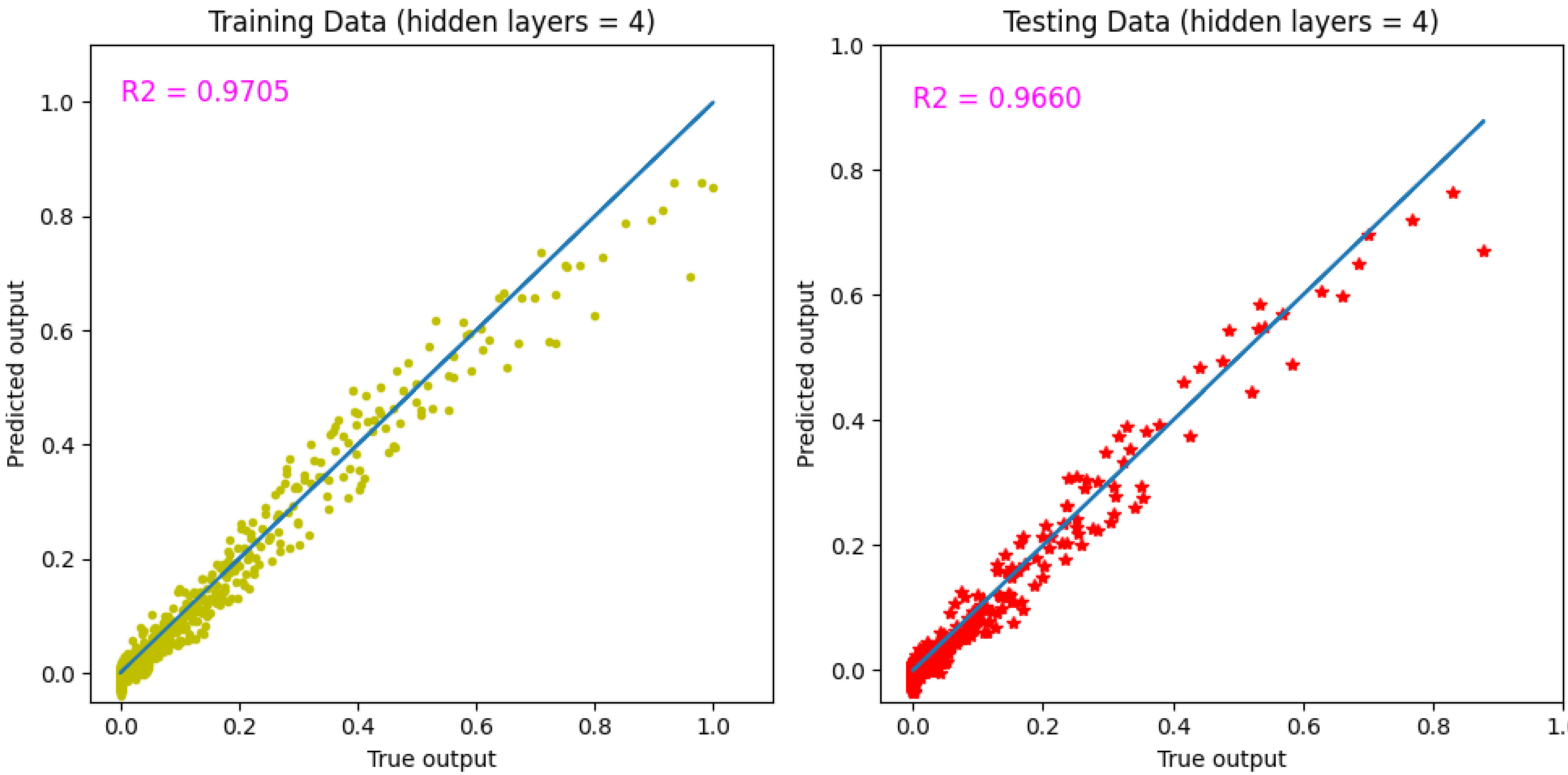


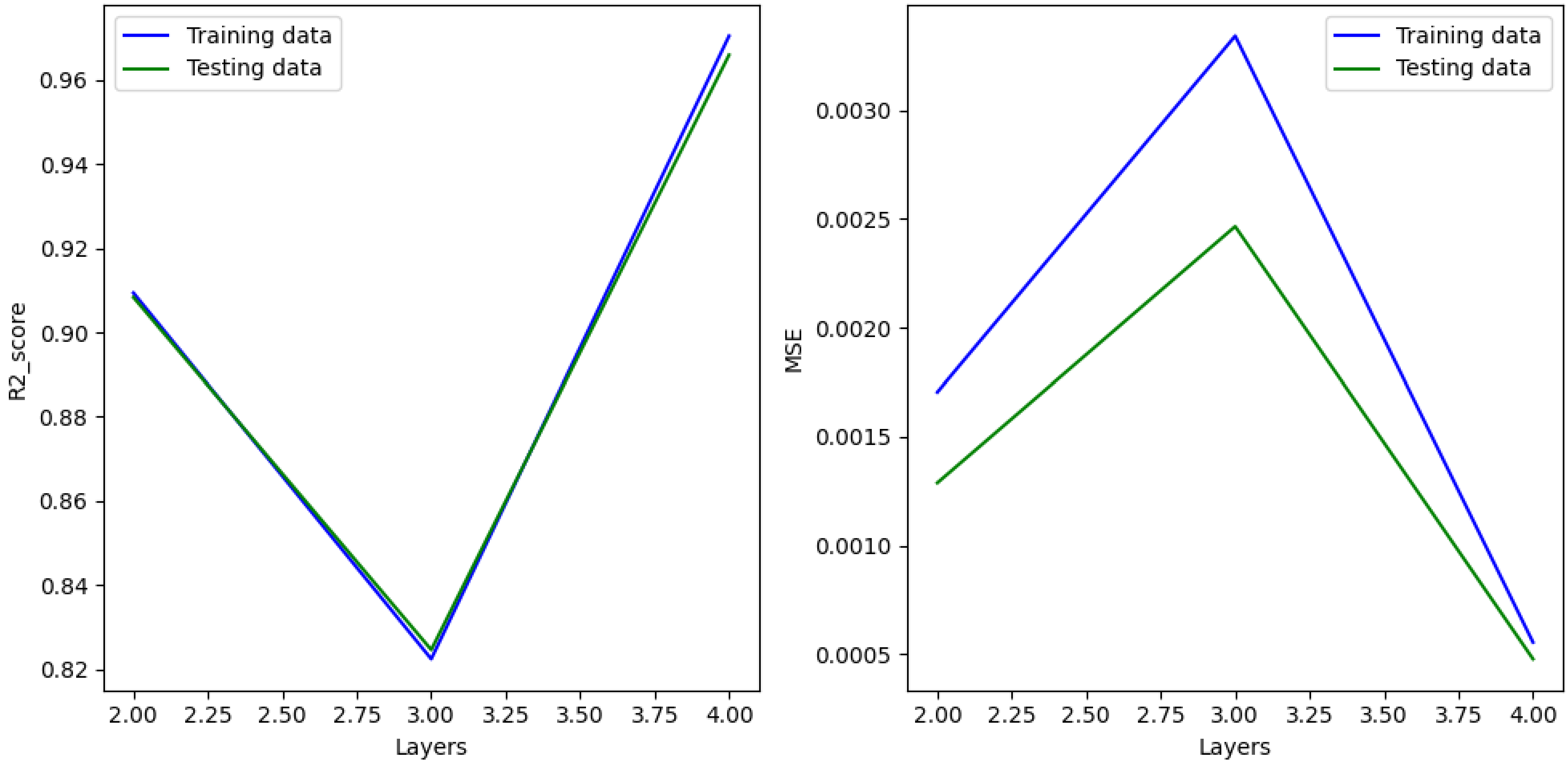

```
Mean Squared Error: 0.0033
R2 Score: 0.8224
Mean Squared Error: 0.0025
R2 Score: 0.8246
```






```
Mean Squared Error: 0.0006
R2 Score: 0.9705
Mean Squared Error: 0.0005
R2 Score: 0.9660
```


Training Data (hidden layers = 4)
R2 = 0.9705
Predicted output
True output
Testing Data (hidden layers = 4)
R2 = 0.9660
Predicted output
True output

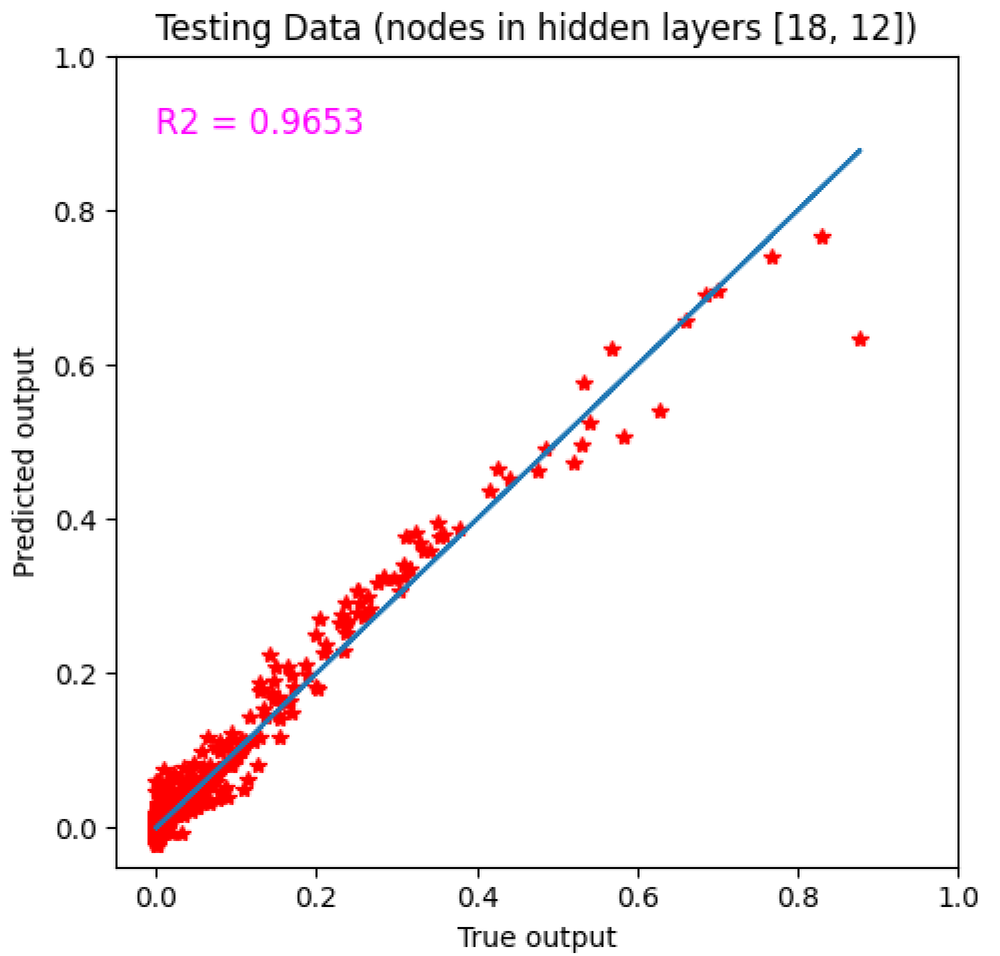


In [ ]:

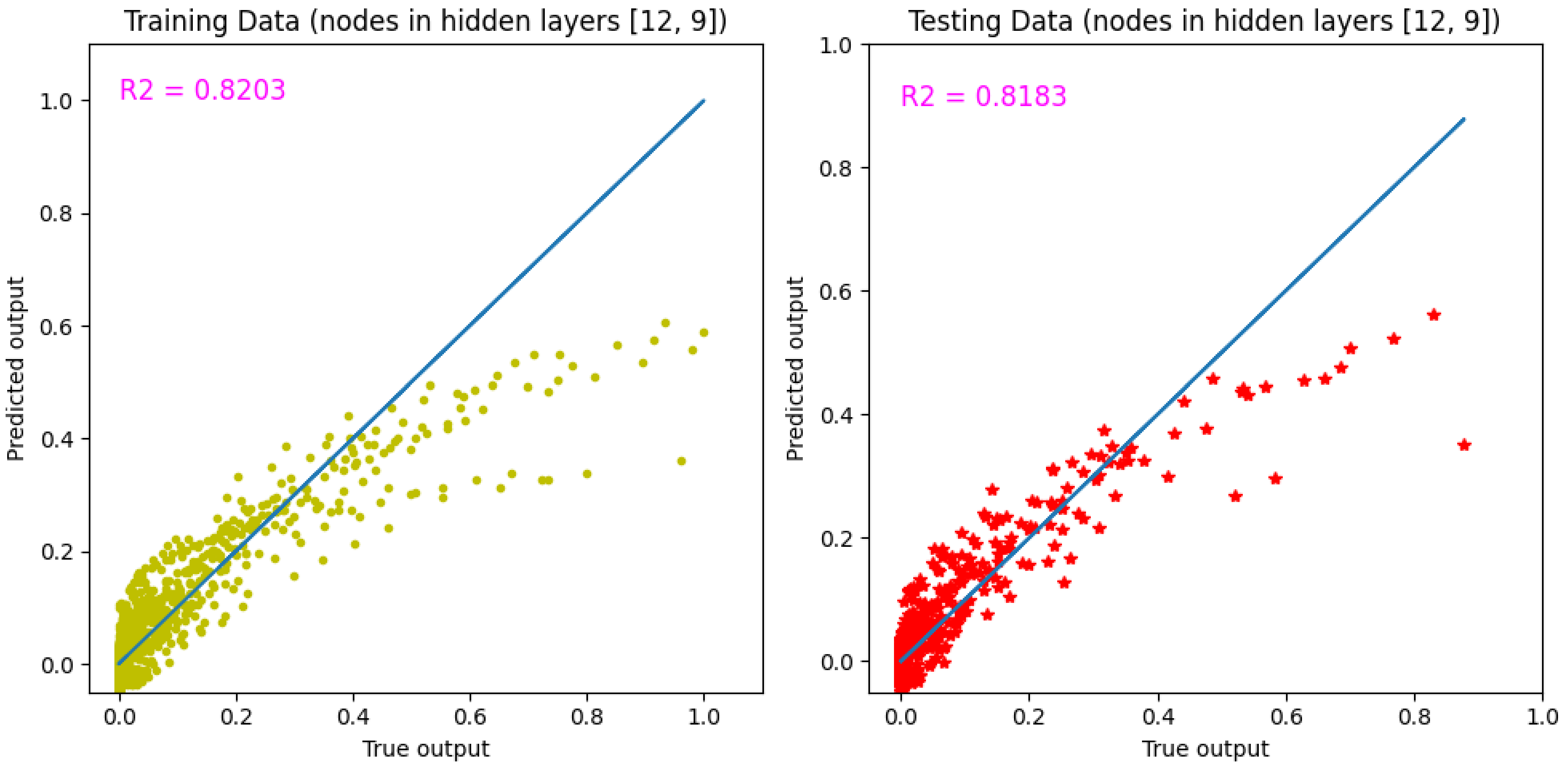
```
# Varying nodes
tr_r72,tr_mse72,te_r72,te_mse72 = [],[],[],[]
for b in range(0,3,1):
    iter72 = f'nodes in hidden layers {ex_nodes[b:b+2]}'
    # ex_nodes2.append(ex_nodes[b:b+2])
    #Creating a network
    ann721 = network(ex_layers[0],ex_nodes[b:b+2],ex_activation_functions[0])
    # Training
    ann722 = training(ann721,'adam','MSE',splitted_data[0],splitted_data[2],ex_epochs[1])
    # Train data results
    z72_train = ann722.predict(splitted_data[0],verbose=0)
    # Test data results
    z72_test = ann722.predict(splitted_data[1],verbose=0)
    # Results
```

```
    r72,mse72 = result(z72_train,splitted_data[2])
    r722,mse722 = result(z72_test,splitted_data[3])
    tr_r72.append(r72)
    tr_mse72.append(mse72)
    te_r72.append(r722)
    te_mse72.append(mse722)
    #Plotting
    plotting(splitted_data,z72_train,z72_test,r72,r722,iter72)
plotting2(tr_r72,tr_mse71,te_r72,te_mse71)
```

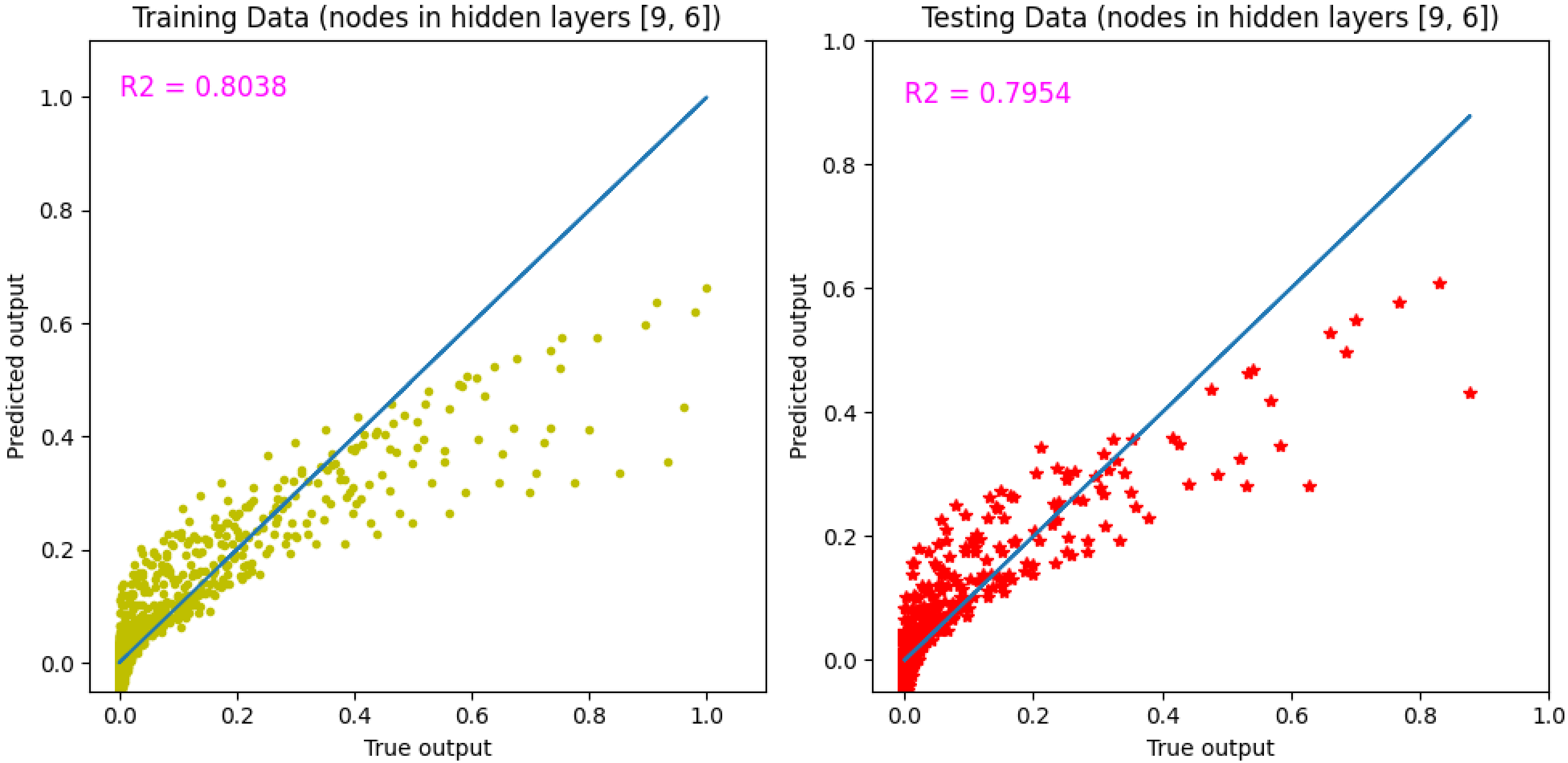
```
Mean Squared Error: 0.0006
R2 Score: 0.9669
Mean Squared Error: 0.0005
R2 Score: 0.9653
```

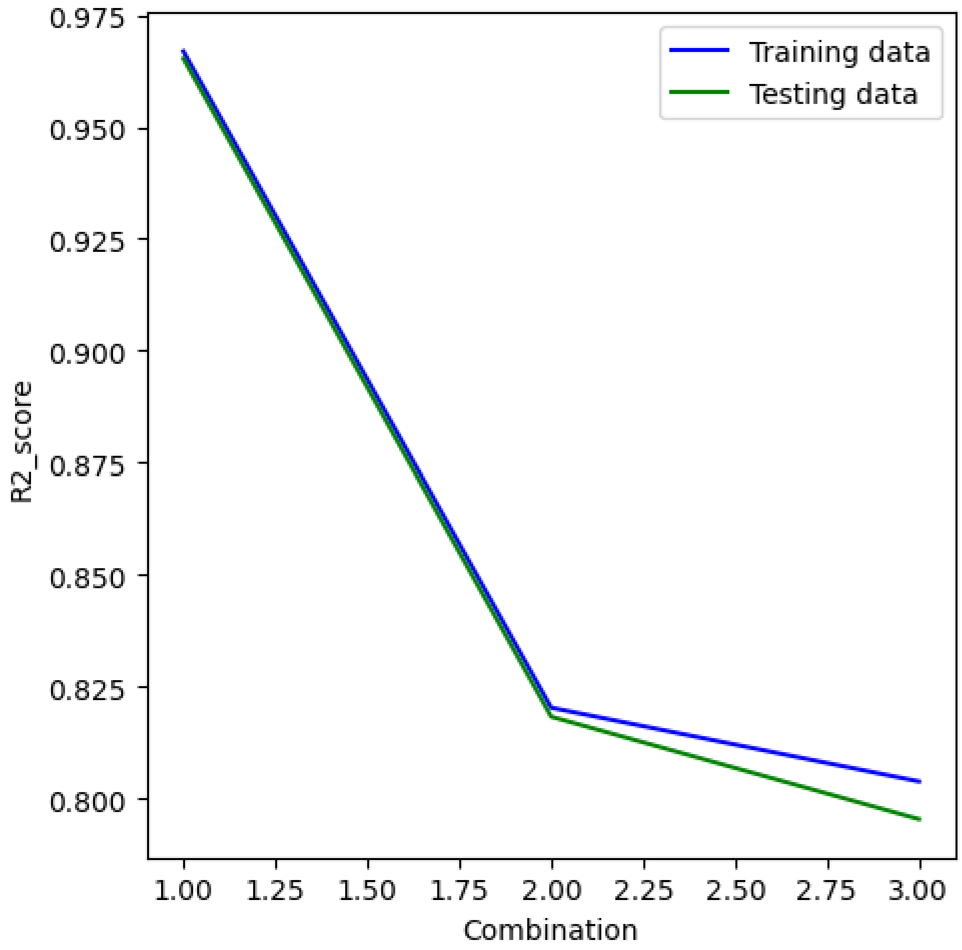


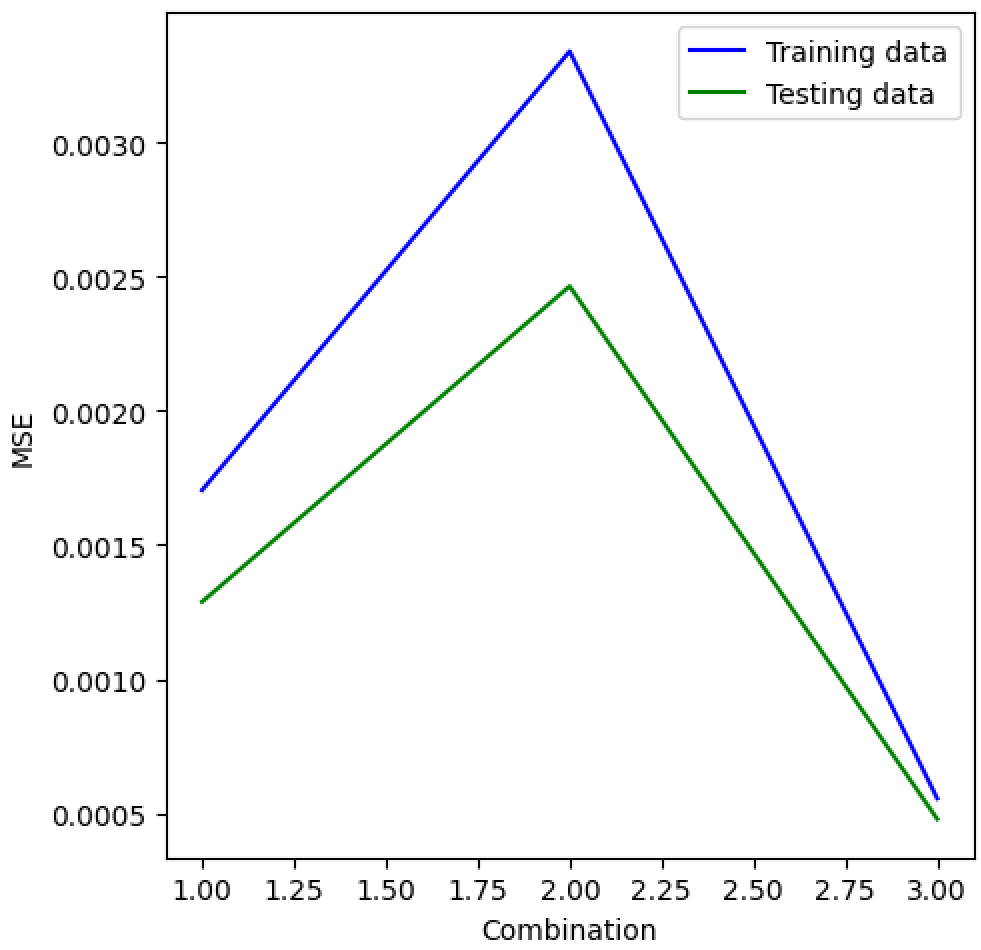
```
Mean Squared Error: 0.0034
R2 Score: 0.8203
Mean Squared Error: 0.0026
R2 Score: 0.8183
```




```
Mean Squared Error: 0.0037
R2 Score: 0.8038
Mean Squared Error: 0.0029
R2 Score: 0.7954
```


Training Data (nodes in hidden layers [9, 6])
R2 = 0.8038
Predicted output
True output
0.0
0.2
0.4
0.6
0.8
1.0
Testing Data (nodes in hidden layers [9, 6])
R2 = 0.7954
Predicted output
True output



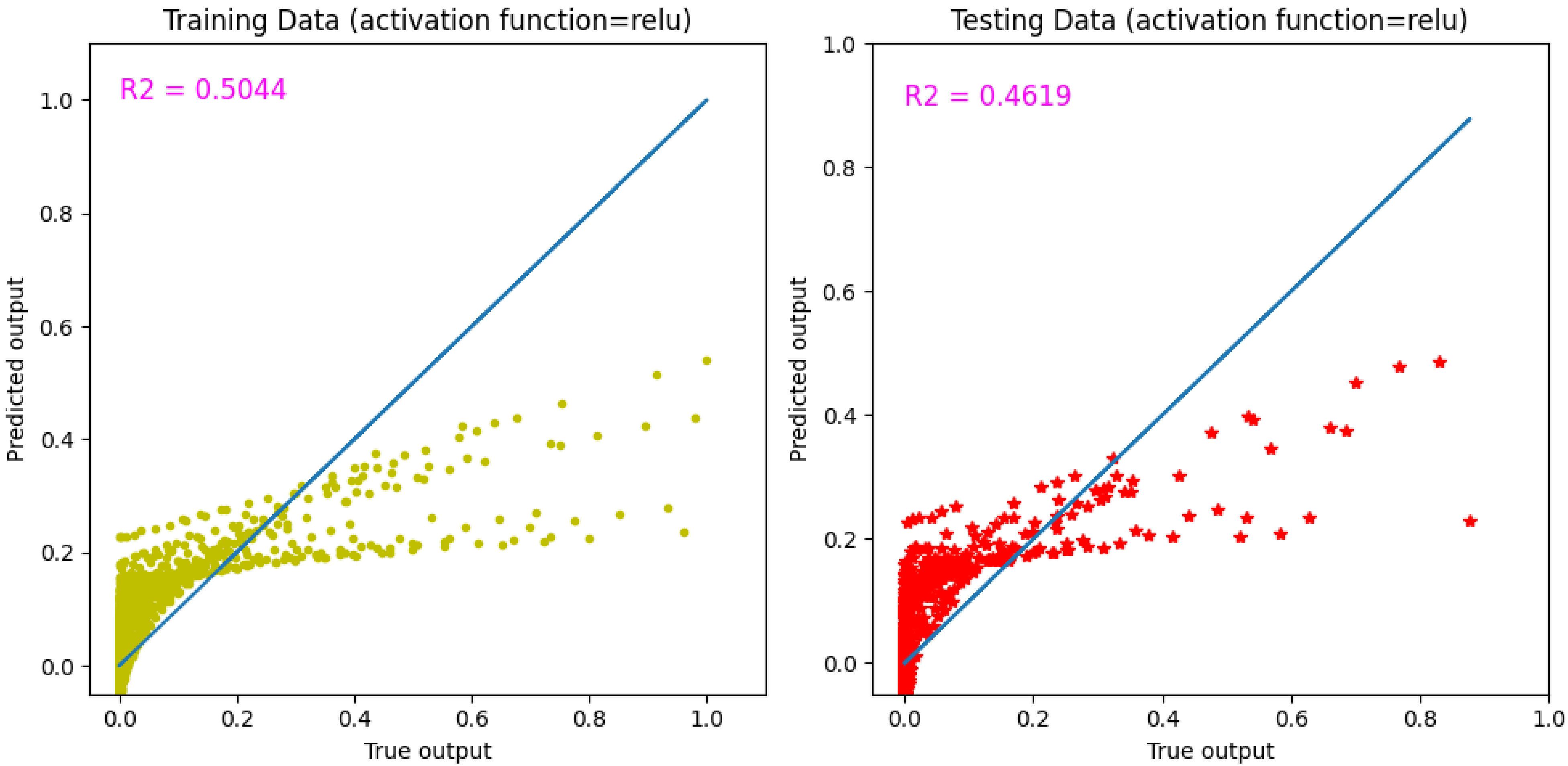


In [ ]:

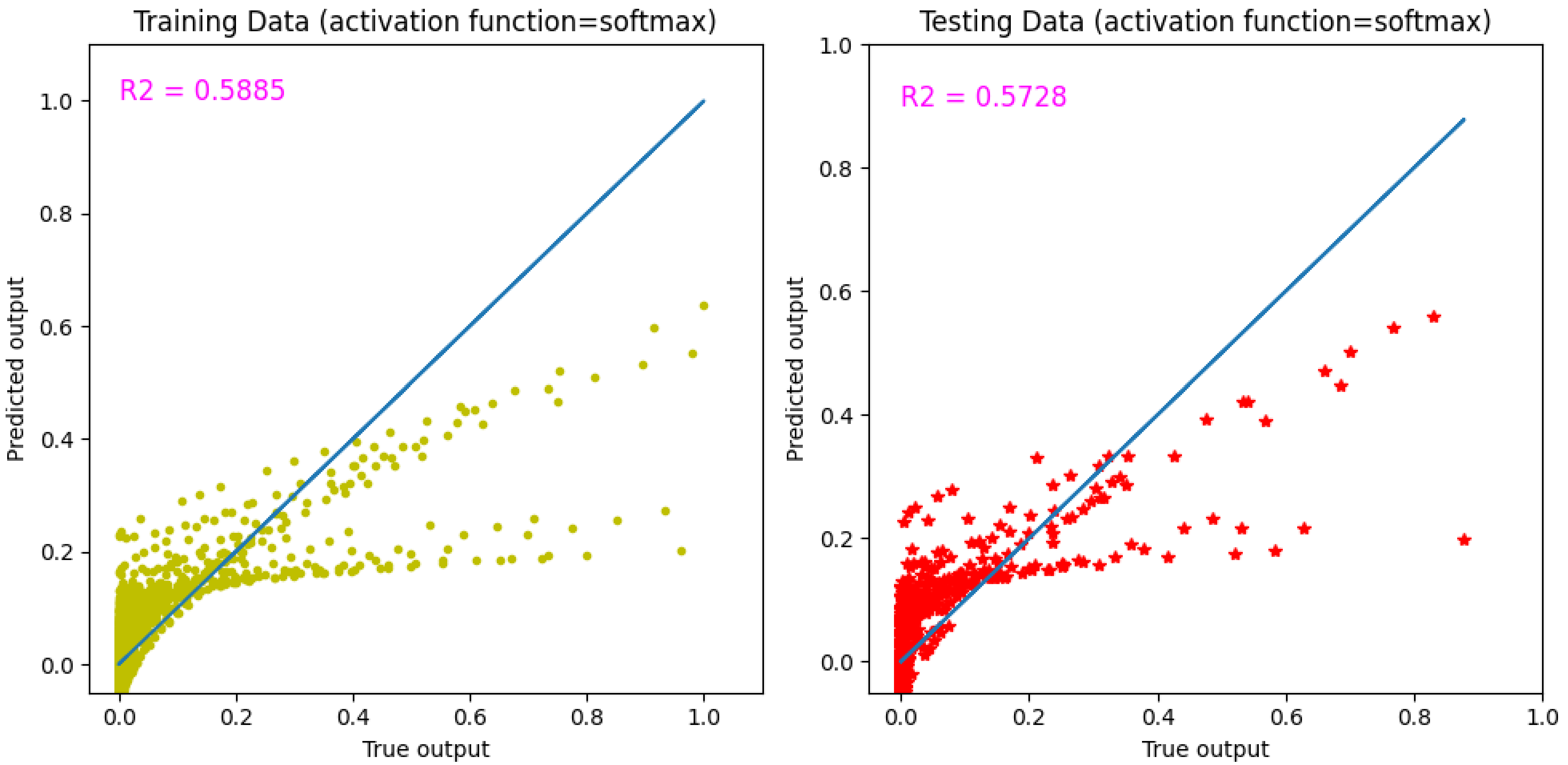
```
# Varying activation functions
tr_r73,tr_mse73,te_r73,te_mse73 = [],[],[],[]
for c in range(0,3,1):
    iter73 = f'activation function={ex_activation_functions[c]}'
    ann731 = network(ex_layers[0],ex_nodes,ex_activation_functions[c])
    ann732 = training(ann732,'adam','MSE',splitted_data[0],splitted_data[2],ex_epochs[1])
    # Train data
    z73_train = ann732.predict(splitted_data[0],verbose=0)
    # Test data
    z73_test = ann732.predict(splitted_data[1],verbose=0)
    # Results
    r73,mse73 = result(z73_train,splitted_data[2])
    r733,mse733 = result(z73_test,splitted_data[3])
    tr_r73.append(r73)
```

```
    tr_mse73.append(mse73)
    te_r73.append(r733)
    te_mse73.append(mse733)
    #Plotting
    plotting(splitted_data,z73_train,z73_test,r73,r733,iter73)
plotting2(tr_r73,tr_mse73,te_r73,te_mse73)
```

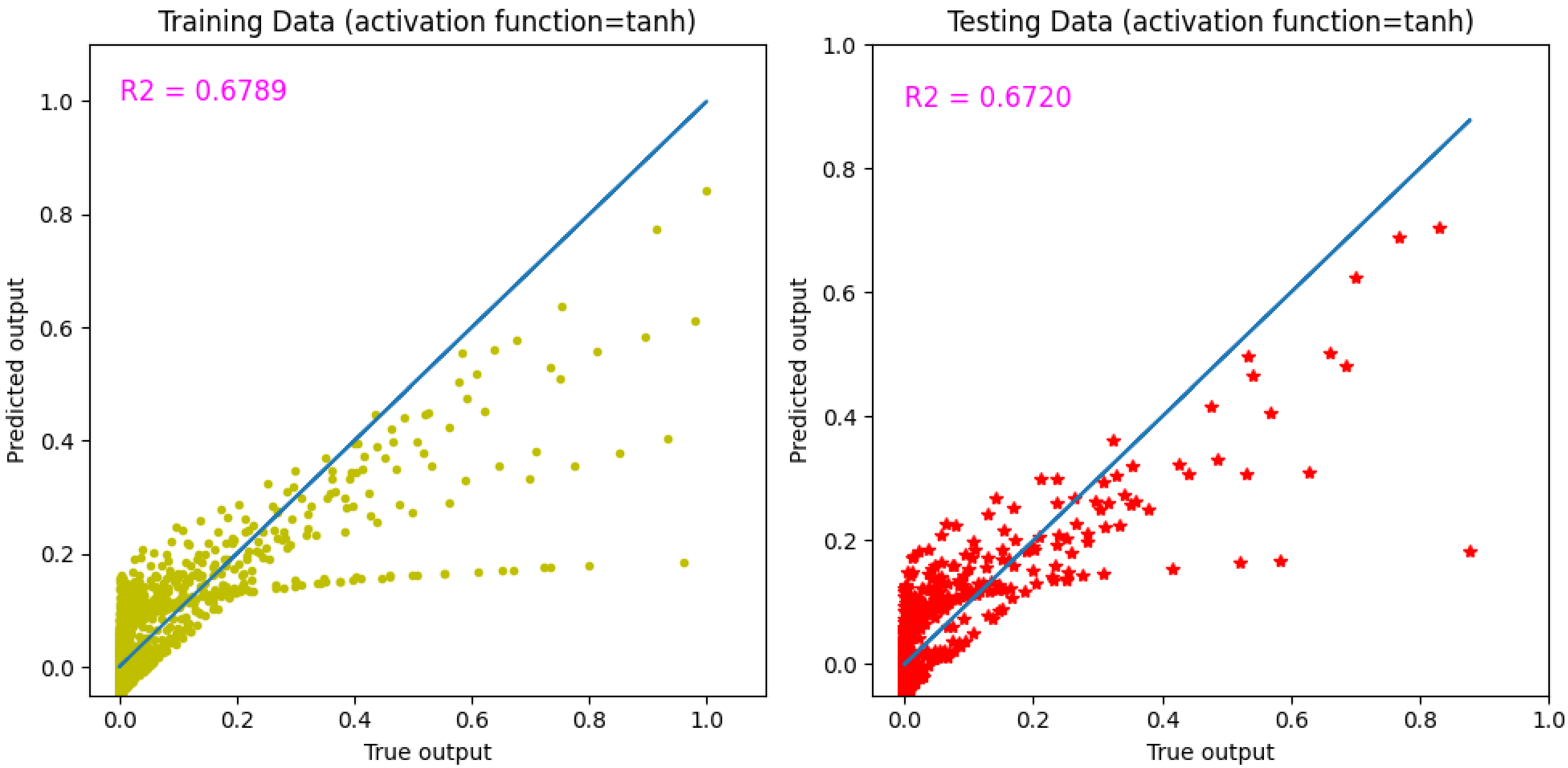
```
Mean Squared Error: 0.0093
R2 Score: 0.5044
Mean Squared Error: 0.0076
R2 Score: 0.4619
```



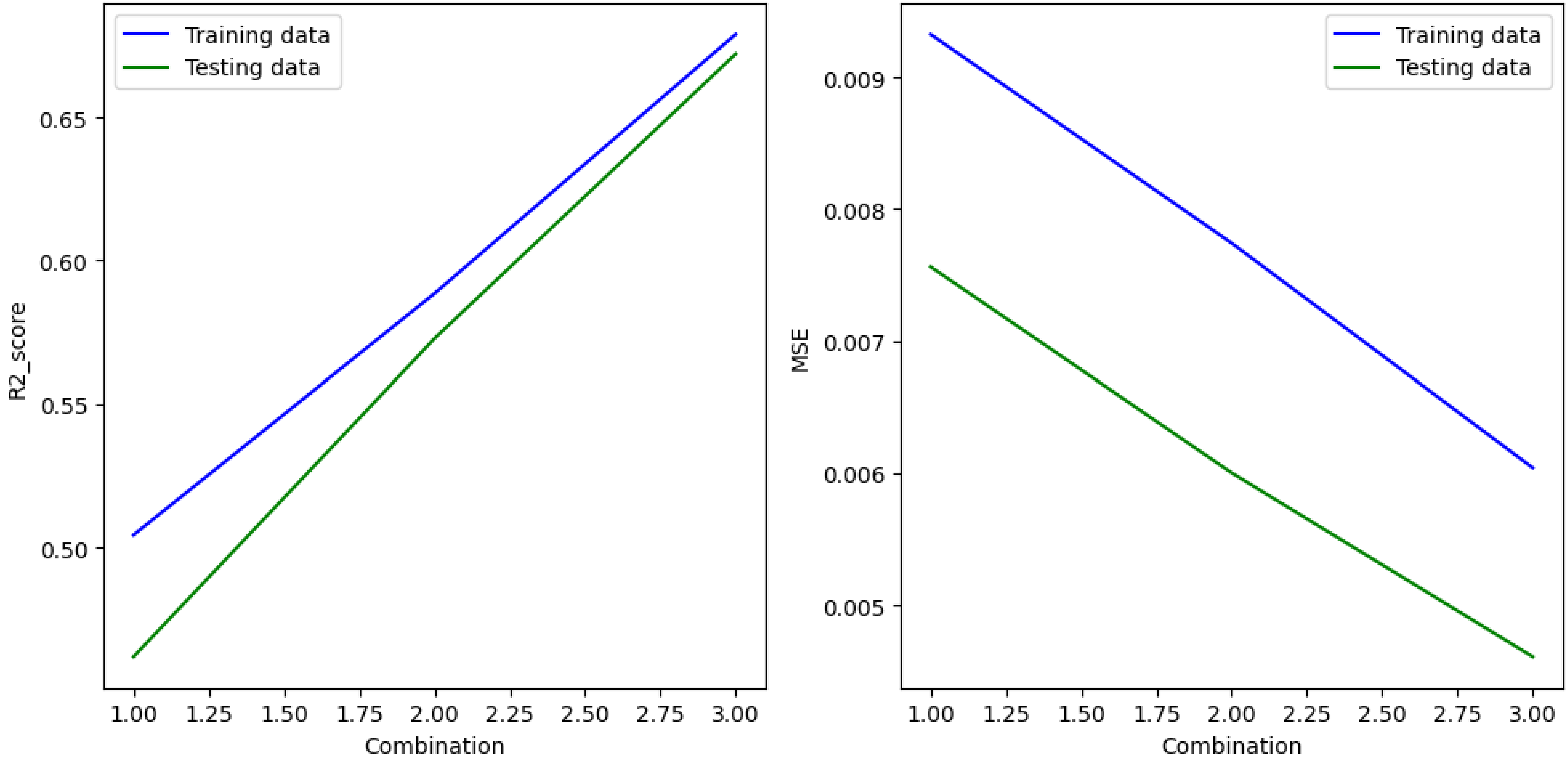

```
Mean Squared Error: 0.0077
R2 Score: 0.5885
Mean Squared Error: 0.0060
R2 Score: 0.5728
```




```
Mean Squared Error: 0.0060
R2 Score: 0.6789
Mean Squared Error: 0.0046
R2 Score: 0.6720
```



Training Data (activation function=tanh)
R2 = 0.6789
Predicted output
True output
0.0
0.2
0.4
0.6
0.8
1.0
Testing Data (activation function=tanh)
R2 = 0.6720
Predicted output
True output
0.0
0.2
0.4
0.6
0.8
1.0

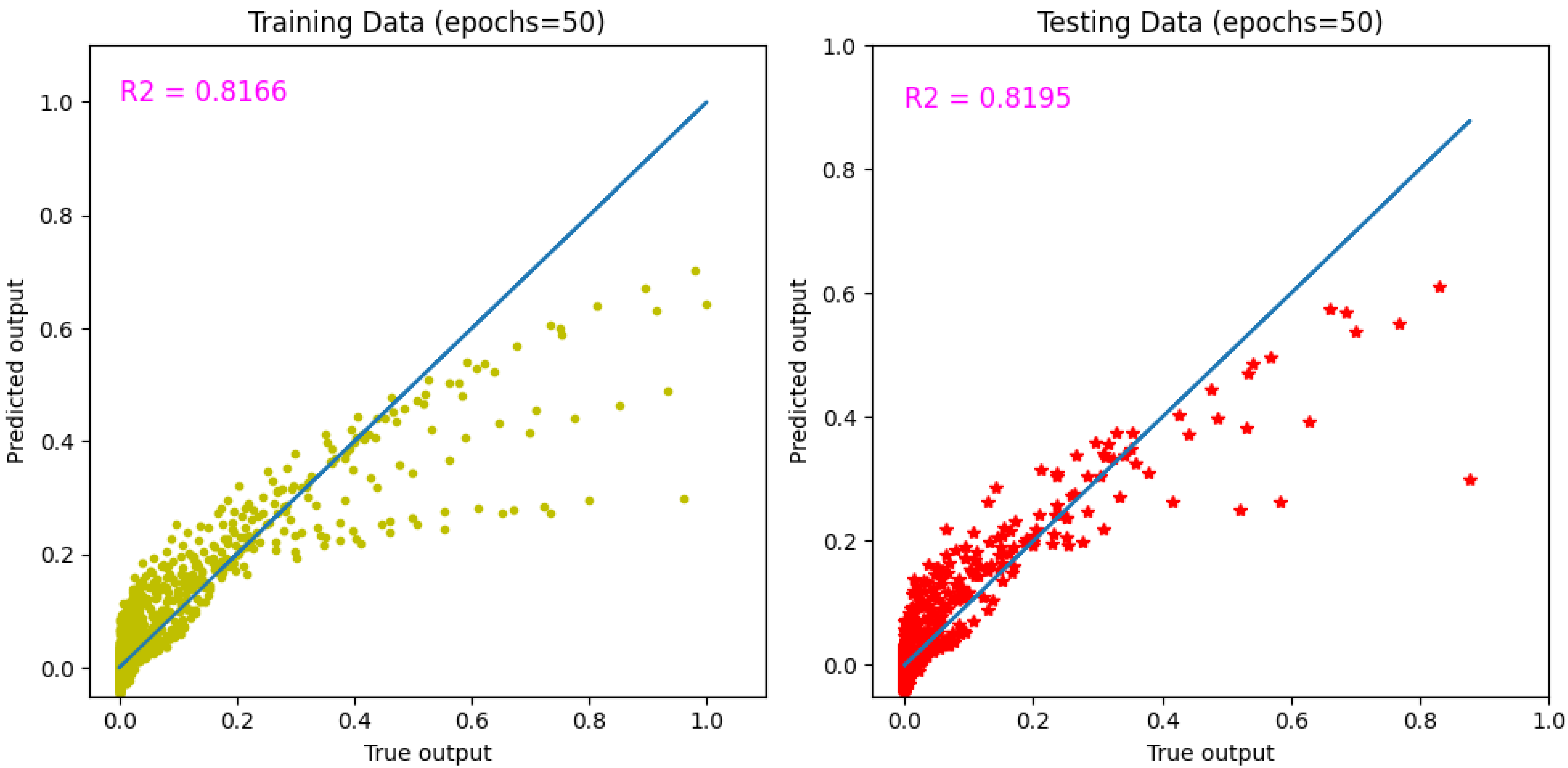


In [ ]:

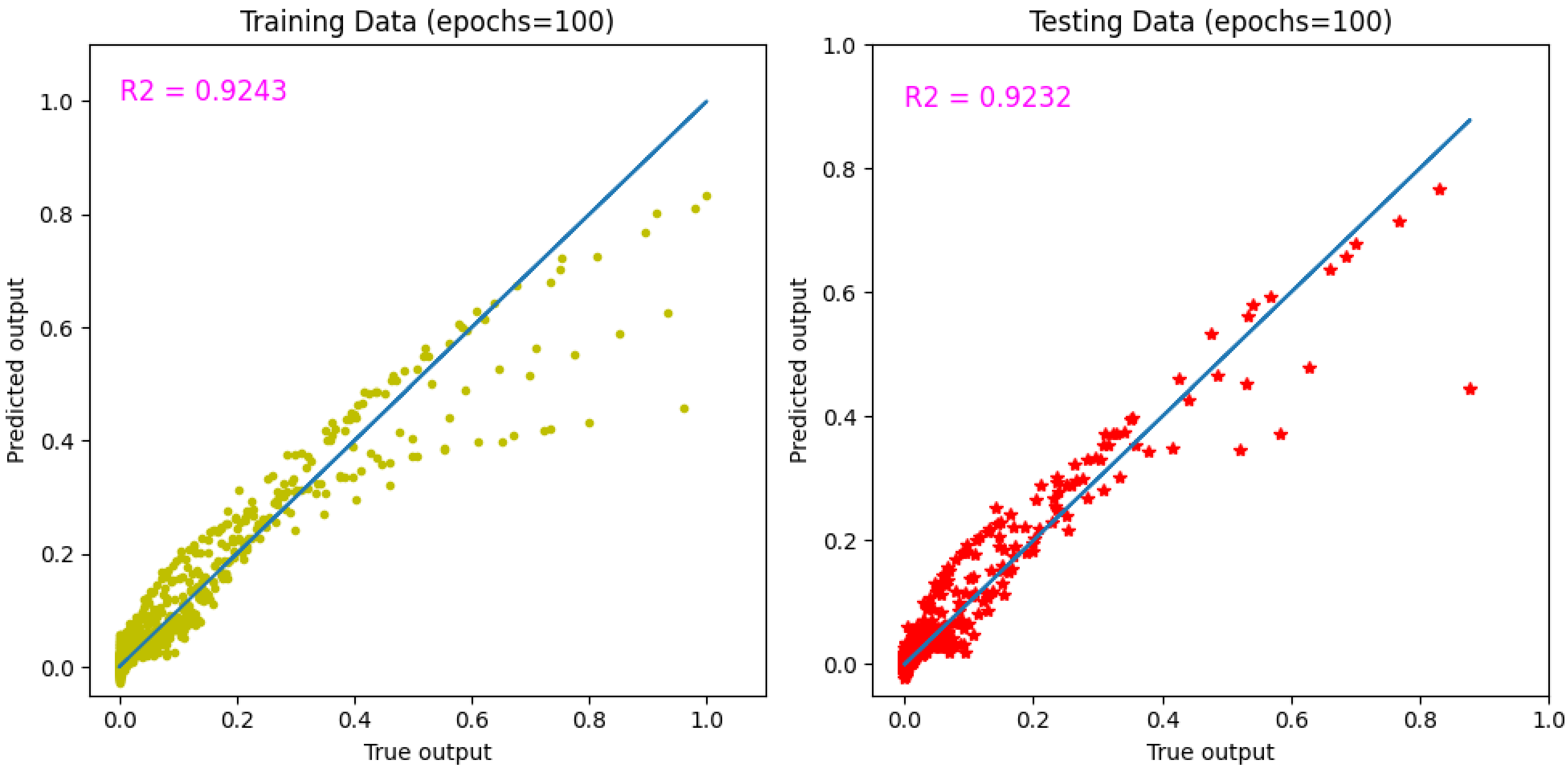
```
# Varying epochs
tr_r74,tr_mse74,te_r74,te_mse74 = [],[],[],[]
for d in range(0,3,1):
    iter74 = f'epochs={ex_epochs[d]}'
    ann741 = network(ex_layers[0],ex_nodes,ex_activation_functions[0])
    ann742 = training(ann741,'adam','MSE',splitted_data[0],splitted_data[2],ex_epochs[d])
    # Train data
    z74_train = ann742.predict(splitted_data[0],verbose=0)
    # Test data
    z74_test = ann742.predict(splitted_data[1],verbose=0)
    # Results
    r74,mse74  = result(z74_train,splitted_data[2])
    r744,mse744 = result(z74_test,splitted_data[3])
    tr_r74.append(r74)
```

```
    tr_mse74.append(mse74)
    te_r74.append(r744)
    te_mse74.append(mse744)
    #Plotting
    plotting(splitted_data,z74_train,z74_test,r74,r744,iter74)
plotting2(tr_r74,tr_mse74,te_r74,te_mse74,ex_epochs,'epochs')
```

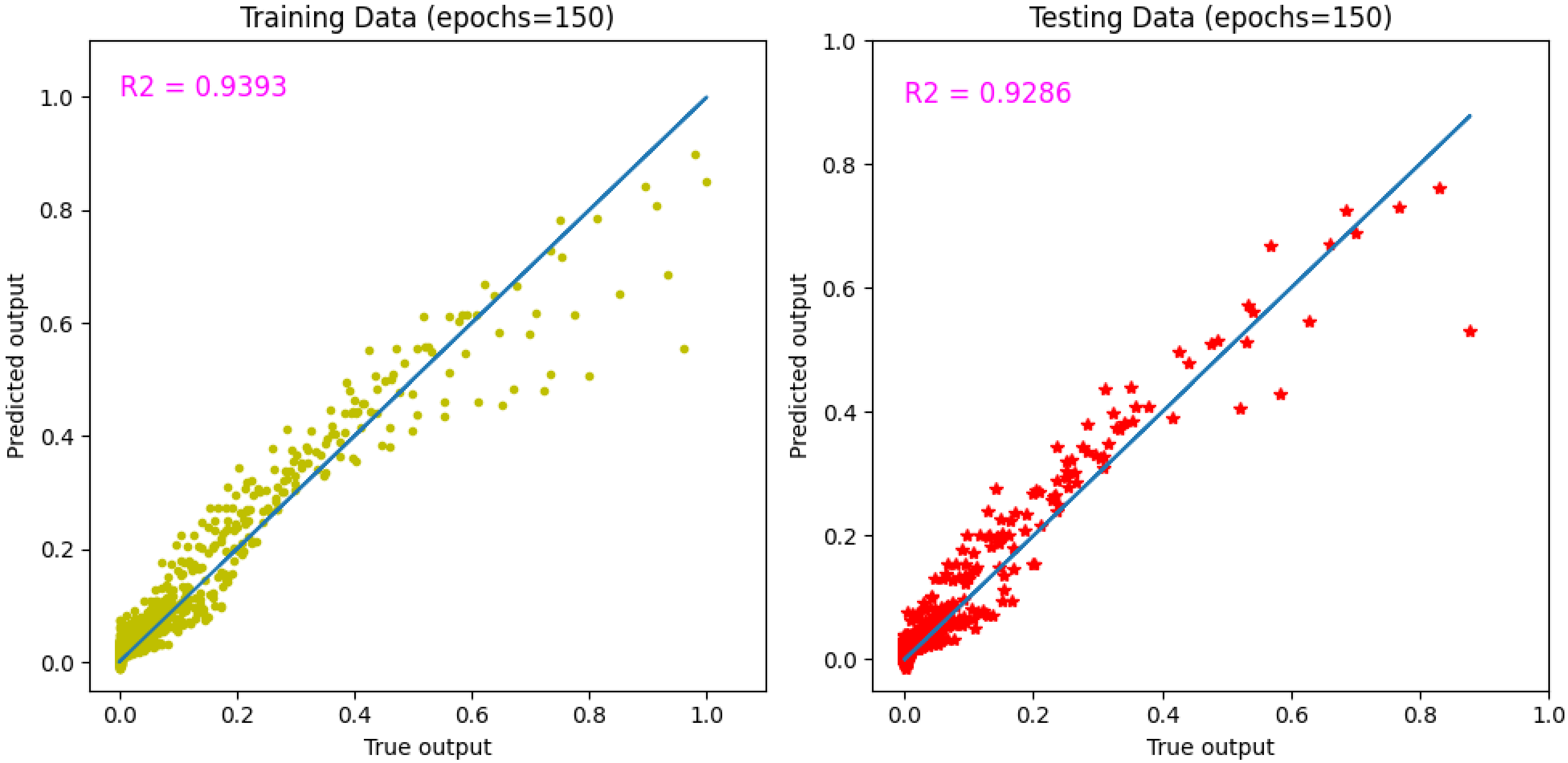
```
Mean Squared Error: 0.0035
R2 Score: 0.8166
Mean Squared Error: 0.0025
R2 Score: 0.8195
```



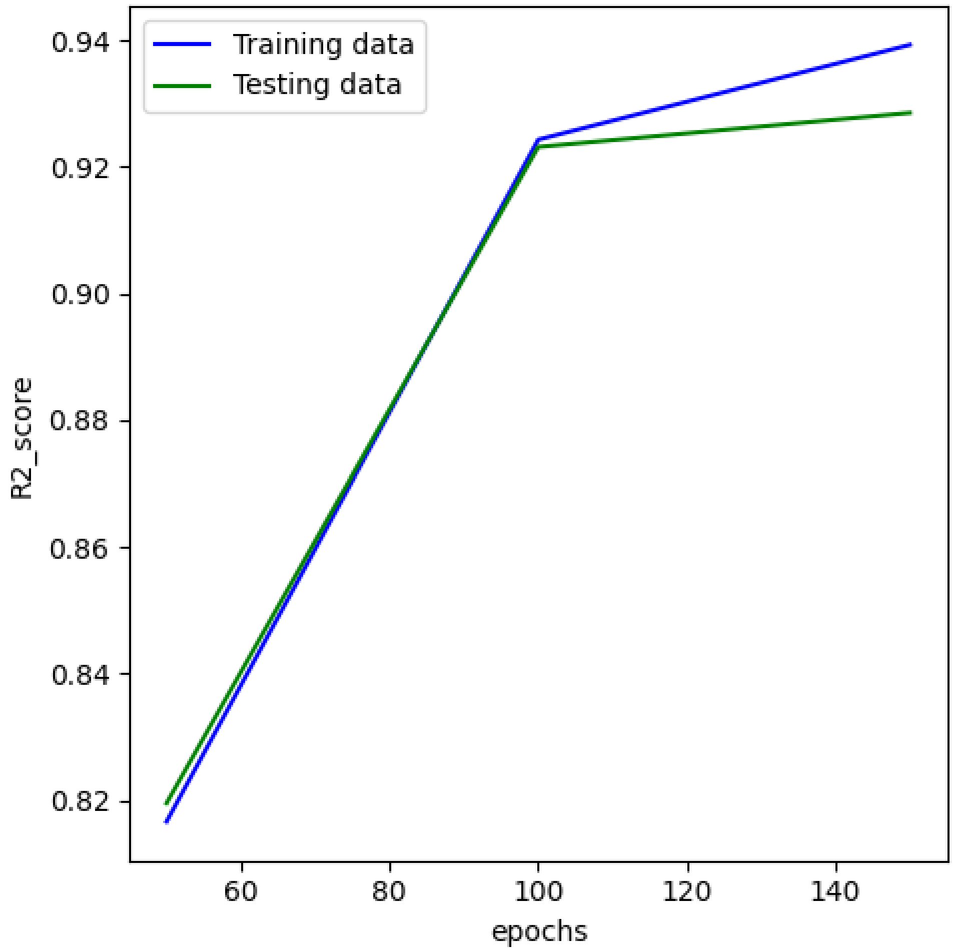

```
Mean Squared Error: 0.0014
R2 Score: 0.9243
Mean Squared Error: 0.0011
R2 Score: 0.9232
```

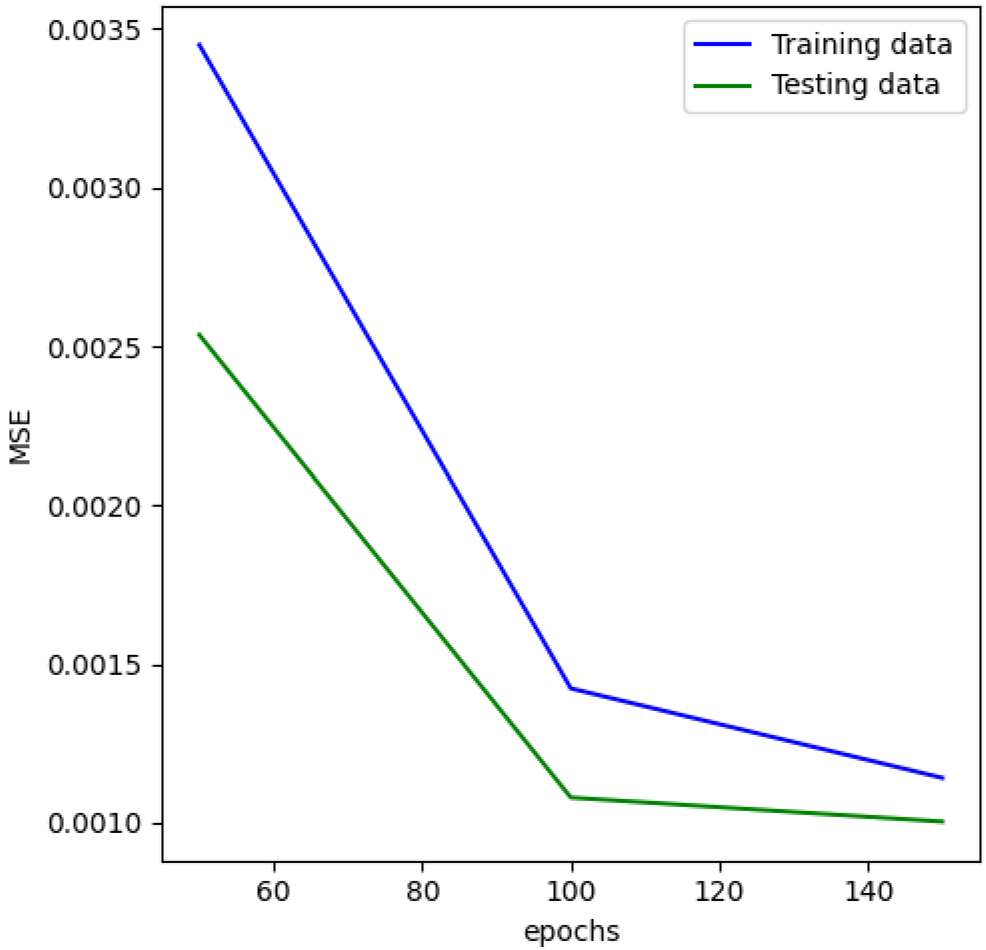



```
Mean Squared Error: 0.0011
R2 Score: 0.9393
Mean Squared Error: 0.0010
R2 Score: 0.9286
```



Training Data (epochs=150)
R2 = 0.9393
Predicted output
True output
0.0
0.2
0.4
0.6
0.8
1.0
Testing Data (epochs=150)
R2 = 0.9286



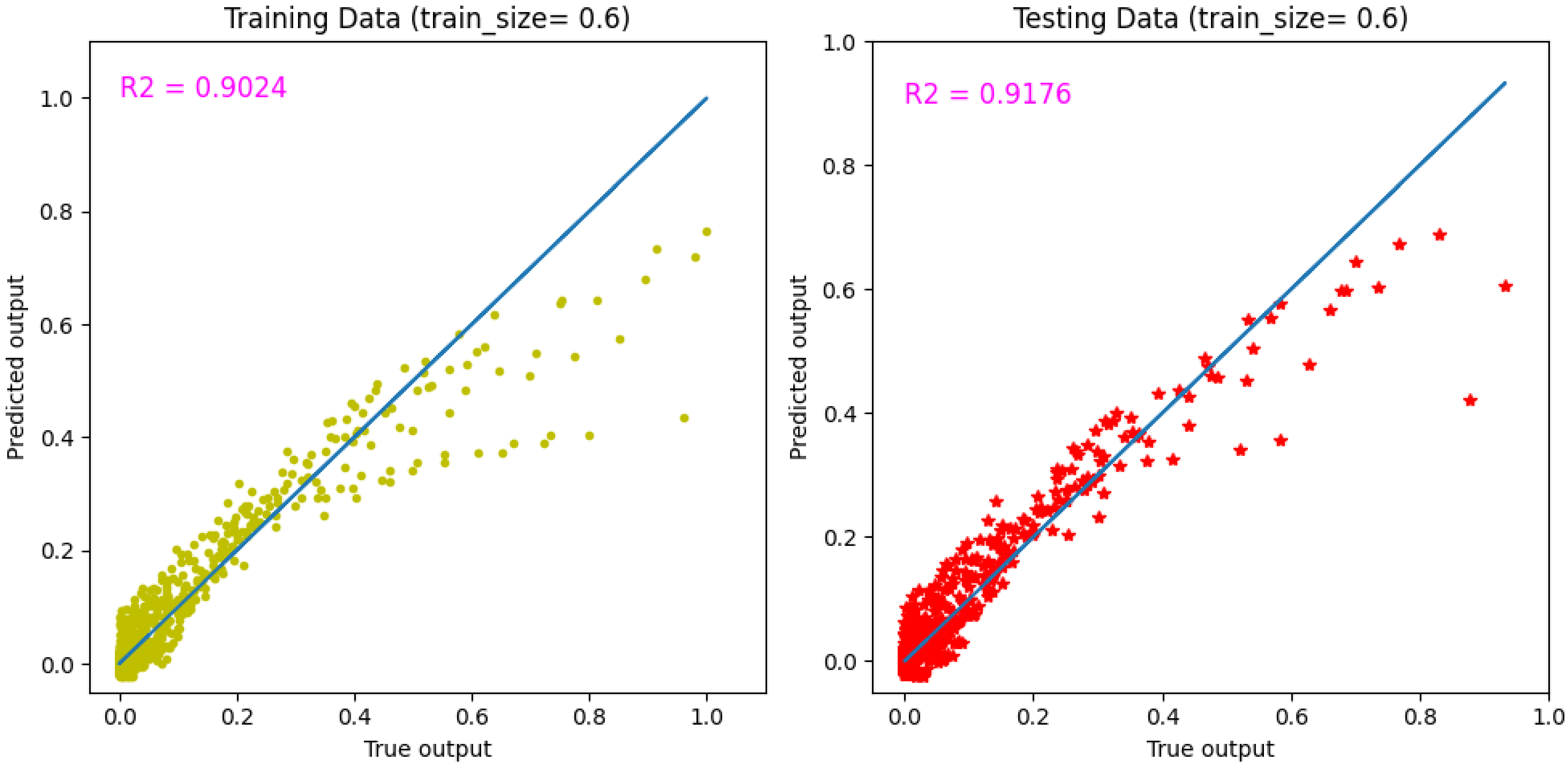


In [ ]:

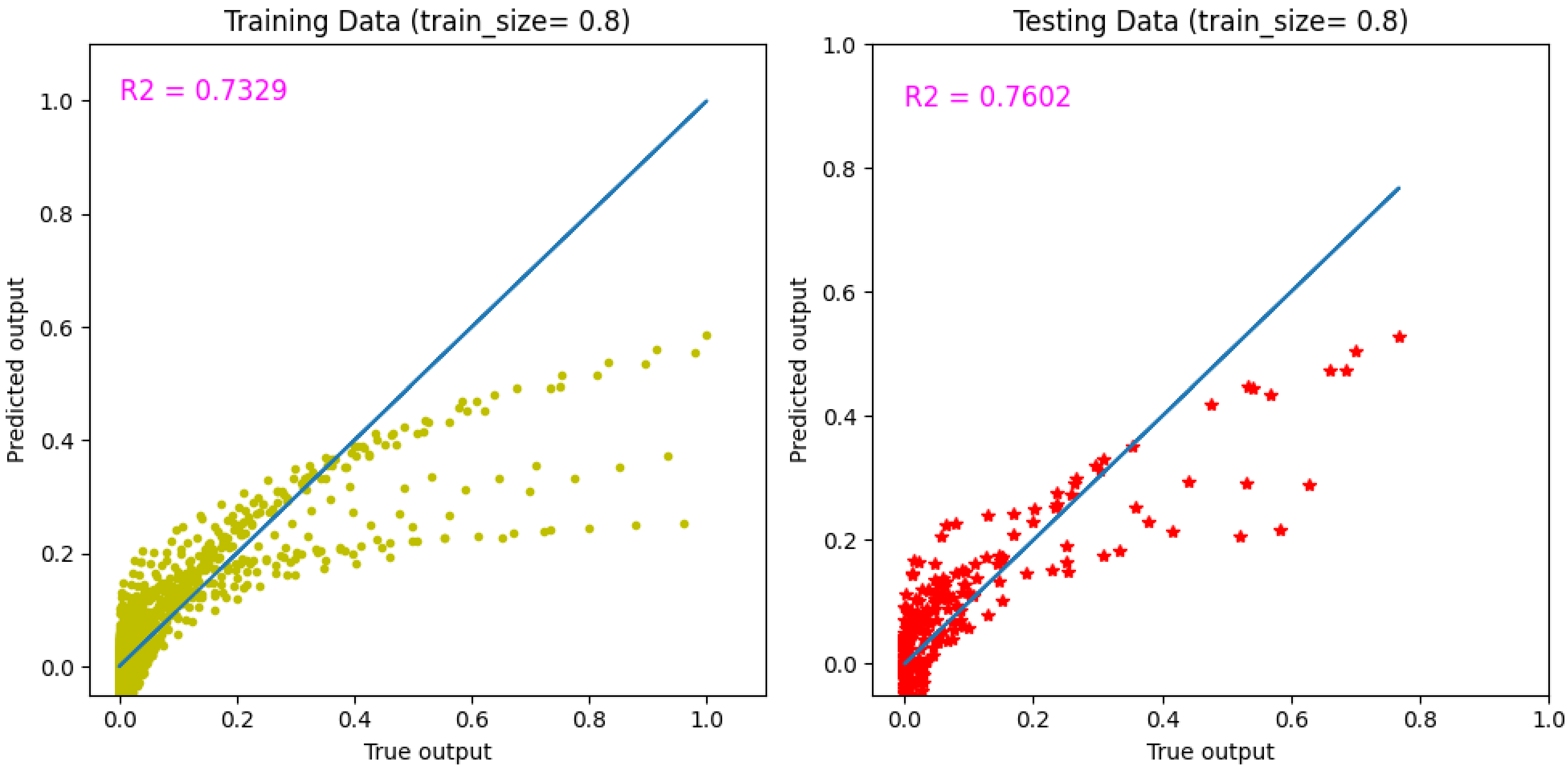
```
# Varying sample size
tr_r75,tr_mse75,te_r75,te_mse75 = [],[],[],[]
for e in range(0,3,1):
    iter75 = f'train_size= {ex_train_size[e]}'
    sp_data = split(data3,ex_train_size[e])
    ann751 = network(ex_layers[0],ex_nodes,ex_activation_functions[0])
    ann752 = training(ann751,'adam','MSE',sp_data[0],sp_data[2],ex_epochs[1])
    # Train data
    z75_train = ann752.predict(sp_data[0],verbose=0)
    # Test data
    z75_test = ann752.predict(sp_data[1],verbose=0)
    # Results
    r75,mse75 = result(z75_train,sp_data[2])
    r755,mse755 = result(z75_test,sp_data[3])
```

```
    tr_r75.append(r75)
    tr_mse75.append(mse75)
    te_r75.append(r755)
    te_mse75.append(mse755)
    #Plotting
    plotting(sp_data,z75_train,z75_test,r75,r755,iter75)
plotting2(tr_r75,tr_mse75,te_r75,te_mse75,ex_train_size,'Training Data size')
```

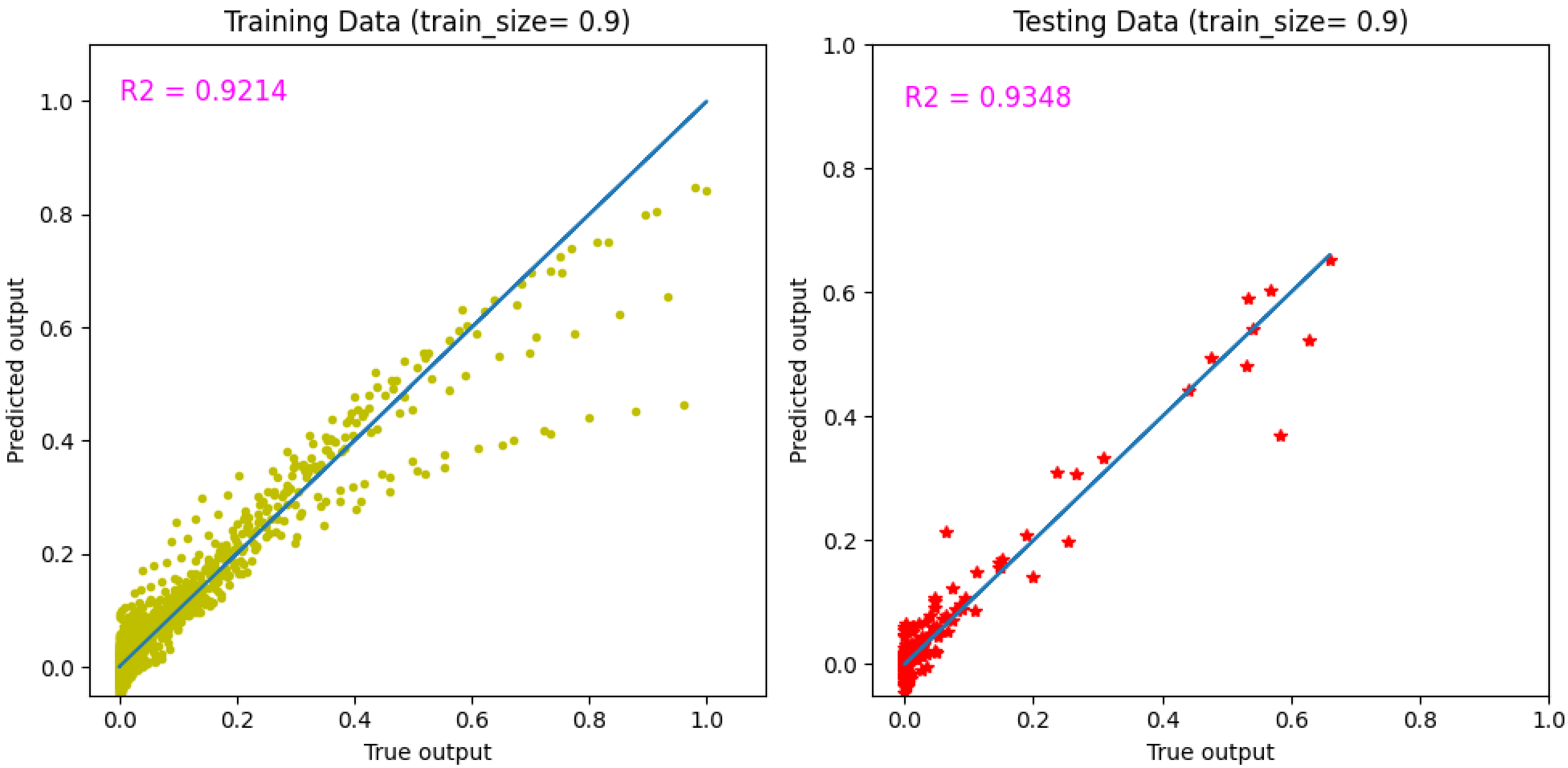
```
Mean Squared Error: 0.0019
R2 Score: 0.9024
Mean Squared Error: 0.0012
R2 Score: 0.9176
```

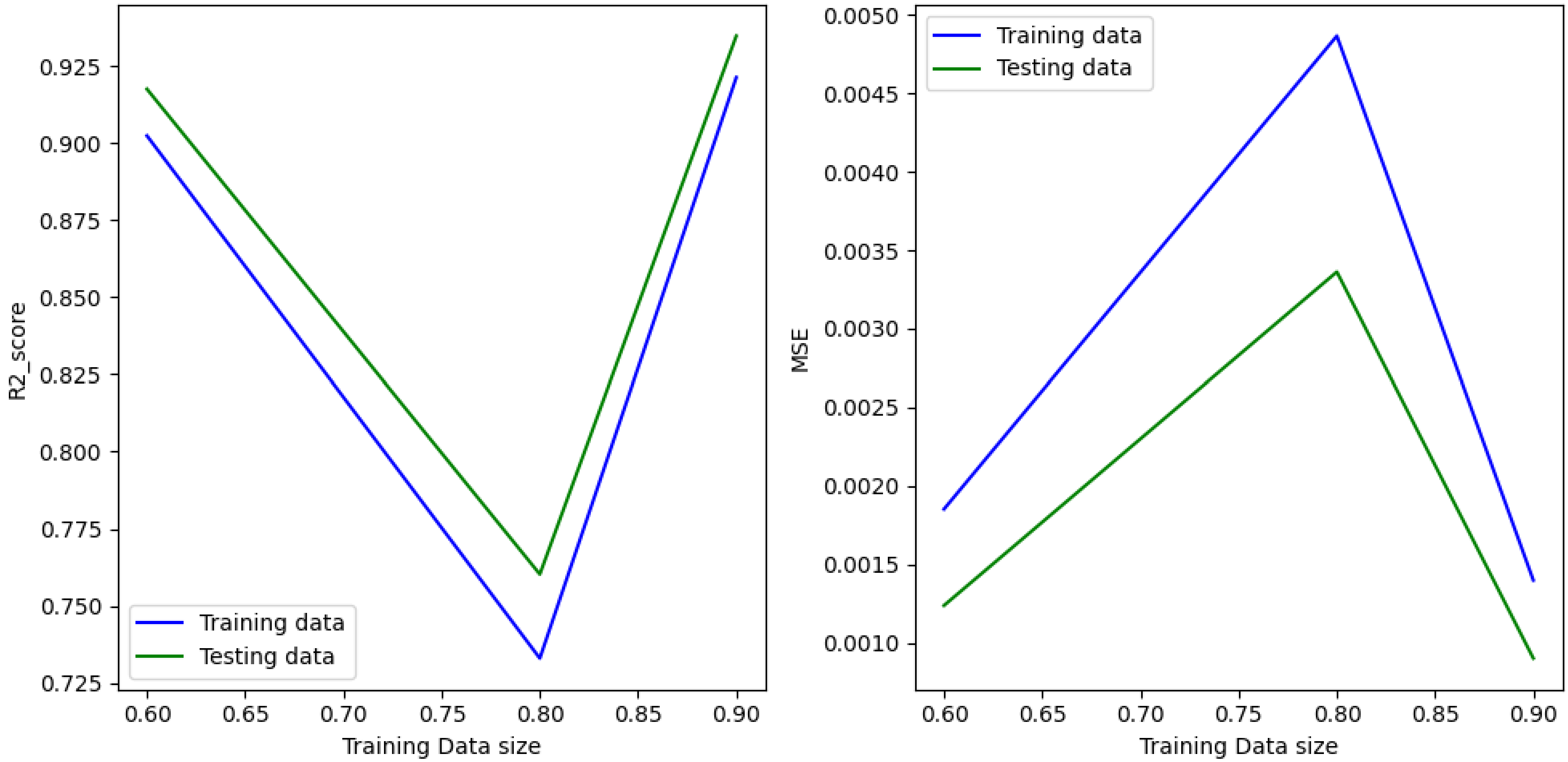


```
Mean Squared Error: 0.0049
R2 Score: 0.7329
Mean Squared Error: 0.0034
R2 Score: 0.7602
```




```
Mean Squared Error: 0.0014
R2 Score: 0.9214
Mean Squared Error: 0.0009
R2 Score: 0.9348
```


Training Data (train_size= 0.9)
R2 = 0.9214
Predicted output
True output
0.0
0.2
0.4
0.6
0.8
1.0
Testing Data (train_size= 0.9)
R2 = 0.9348
Predicted output
True output


R2_score
Training Data size
Training data
Testing data
MSE
Training Data size
Training data
Testing data