Comparative Study of the Design of Neural Networks Architecture for the Diagnosis of Heart Disease.

55-700241 Applicable Artificial Intelligence

BY

AIYELARI TEMILOLORUN JUDE

31052830

2023

Contents

Introduction

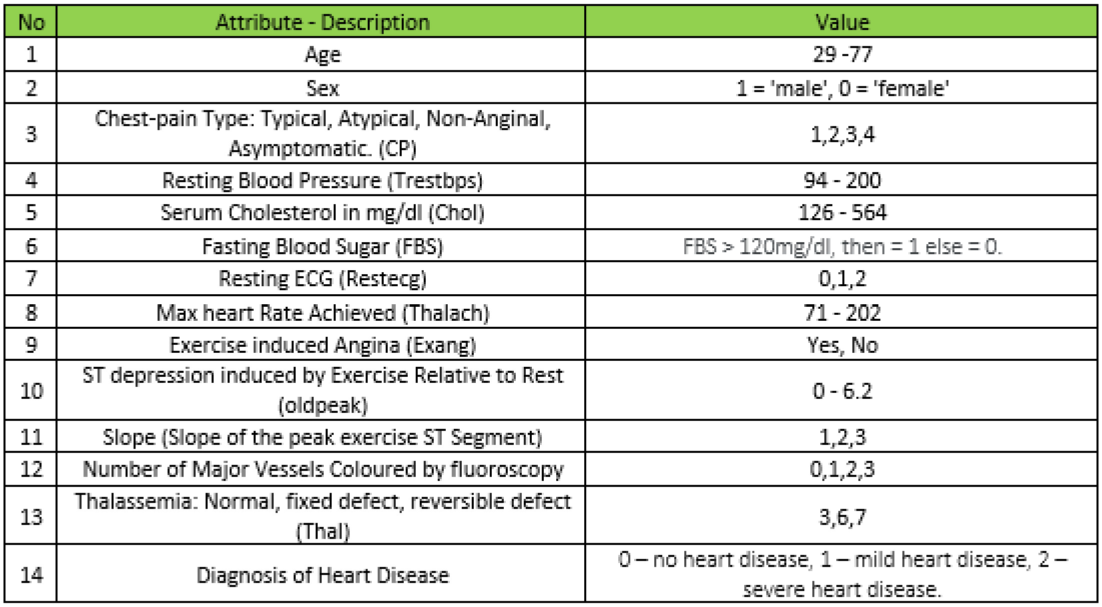
Every day, the human heart beats over 100,000 times a day pumping the equivalent of 1.5 gallons of blood every 60 seconds. This adds up to over two thousand gallons of blood in a 24-hour cycle. The heart pumps blood round the human body making use of over 60,000 miles of blood vessels in the body.

Heart disease refers to a range of conditions that affect the normal function of the heart. These includes but is not limited to blood vessel disease such as coronary artery disease, irregular heartbeat also known as arrhythmias, heart muscle disease, heart valve disease, etc.

According to the WHO, CVDs are the leading cause of death globally, taking an estimated 17.9millions live each year. More than 4/5 of CVD deaths are due to heart attacks and strokes, while 1/3 of these deaths occur prematurely in people under 70 years of age. These stats alone make heart disease a major concern in the world. But it is difficult to identify heart disease because of several contributory risk factors such as unhealthy diet, physical inactivity, tobacco use, etc. One popular approach is to use machine learning techniques to soft over recorded data and try to obtain a relationship between the data and the different heart conditions.

For our report, we would use a subset of the Cleveland heart disease dataset taken from the UCI repository. This dataset is a cleaned-up subset of 297 observation,13 features and 1 target class indicating 3 different classes (0 – no heart disease, 1 – mild heart disease, 2 – severe heart disease.)

The fourteen features are described as below:



No	Attribute - Description	Value
1	Age	29 -77
2	Sex	1 = 'male', 0 = 'female'
3	Chest-pain Type: Typical, Atypical, Non-Anginal, Asymptomatic. (CP)	1,2,3,4
4	Resting Blood Pressure (Trestbps)	94 - 200
5	Serum Cholesterol in mg/dl (Chol)	126 - 564
6	Fasting Blood Sugar (FBS)	FBS > 120mg/dl, then = 1 else = 0.
7	Resting ECG (Restecg)	0,1,2
8	Max heart Rate Achieved (Thalach)	71 - 202
9	Exercise induced Angina (Exang)	Yes, No
10	ST depression induced by Exercise Relative to Rest (oldpeak)	0 - 6.2
11	Slope (Slope of the peak exercise ST Segment)	1,2,3
12	Number of Major Vessels Coloured by fluoroscopy	0,1,2,3
13	Thalassemia: Normal, fixed defect, reversible defect (Thal)	3,6,7
14	Diagnosis of Heart Disease	0 – no heart disease, 1 – mild heart disease, 2 – severe heart disease.

Table 1: Cleveland dataset attributes

Requirements Analysis

The requirement is that the artificial neural network (ANN) model that will be developed in my report can achieve a cross-validated classification rate as close as possible to the current state of the art which is currently around 90%. This means that our model should be able to recognize the correct class based on the 13 input features to correctly classify the class of heart disease.

Design Considerations

The main design consideration is to develop a model that will perform as close as possible to the current state of the art on unseen data as on train data. To achieve this, we will follow the steps below:

1. Exploratory Data Analysis

a. Distribution of Heart Disease.

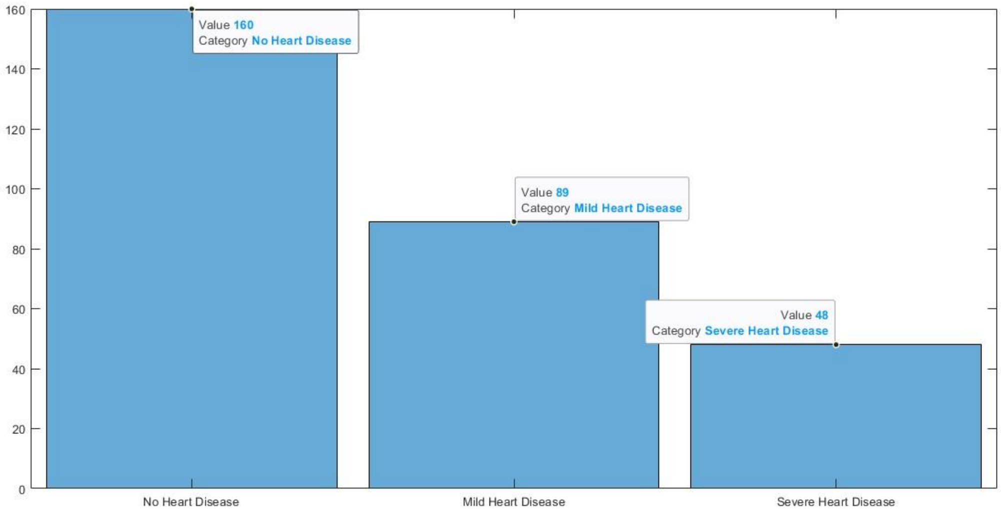


Figure 1: Distribution of heart disease

We have 160 persons with no heart disease, 89 persons with mild heart disease and 48 persons with sever heart disease. We can see that the data is not balanced for this trivariate class but if we decide to proceed by classifying at people with “heart disease’ and ‘no heart disease’, our data become fairly balance.

b. Sex Distribution

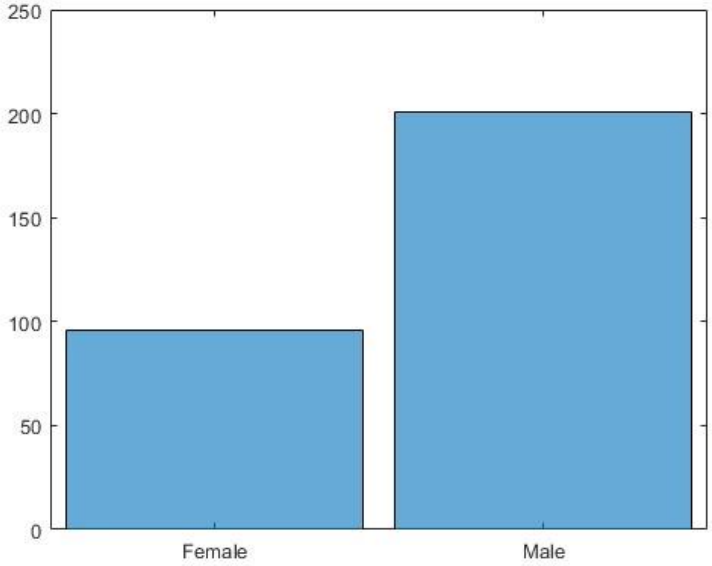


Figure 2: Sex distribution

The data set has more males than females.

c. Correlation Matrix of the dataset

To test the correlation between the dataset, we explore using the 3 types of methods available within matlab.

i. Pearson Correlation

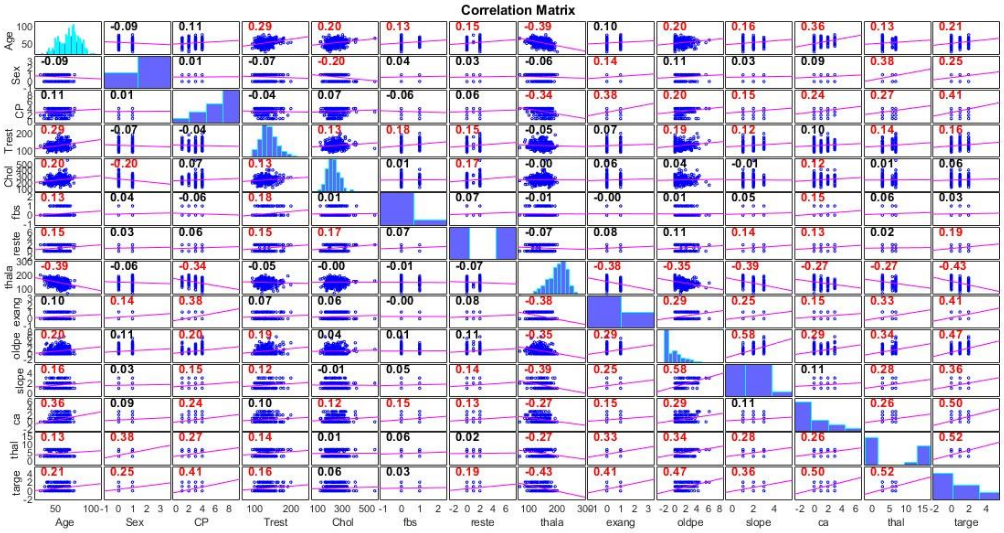


Figure 3: Pearson correlation matrix

For this, we see the following:

a. Categories: CP, EXANG, OLDPEAK, CA and THAL all have correlation above 0.40 with our target class.
b. Category chols and fbs are least correlated with our targets.
c. All other categories have some significant correlation with the target class.

ii. Spearman

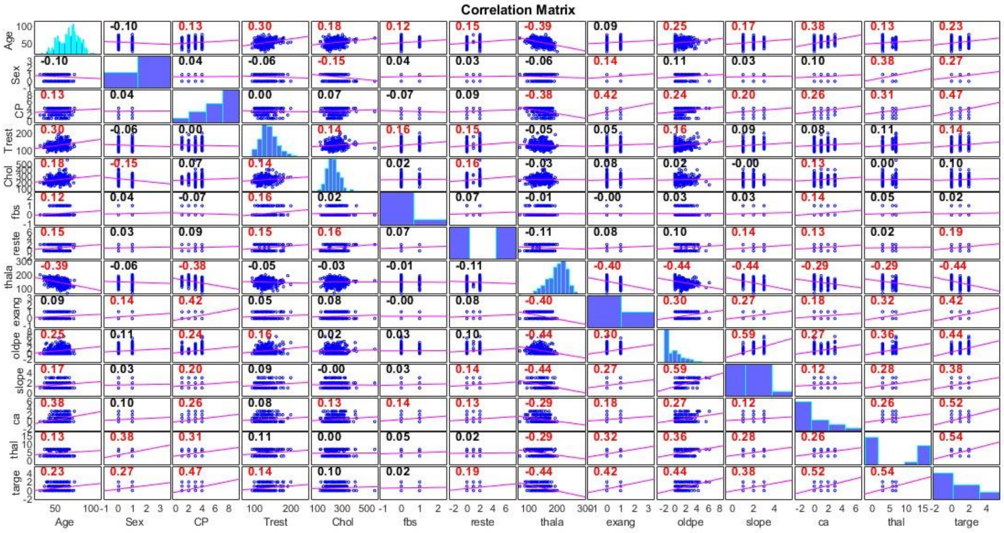


Figure 4:Spearman correlation matrix

For this, we see that the following:

a. Categories: CP, EXANG, OLDPEAK, CA and THAL all have correlation of greater than or equal to 0.40 with our target class.
b. Category chols and fbs are least correlated with our targets.
c. All other categories have some significant correlation with the target class.

iii. Kendall

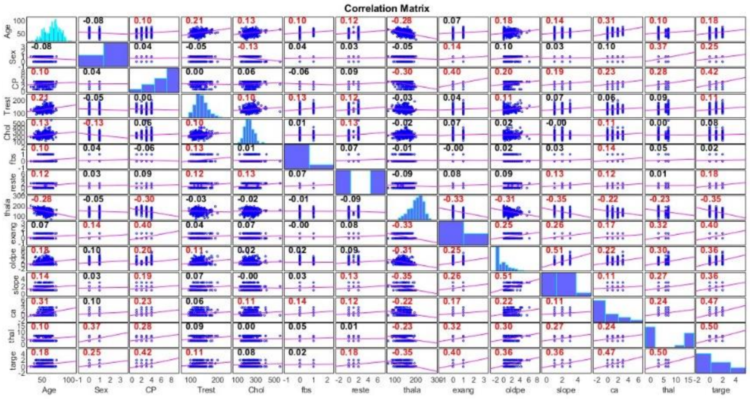


Figure 5:Kendall correlation matrix

For this, we see that the following:

a. categories: CP, EXANG, CA and THAL all have correlation of greater than or equal to 0.40 with our target class.
b. Category chols and fbs are least correlated with our targets.
c. All other categories have some significant correlation with the target class.

For correlation, we see that category ‘chols’ and ‘fbs’ which represent, the cholesterol and fasting blood sugar are not correlated with the target. We will use this knowledge to select the features that will be used for training our ANN classifier.

d. Histograms of continuous values.

For this, I used two different data tables to help understand the distribution properly. In one, I used the data as it was and in the other, I combined the class ‘mild’ and class ‘severe’ to one class ‘diseased heart’. See below plots.

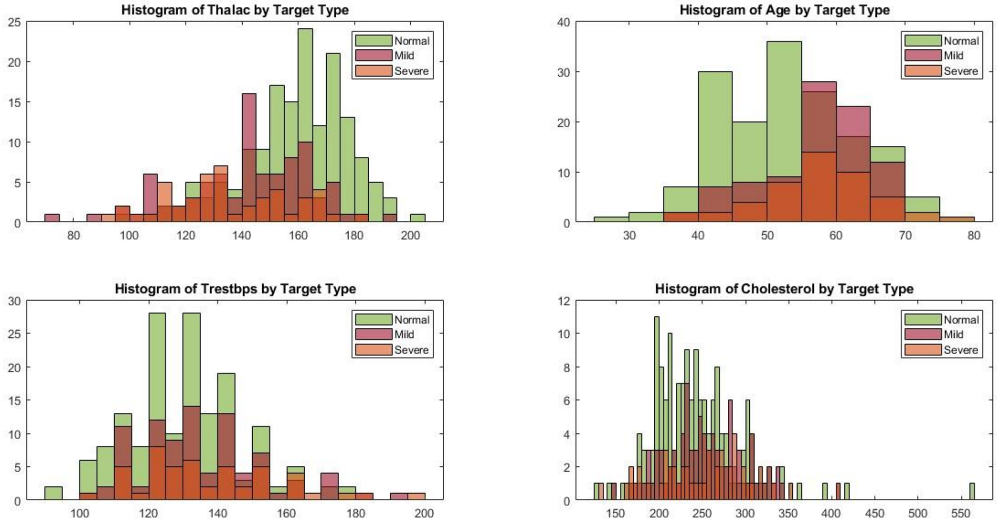


Figure 6: Histogram of continuous values.

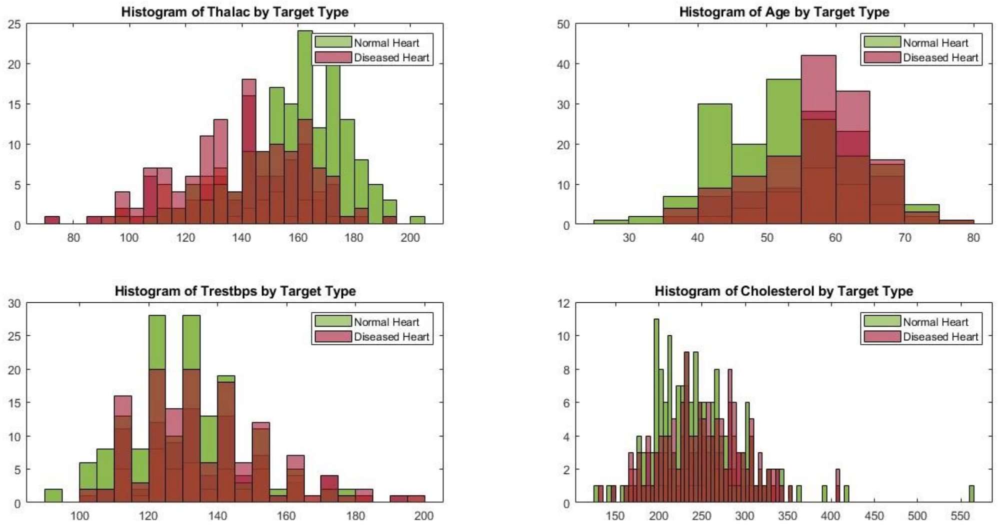


Figure 7:Histogram of other continuous values

From both plots, we can infer the following:

a. A 'thalac' value less than 160 might be an indication of a heart condition.
b. A 'Trestbps' value of over 120 might be a serious indicator of a heart disease.
c. A 'chol' value of above 180 might be a serious indicator of a heart disease.
d. People above 50 are more likely to have a heart disease.

e. Scatterplots.

We will use scatterplots to find any outliers in the data.

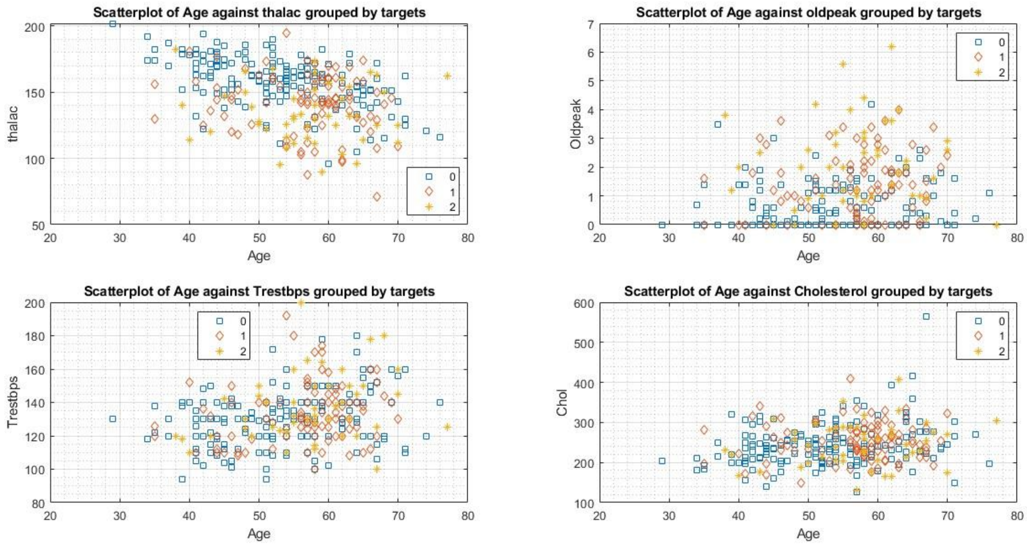


Figure 8: Scatter plot

2. Feature Selection.

   We need to select features that are important so that we can use them in the ANN network. We have already used some methods previously that checked the correlation of the input features and the output. We used the spearman, Kendall, and Pearson methods, which are numerical methods. We will use other methods and compare the results. The methods we will use are as below:

   a. Correlation-based Feature Selection.
   b. Ensemble Feature Selection.

a. Correlation-based Feature Selection:

This method uses the Pearson's correlation coefficient to evaluate the worth of each feature by computing the correlation between the feature and the target variable and between other features in the dataset.

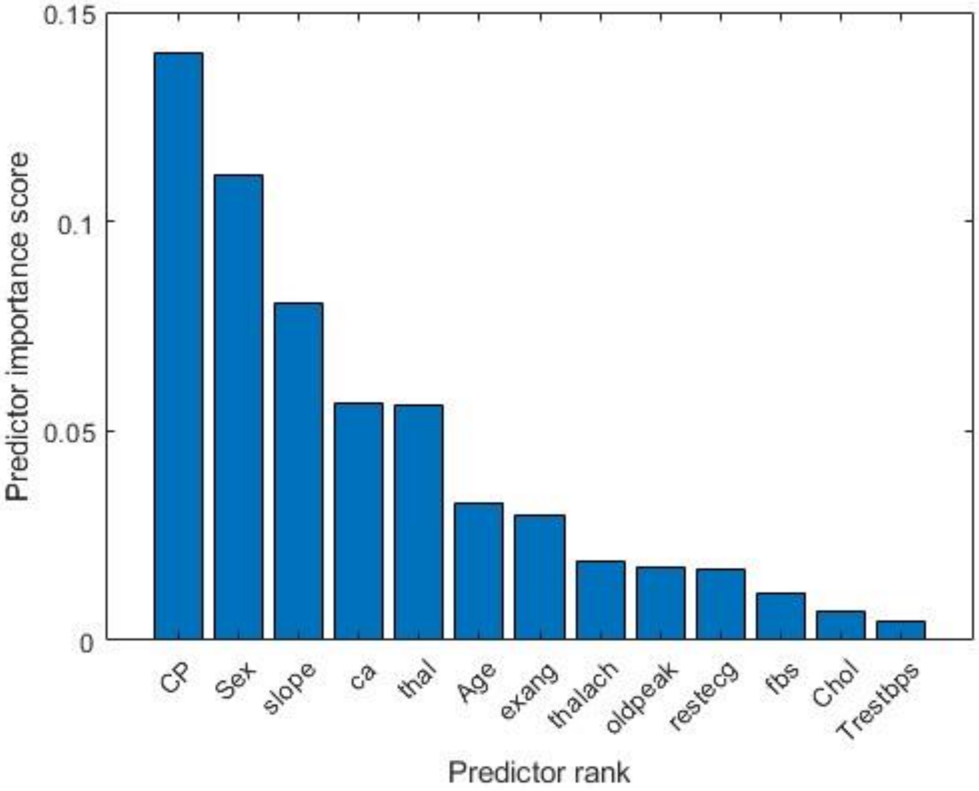


Figure 9:Correlation-based Feature Selection

We can see that the five most important features are CP, SEX, SLOPE CA and THAL in this case.

b. Ensemble feature selection:

This is a machine learning technique that combines a variety of feature selection methods to enhance the accuracy and robustness of feature selection. The idea is to create a diverse set of features subsets. Then evaluate each using a machine learning algorithm. The feature sets that performed well are then combined to form a final set of features.

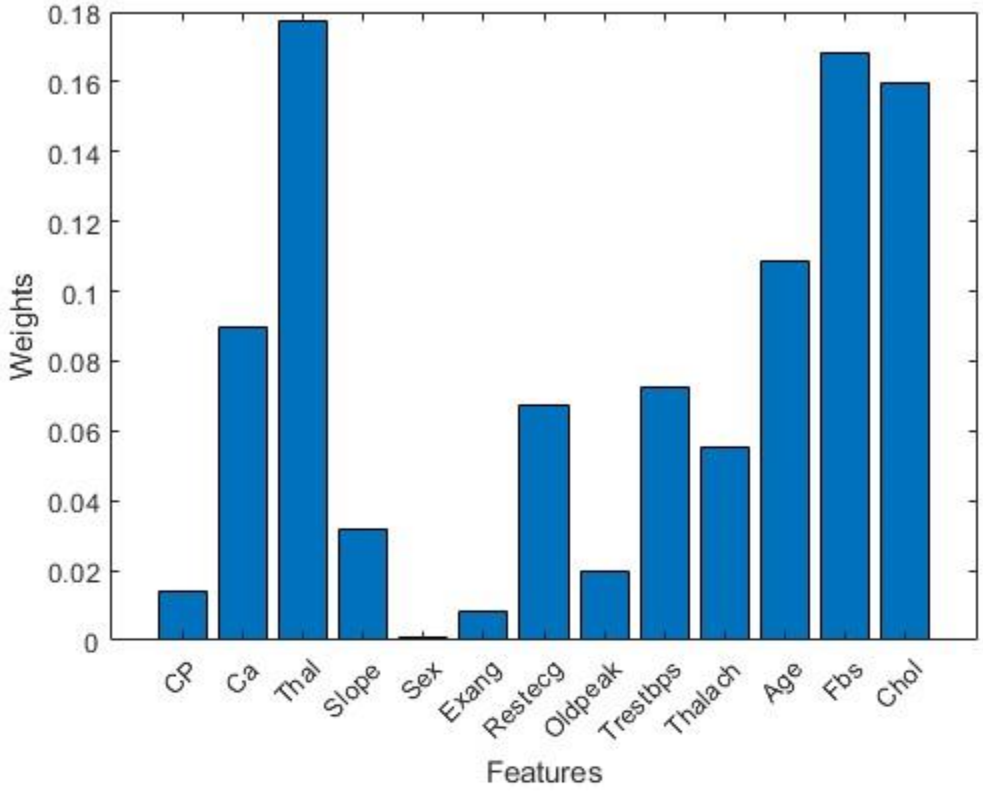


Figure 10: Ensemble feature selection

We see that the five most important features using this method are CA, THAL, AGE, FBS, CHOL.

For our model, we will combine both features from these two methods to build our features list. We will use the following features: CP, SEX, SLOPE, AGE, FBS, CHOL, CA and THAL.

3. Data Pre-Processing:

Here we are going to combine several techniques to clean and transform our data into a suitable format suitable for training machine learning model. We will be using normalization for our input features.

Rajeswari et al (2020) studied the performance of various data normalization techniques on different heart disease datasets. They looked at 3 main methods: Min-Max, z-score, and decimal scaling normalization, all of which enhance the model performance. Their results show that the effect of normalization techniques significantly improves the classification accuracy of the models. Their results also showed that the Min-Max method had the best performance across all the datasets used.

Min-max gives a linear transformation to normalize the raw dataset into a new fixed range of 0 to 1. This method reserves the connection between the input value and the scaled value. The formula for the transformation is as below:

$$pt = \frac{(patterns - \min(patterns)}{(\max(patterns) - \min\,(patterns))}$$

4. Test set and Training Set preparation.
   After our data pre-processing is completed, we split the data into a training set and a testing set. I use 80% of the data in the training set and 10% for validation and the last 10% for testing.

5. Design of multi-layer perceptron.

   Our dataset consists of thirteen input features and three output features. This means that the multi-layer perceptron will have thirteen input nodes, or one for each feature that we eventually decide to use. The output layer will also have one node each representing each possible classification: 'normal, 'mild heart disease' and 'severe heart disease' or 'normal heart' and 'diseased heart' in the second case.

   We will use the 'feedforwardnet' command in MATLAB to build this network and tune some if its parameters to achieve good classification. We will consider the following:

   a. **Hidden layers:** The number and size of the hidden layer varies depending on the complexity of the problem. A good rule of thumb is to start with one or even two hidden layers and gradually increase until desired performance is achieved. Ajam (2015) proposed a network with thirteen input neurons, a hidden layer with 20 neurons and an output layer of one neuron. The results showed that the network was able to classify 88% of the cases in the testing set.
   b. **Activation Function:** The type of activation function to be used in the hidden layers need to be chosen based on the specific requirements of our problem. Antar, Lotaibi and Ghamdi(2021) studied the use of various network structures with different activation function and compared the performance of the model. Their model with 3 hidden layers with ReLU activation functions [6 3 3] and output layer with the sigmoid function achieved an accuracy of 85.07%. Ajam (2015) used the sigmoid function in the hidden layers and linear transfer function in the output layers.
   c. **Loss Function:** This is typically used to optimize the network. Antar, Lotaibi and Ghamdi(2021) used the popular Mean-Square Error (MSE) in their seminal work that achieved 85.07%. Ajam (2015) also used this same technique to achieve the performance of 88%.
   d. **Optimization algorithm:** The choice of the algorithms used here to train the network can have a significant impact of the performance of the network. Antar, Lotaibi and Ghamdi(2021) used the Adam algorithm for their paper; Ajam (2015) used the Feed forward Back Propagation learning algorithm (trainlm)

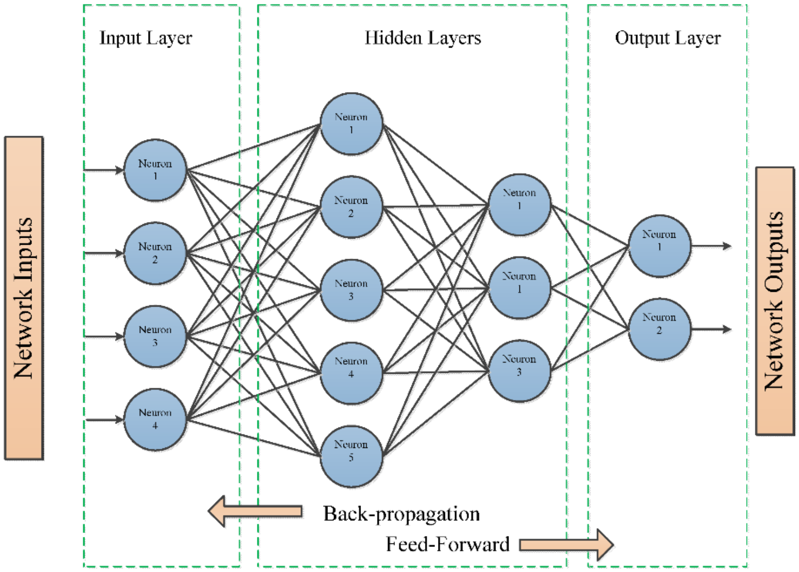


Figure 11: Multi-layer perceptron

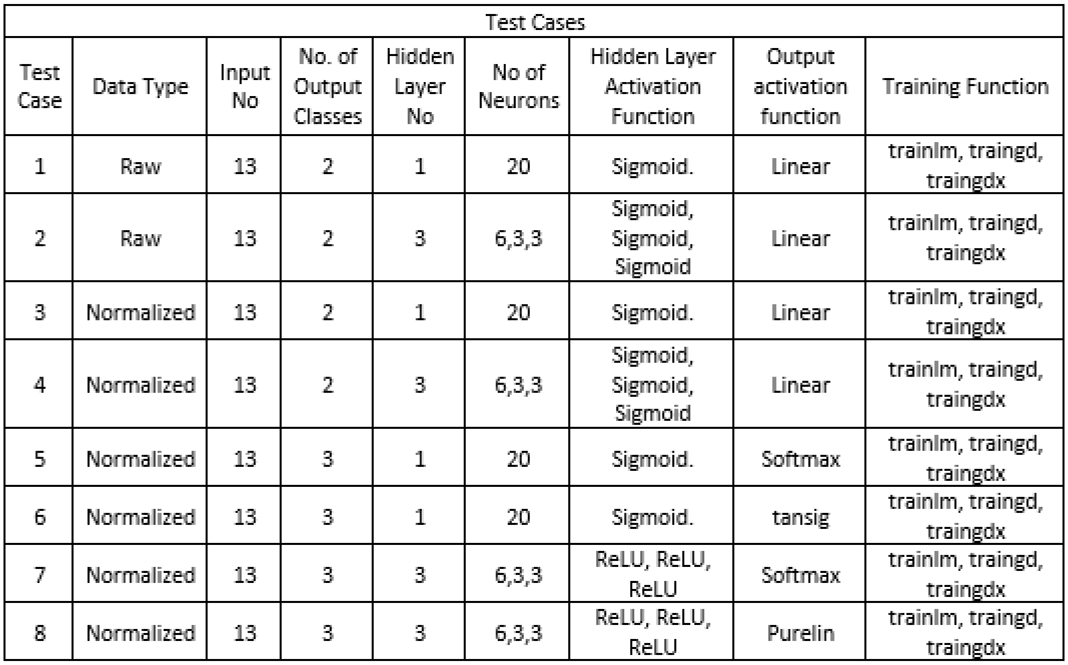
Abdolrasol, M.G et al.

e. **Training:** After designing, the next thing is to train the network. MATLAB allows several different commands to specify how the network should be trained, ‘dividetrain’ and ‘dividerand’, each used for different purposes. Dividetrain when used assigns all targets to the training set and no targets to the validation or test sets while dividerand takes the number of targets and divides this up based on the ration for training, validation and testing specified by the user. Divideblock separate targets into three sets: training, validation and testing based on the user inputs.

f. **Simulating and Testing:** After the training of the network is done, we will then simulate the network and give it data it has never seen before to test its performance.

Methodology:

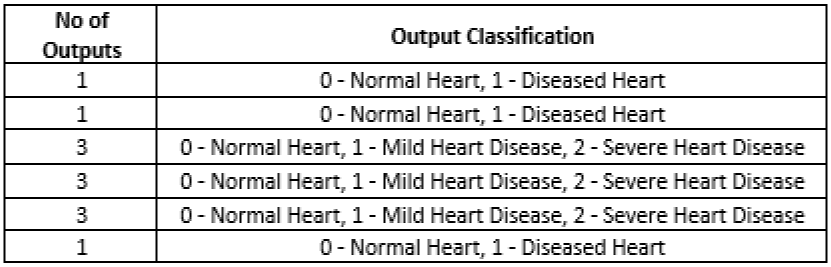
For this report, the proposed models are as below:



Test Cases								
Test Case	Data Type	Input No	No. of Output Classes	Hidden Layer No	No of Neurons	Hidden Layer Activation Function	Output activation function	Training Function
1	Raw	13	2	1	20	Sigmoid.	Linear	trainlm, traingd, traingdx
2	Raw	13	2	3	6,3,3	Sigmoid, Sigmoid, Sigmoid	Linear	trainlm, traingd, traingdx
3	Normalized	13	2	1	20	Sigmoid.	Linear	trainlm, traingd, traingdx
4	Normalized	13	2	3	6,3,3	Sigmoid, Sigmoid, Sigmoid	Linear	trainlm, traingd, traingdx
5	Normalized	13	3	1	20	Sigmoid.	Softmax	trainlm, traingd, traingdx
6	Normalized	13	3	1	20	Sigmoid.	tansig	trainlm, traingd, traingdx
7	Normalized	13	3	3	6,3,3	ReLU, ReLU, ReLU	Softmax	trainlm, traingd, traingdx
8	Normalized	13	3	3	6,3,3	ReLU, ReLU, ReLU	Purelin	trainlm, traingd, traingdx

Table 2: Proposed models for testing

We will also compare the performance of each design category based on the use of raw data or normalized data to see which performance is better. For the instance with different outputs, the classification is as below.



No of Outputs	Output Classification
1	0 - Normal Heart, 1 - Diseased Heart
1	0 - Normal Heart, 1 - Diseased Heart
3	0 - Normal Heart, 1 - Mild Heart Disease, 2 - Severe Heart Disease
3	0 - Normal Heart, 1 - Mild Heart Disease, 2 - Severe Heart Disease
3	0 - Normal Heart, 1 - Mild Heart Disease, 2 - Severe Heart Disease
1	0 - Normal Heart, 1 - Diseased Heart

Table 3: Output design category

For testing, our data is split into training set, validation set, and a testing set. The ratio is **80% for training, 10% for validation and 10% for testing.**

Measuring Performance

a. **Accuracy:** this is defined as the percentage of correct predictions for the data. It is calculated by dividing the number of correct predictions by the total number of predictions.

$$Accuracy = \frac{True_{positive} + True_{negative}}{True_{positive} + True_{negative} + False_{positive} + False_{negative}}$$

b. **Recall**: This is defined as the fraction examples which are predicted to belong to a class with respect to all the examples that really belong to that class.

$$Recall = \frac{True_{positive}}{True_{positive} + False_{negative}}$$

c. **Precision:** this is defined as the fraction of true positives among all the examples which were predicted to belong to a certain class.

$$Precision = \frac{True_{positive}}{True_{positive} + False_{positive}}$$

Precision and recall are going to be important for our models because the data is not even distributed. As applied to our case, recall ensures that we are not overlooking people who might have the heart disease while precision ensures that we are not misclassifying people as having heart disease when in fact they do not.

d. **F1-score:** this is defined as a measure of a model's accuracy. It is calculated by combining the precision and accuracy of the model. It is also the harmonic mean of the precision and recall. The formula is as below:

$$\text{F1-score} = 2.\frac{\text{Precision}.\,\text{Recall}}{\text{Precision} + \text{Recall}}$$

When working with classification models like the ones we have proposed above, there are 4 types of outcomes that can generally occur.

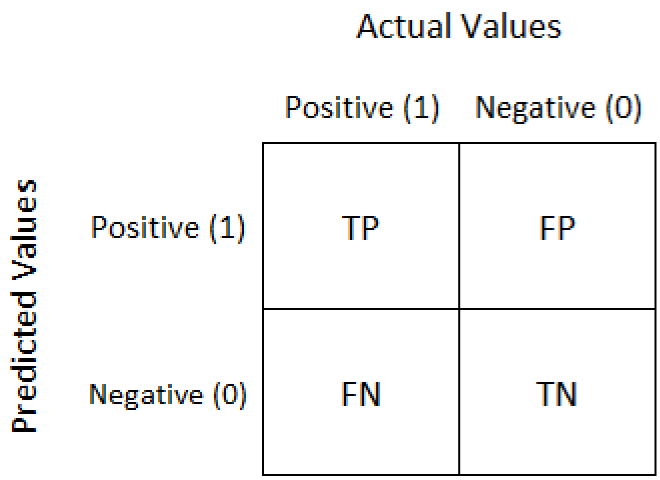
i. **True positives (TP):** this is when the model predicts an observation to belong to a class and it belongs to that class.

ii. **True negatives (TN):** this is when the model predicts an observation does not belong to a class and it does not belong to that class.

iii. **False positives (FP):** this is when the model predicts an observation to belong to a class and it does not actually belong to that class.

iv. **False negatives (FN):** this is when the model predicts an observation does not belong to a particular when it does belong to it.

These four metrics are then used to plot a confusion matrix for the classification of the model as in the example below.



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 12 Confusion Matrix:

https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

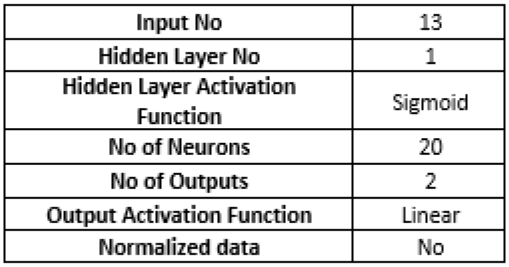
Experimental Results:

Section 1: Raw Data and 2 Output Class

For this section the raw data is fed into the network with the hyperparameter settings and classification is done. The only change to the input data is the change in output class. Because the data is not balanced, Class 1 and Class 2 are combined to one class called 'diseased heart' while class 0 remains as 'normal heart'. The results for the different hyperparameter will be discussed at the end of this section.

Test Case 1:

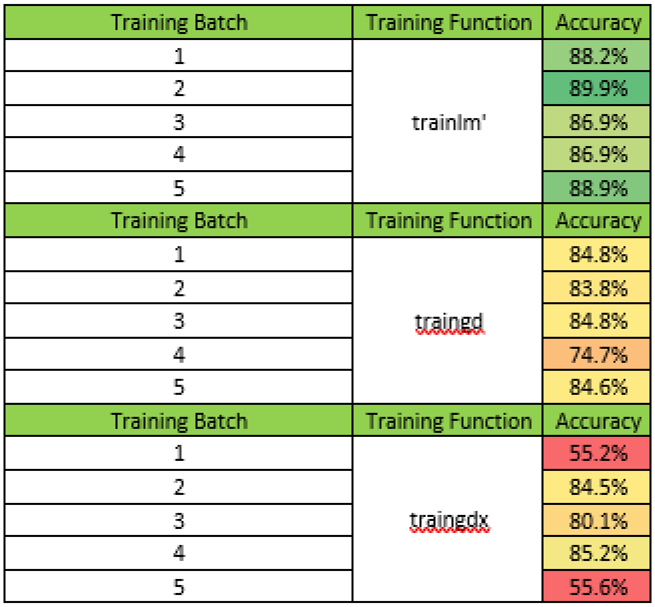
For this case, we will use the hyperparameters in the table below to build the network. We will try 3 different training function, 'trainlm', 'traingd' and 'traingdx'.



Input No	13
Hidden Layer No	1
Hidden Layer Activation Function	Sigmoid
No of Neurons	20
No of Outputs	2
Output Activation Function	Linear
Normalized data	No

Table 4: Hyperparameters for testcase 1

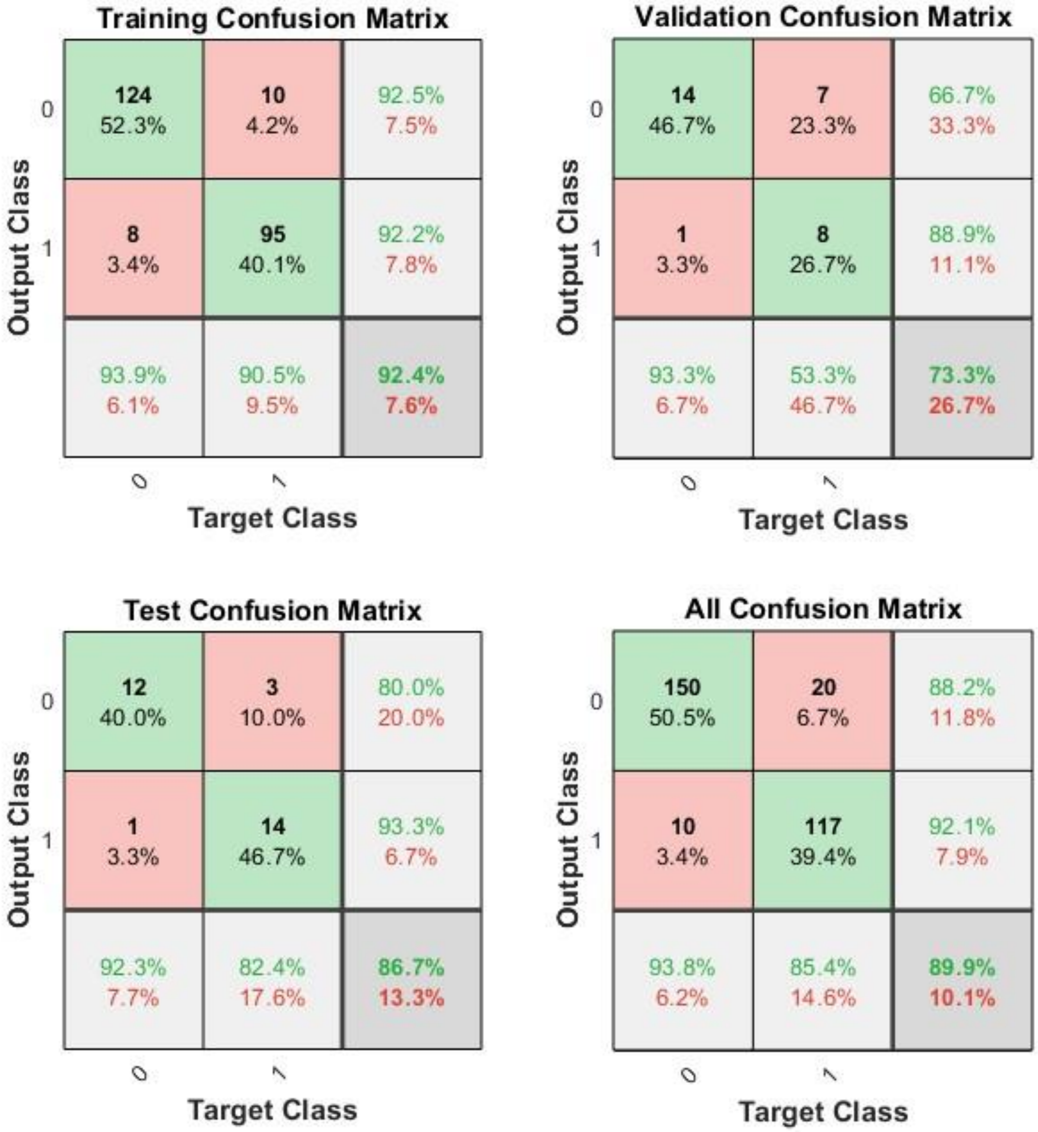
The results for this configuration are as below:



Training Batch	Training Function	Accuracy
1	trainlm'	88.2%
2		89.9%
3		86.9%
4		86.9%
5		88.9%
Training Batch	Training Function	Accuracy
1	traingd	84.8%
2		83.8%
3		84.8%
4		74.7%
5		84.6%
Training Batch	Training Function	Accuracy
1	traingdx	55.2%
2		84.5%
3		80.1%
4		85.2%
5		55.6%

Table 5: Results for testcase 1

We see that the best performance with this set of hyperparameter is 89.9% gotten with the 'trainlm' function.



Training Confusion Matrix

Output Class \ Target Class	0	1	
0	124 52.3%	10 4.2%	92.5% 7.5%
1	8 3.4%	95 40.1%	92.2% 7.8%
	93.9% 6.1%	90.5% 9.5%	92.4% 7.6%

Validation Confusion Matrix

Output Class \ Target Class	0	1	
0	14 46.7%	7 23.3%	66.7% 33.3%
1	1 3.3%	8 26.7%	88.9% 11.1%
	93.3% 6.7%	53.3% 46.7%	73.3% 26.7%

Test Confusion Matrix

Output Class \ Target Class	0	1	
0	12 40.0%	3 10.0%	80.0% 20.0%
1	1 3.3%	14 46.7%	93.3% 6.7%
	92.3% 7.7%	82.4% 17.6%	86.7% 13.3%

All Confusion Matrix

Output Class \ Target Class	0	1	
0	150 50.5%	20 6.7%	88.2% 11.8%
1	10 3.4%	117 39.4%	92.1% 7.9%
	93.8% 6.2%	85.4% 14.6%	89.9% 10.1%

Figure 13: Confusion matrix for testcase 1

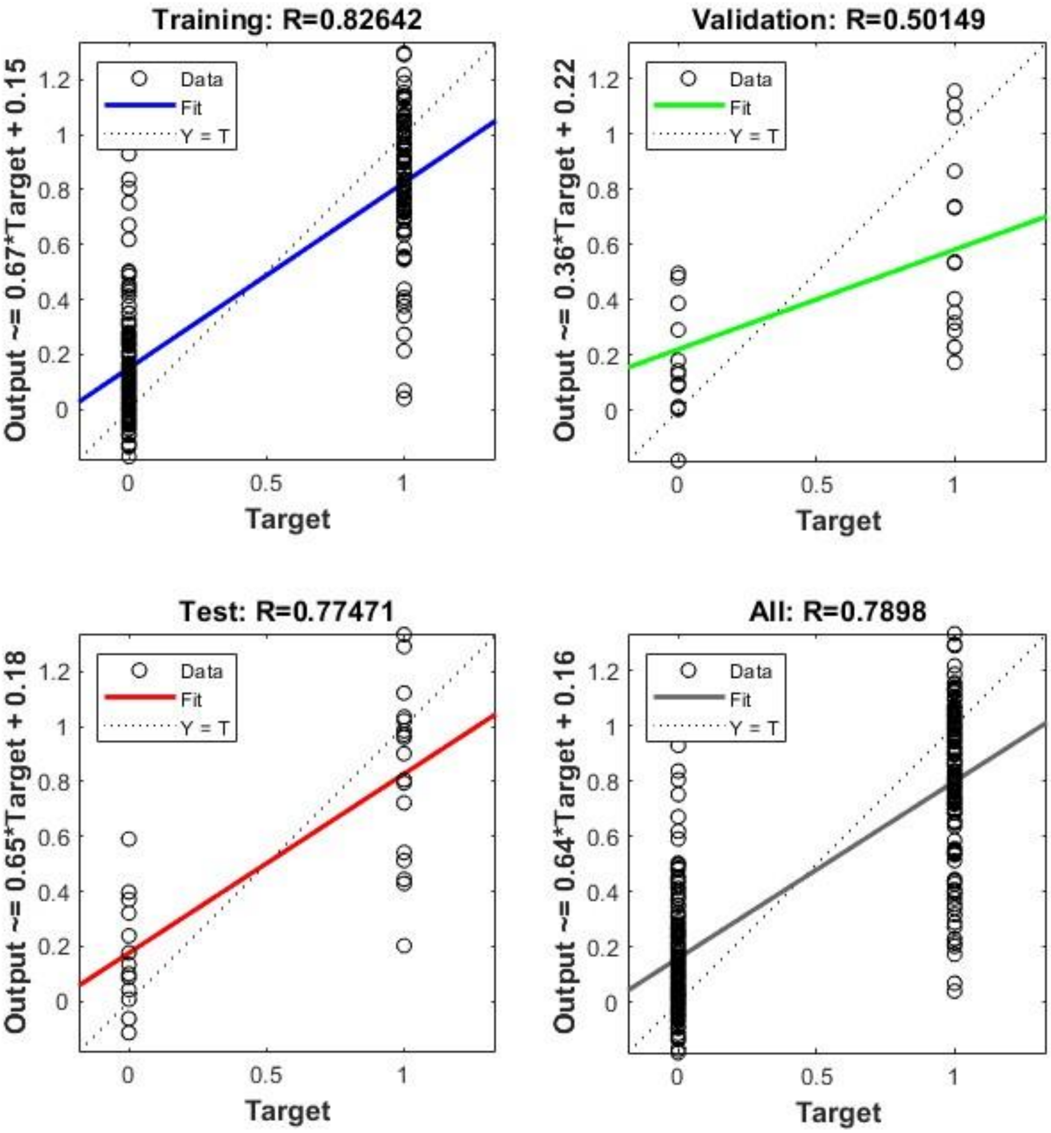


Figure 14: Regression plot for testcase 1

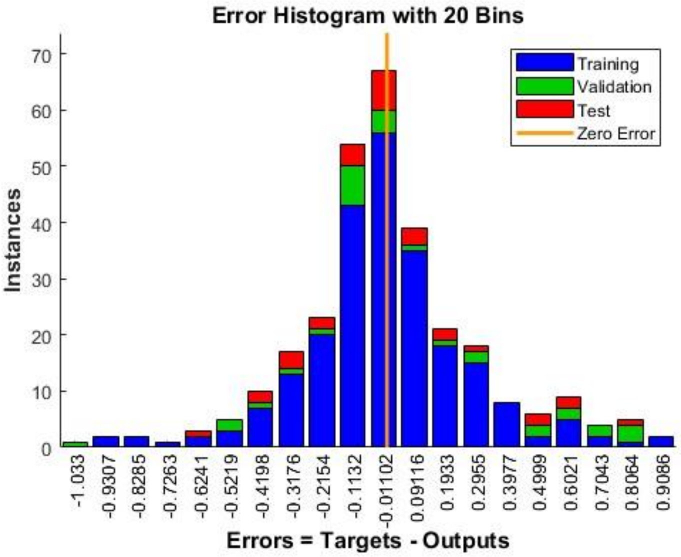


Figure 15: Error histogram testcase 1

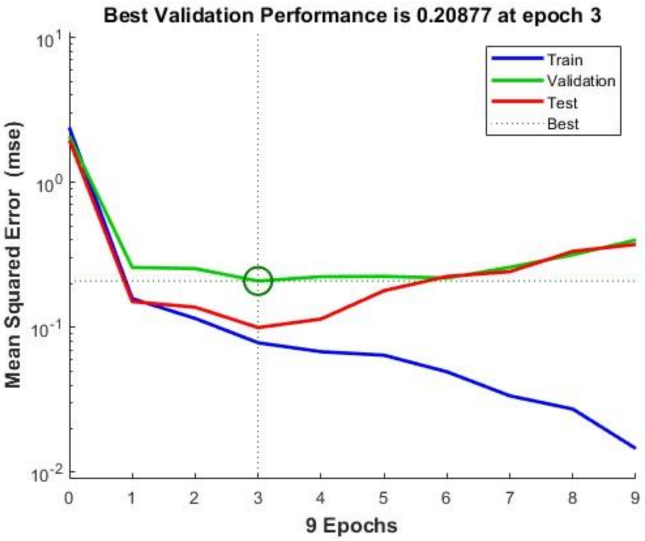
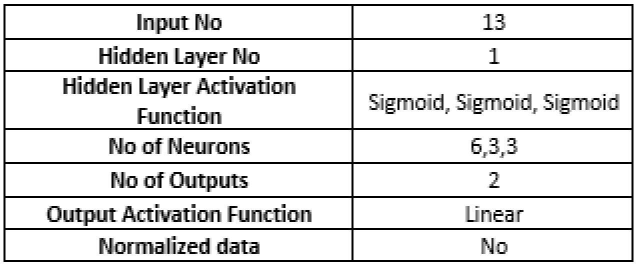


Figure 16: Performance curve

Test Case 2:

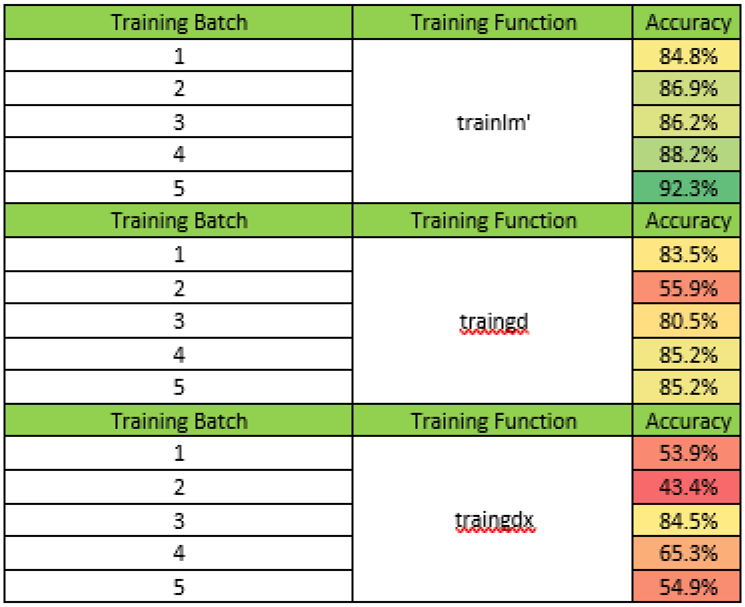
For this case, we will use the hyperparameters in the table below to build the network. Again, we will try 3 different training function, ‘trainlm’, ‘traingd’ and ‘traingdx’.



Input No	13
Hidden Layer No	1
Hidden Layer Activation Function	Sigmoid, Sigmoid, Sigmoid
No of Neurons	6,3,3
No of Outputs	2
Output Activation Function	Linear
Normalized data	No

Table 6:Hyperparameters for testcase 2

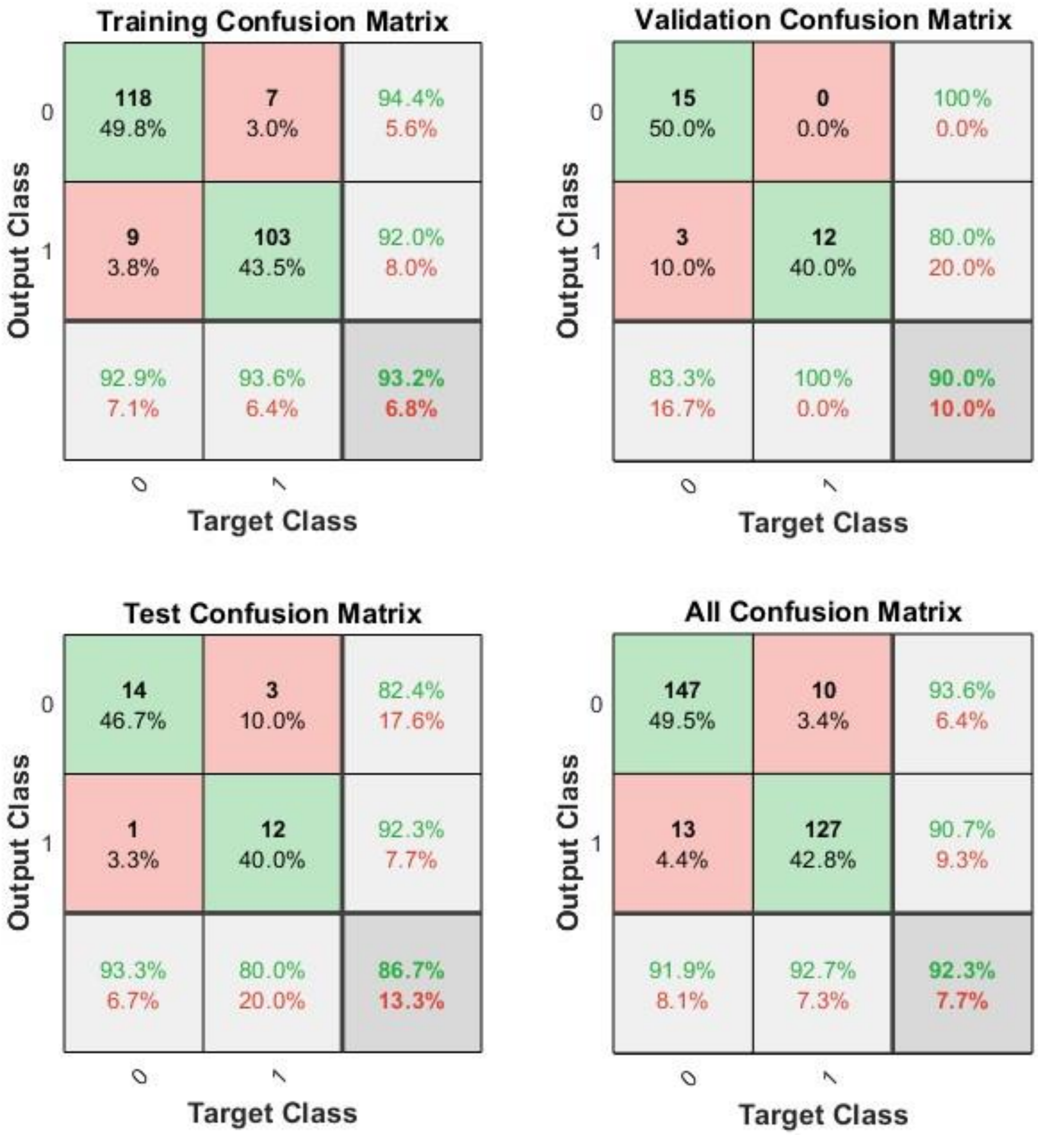
The results for this configuration are as below:



Training Batch	Training Function	Accuracy
1	trainlm'	84.8%
2		86.9%
3		86.2%
4		88.2%
5		92.3%
Training Batch	Training Function	Accuracy
1	traingd	83.5%
2		55.9%
3		80.5%
4		85.2%
5		85.2%
Training Batch	Training Function	Accuracy
1	traingdx	53.9%
2		43.4%
3		84.5%
4		65.3%
5		54.9%

Table 7: Results for testcase 2

Again, the best performance is gotten with the ‘trainlm’ training function, and the worst performance is with the ‘traingdx’ training function.



Training Confusion Matrix

Output Class	0	1	
0	118 49.8%	7 3.0%	94.4% 5.6%
1	9 3.8%	103 43.5%	92.0% 8.0%
	92.9% 7.1%	93.6% 6.4%	93.2% 6.8%

Target Class

Validation Confusion Matrix

Output Class	0	1	
0	15 50.0%	0 0.0%	100% 0.0%
1	3 10.0%	12 40.0%	80.0% 20.0%
	83.3% 16.7%	100% 0.0%	90.0% 10.0%

Target Class

Test Confusion Matrix

Output Class	0	1	
0	14 46.7%	3 10.0%	82.4% 17.6%
1	1 3.3%	12 40.0%	92.3% 7.7%
	93.3% 6.7%	80.0% 20.0%	86.7% 13.3%

Target Class

All Confusion Matrix

Output Class	0	1	
0	147 49.5%	10 3.4%	93.6% 6.4%
1	13 4.4%	127 42.8%	90.7% 9.3%
	91.9% 8.1%	92.7% 7.3%	92.3% 7.7%

Target Class

Figure 17: Confusion matrix testcase 2

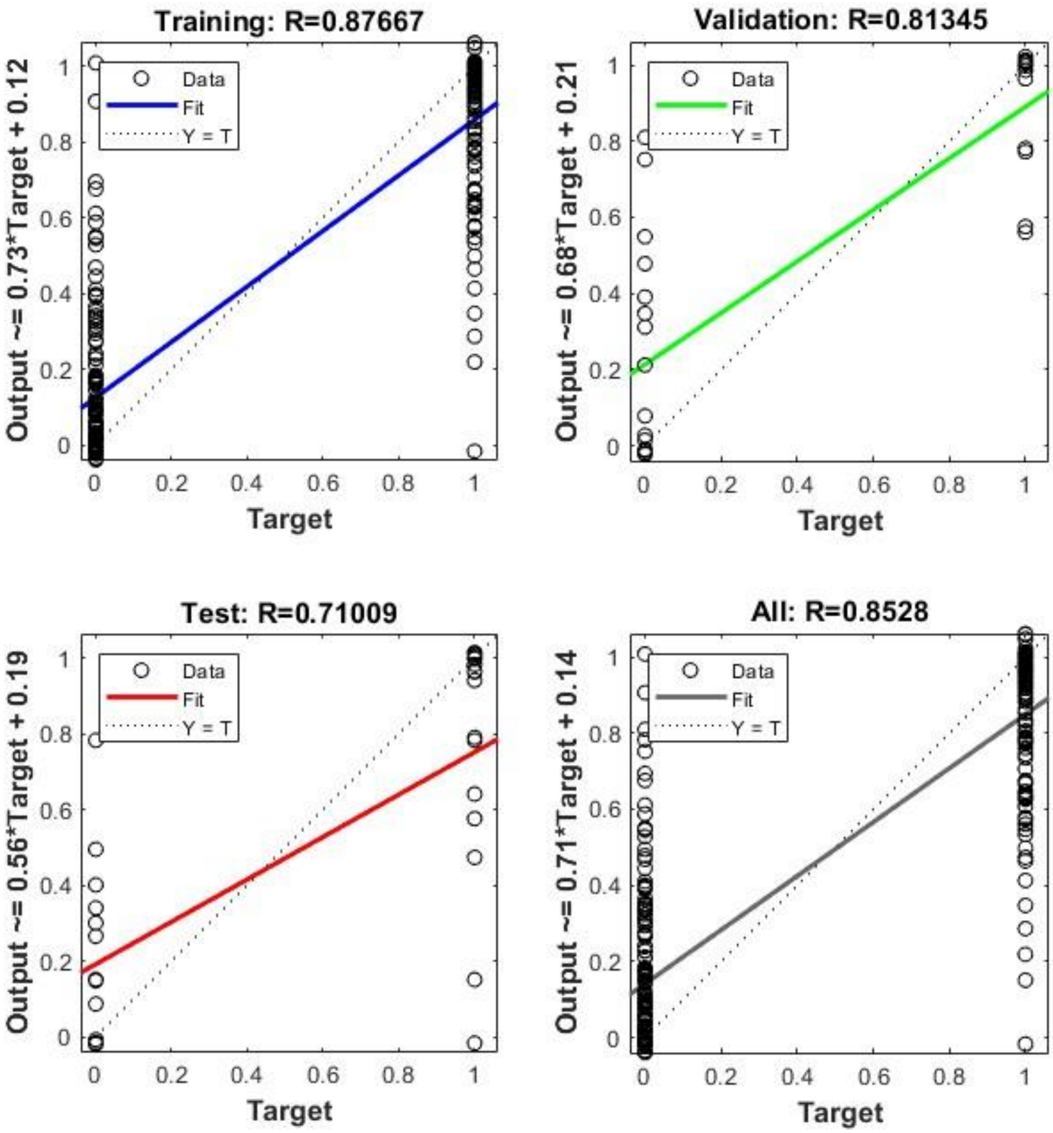


Figure 18: Regression for testcase 2

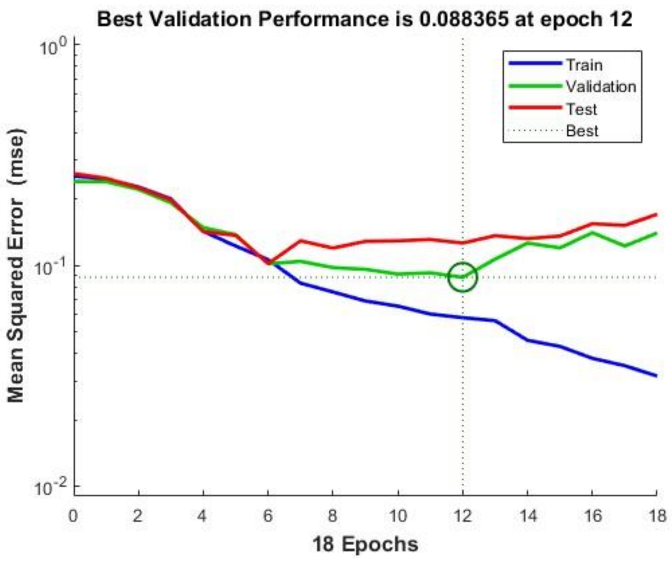


Figure 19: Performance curve testcase 2

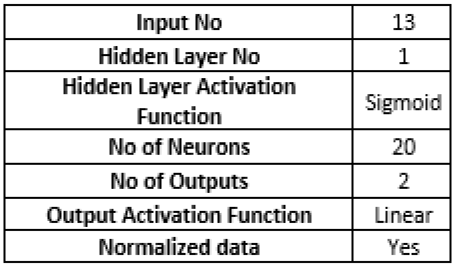
The performance graphs and confusion matrix for the best cases can be found in the appendix.

Section 2: Normalized Data and 2 Output Class

For this section the data is normalized using the min-max algorithm then fed into the network with the hyperparameter settings and classification is done. As in the first scenario, we maintain the two classes that were created to balance the dataset.

Test Case 3:

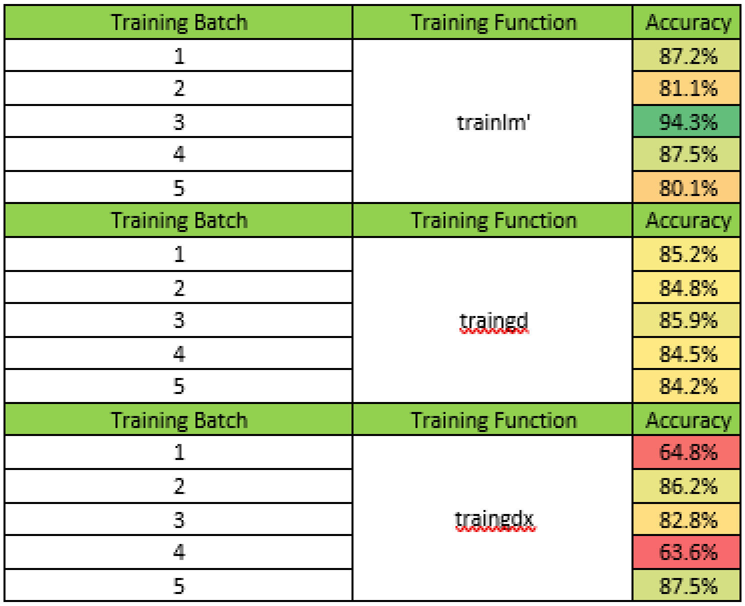
The hyperparameter settings are as below:



Input No	13
Hidden Layer No	1
Hidden Layer Activation Function	Sigmoid
No of Neurons	20
No of Outputs	2
Output Activation Function	Linear
Normalized data	Yes

Table 8:Hyperparameters for testcase 3

The results from this are as below:



Training Batch	Training Function	Accuracy
1	trainlm'	87.2%
2		81.1%
3		94.3%
4		87.5%
5		80.1%
Training Batch	Training Function	Accuracy
1	traingd	85.2%
2		84.8%
3		85.9%
4		84.5%
5		84.2%
Training Batch	Training Function	Accuracy
1	traingdx	64.8%
2		86.2%
3		82.8%
4		63.6%
5		87.5%

Table 9: Results for testcase 3

Again, the best performance is gotten with the ‘trainlm’ training function while the ‘traingdx’ function performed poorly. The results from the ‘traingd’ had the least deviation between best result and worst result.

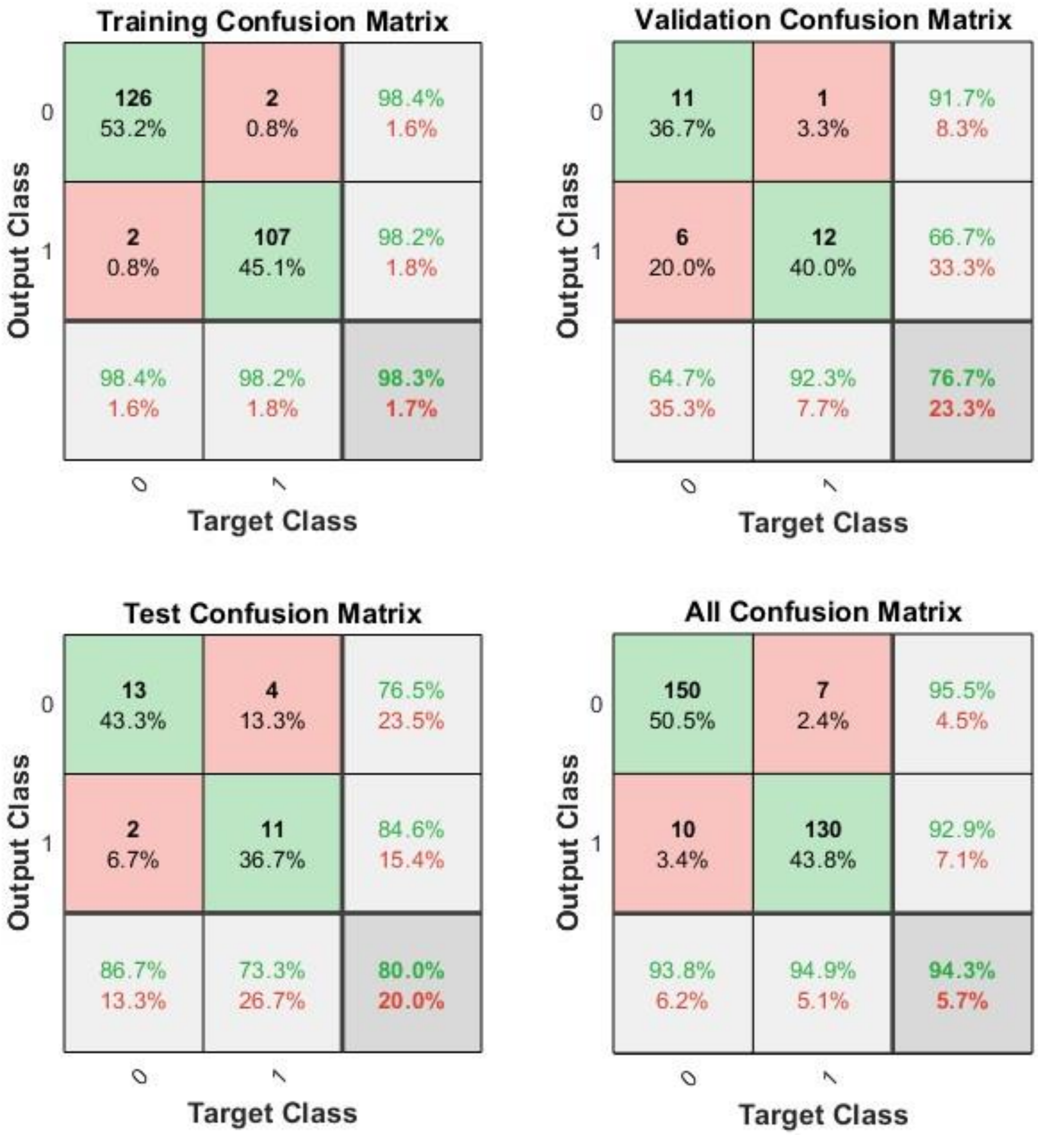


Figure 20: Confusion matrix testcase 3

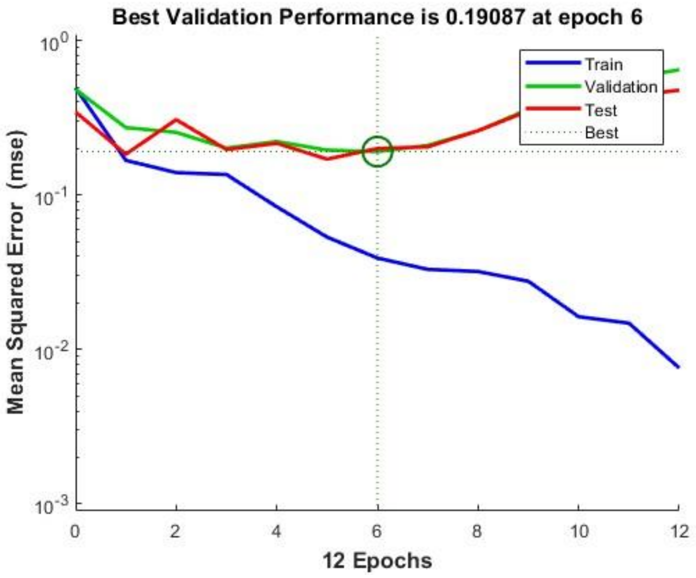
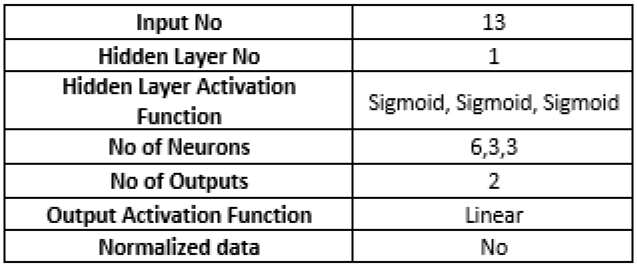


Figure 21: Performance curve testcase 3

Test Case 4:

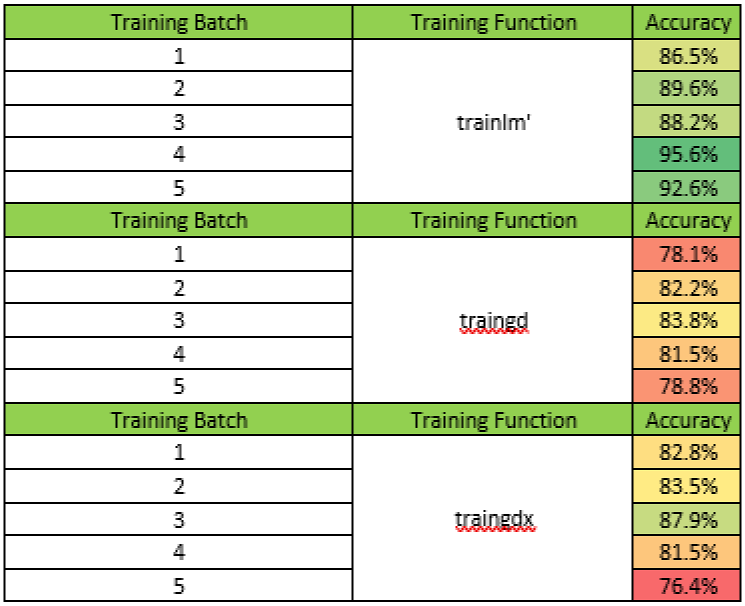
The hyperparameter settings are as below:



Input No	13
Hidden Layer No	1
Hidden Layer Activation Function	Sigmoid, Sigmoid, Sigmoid
No of Neurons	6,3,3
No of Outputs	2
Output Activation Function	Linear
Normalized data	No

Table 10:Hyperparameters for testcase 4

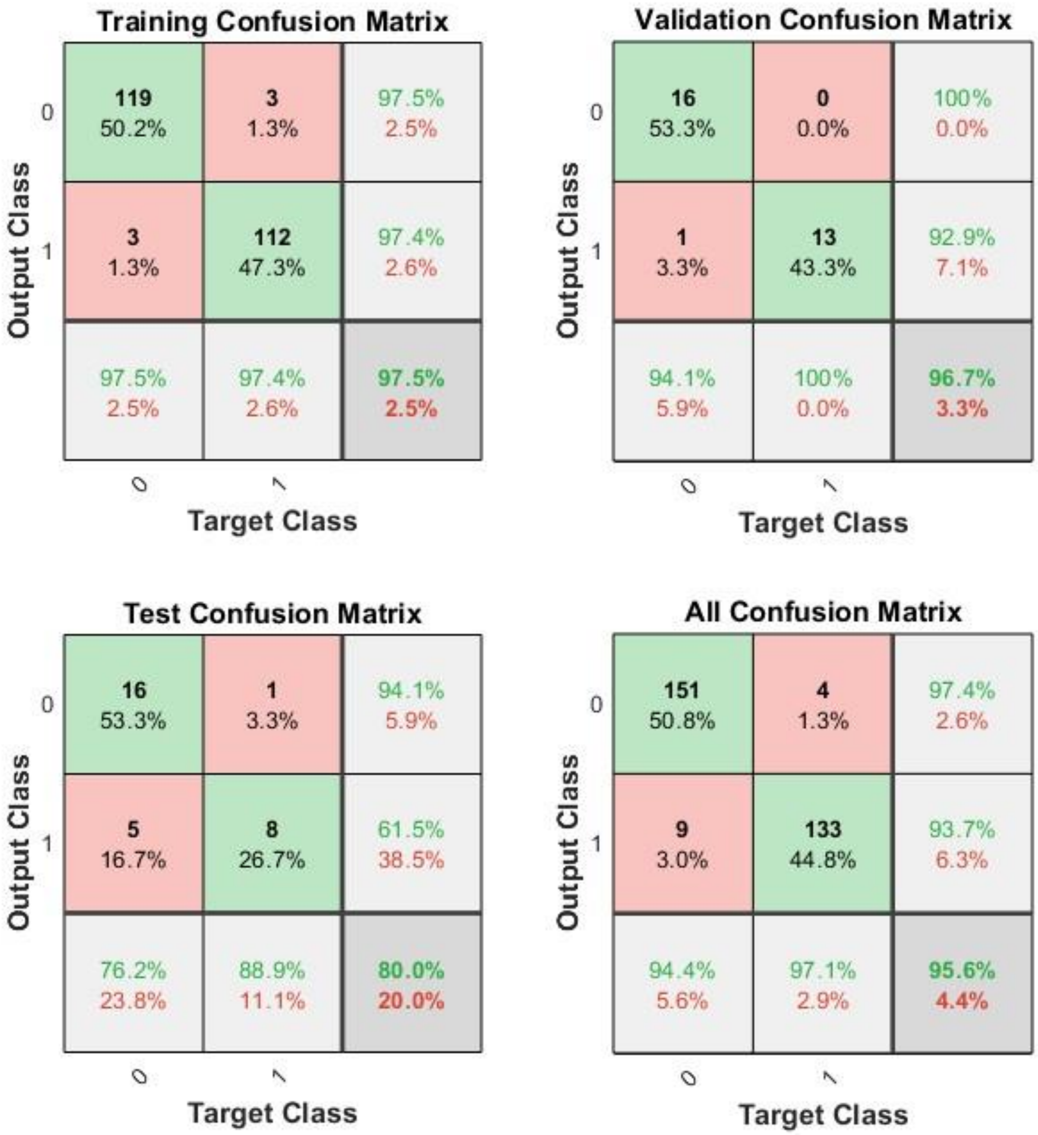
The results for this are as below:



Training Batch	Training Function	Accuracy
1	trainlm'	86.5%
2		89.6%
3		88.2%
4		95.6%
5		92.6%
Training Batch	Training Function	Accuracy
1	traingd	78.1%
2		82.2%
3		83.8%
4		81.5%
5		78.8%
Training Batch	Training Function	Accuracy
1	traingdx	82.8%
2		83.5%
3		87.9%
4		81.5%
5		76.4%

Table 11:Results for testcase 4

Again, the performance we have seen from the previous sections is repeated. The ‘trainlm’ function has the best classification.



Training Confusion Matrix

Output Class \ Target Class	0	1	
0	119 50.2%	3 1.3%	97.5% 2.5%
1	3 1.3%	112 47.3%	97.4% 2.6%
	97.5% 2.5%	97.4% 2.6%	**97.5%** **2.5%**

Validation Confusion Matrix

Output Class \ Target Class	0	1	
0	16 53.3%	0 0.0%	100% 0.0%
1	1 3.3%	13 43.3%	92.9% 7.1%
	94.1% 5.9%	100% 0.0%	**96.7%** **3.3%**

Test Confusion Matrix

Output Class \ Target Class	0	1	
0	16 53.3%	1 3.3%	94.1% 5.9%
1	5 16.7%	8 26.7%	61.5% 38.5%
	76.2% 23.8%	88.9% 11.1%	**80.0%** **20.0%**

All Confusion Matrix

Output Class \ Target Class	0	1	
0	151 50.8%	4 1.3%	97.4% 2.6%
1	9 3.0%	133 44.8%	93.7% 6.3%
	94.4% 5.6%	97.1% 2.9%	**95.6%** **4.4%**

Figure 22: Confusion matrix testcase 4

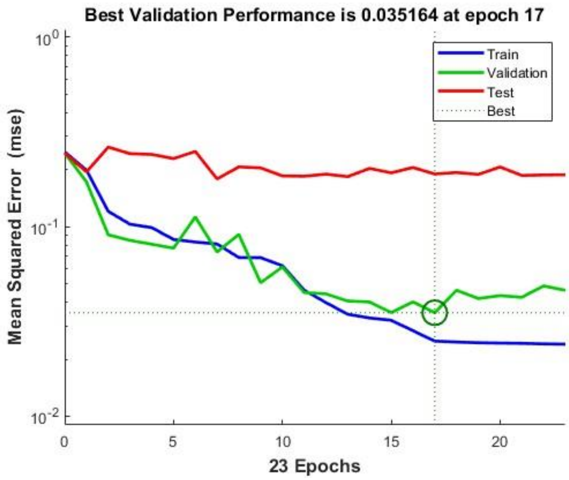
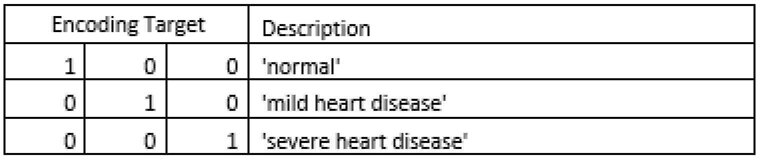


Figure 23: Performance curve testcase 4

Section 3: Normalized Data and 3 Output Class

In this section, we will repeat the classification with the criterion already used. The only difference here is the change in the output classes. We will use the data as it is with the 3 classes: 0 – normal, 1 – mild heart disease and 2 – severe heart disease. To help with the multiclass classification, we will encode the output class as below:

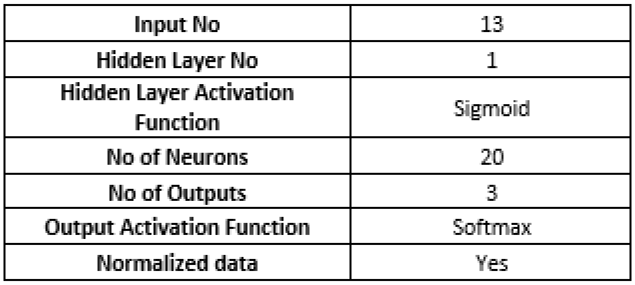


Encoding Target			Description
1	0	0	'normal'
0	1	0	'mild heart disease'
0	0	1	'severe heart disease'

Table 12: One-hot encoding for target class

Test Case 5:

The hyperparameters for this case are as below:

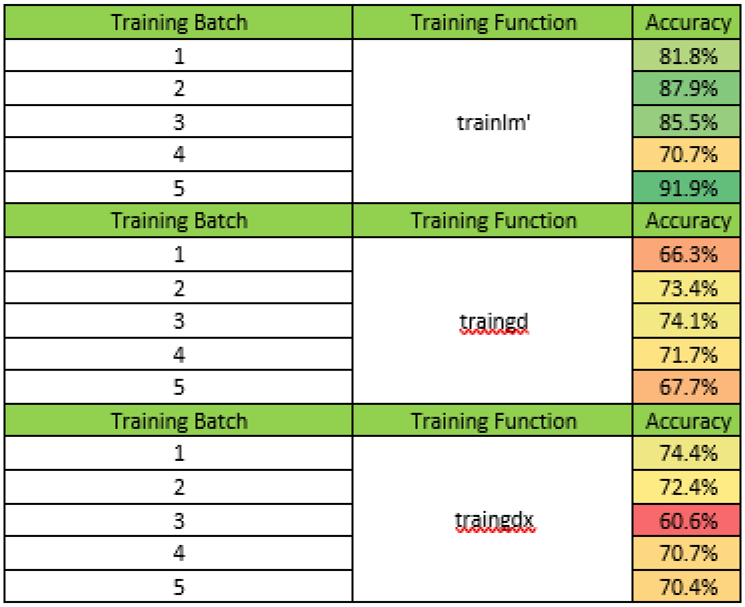


Input No	13
Hidden Layer No	1
Hidden Layer Activation Function	Sigmoid
No of Neurons	20
No of Outputs	3
Output Activation Function	Softmax
Normalized data	Yes

Table 13: Hyperparameters for testcase 5

I decided to go with normalized data as this got the best performance from the previous section. The ‘Softmax’ function was used at the output in this case.

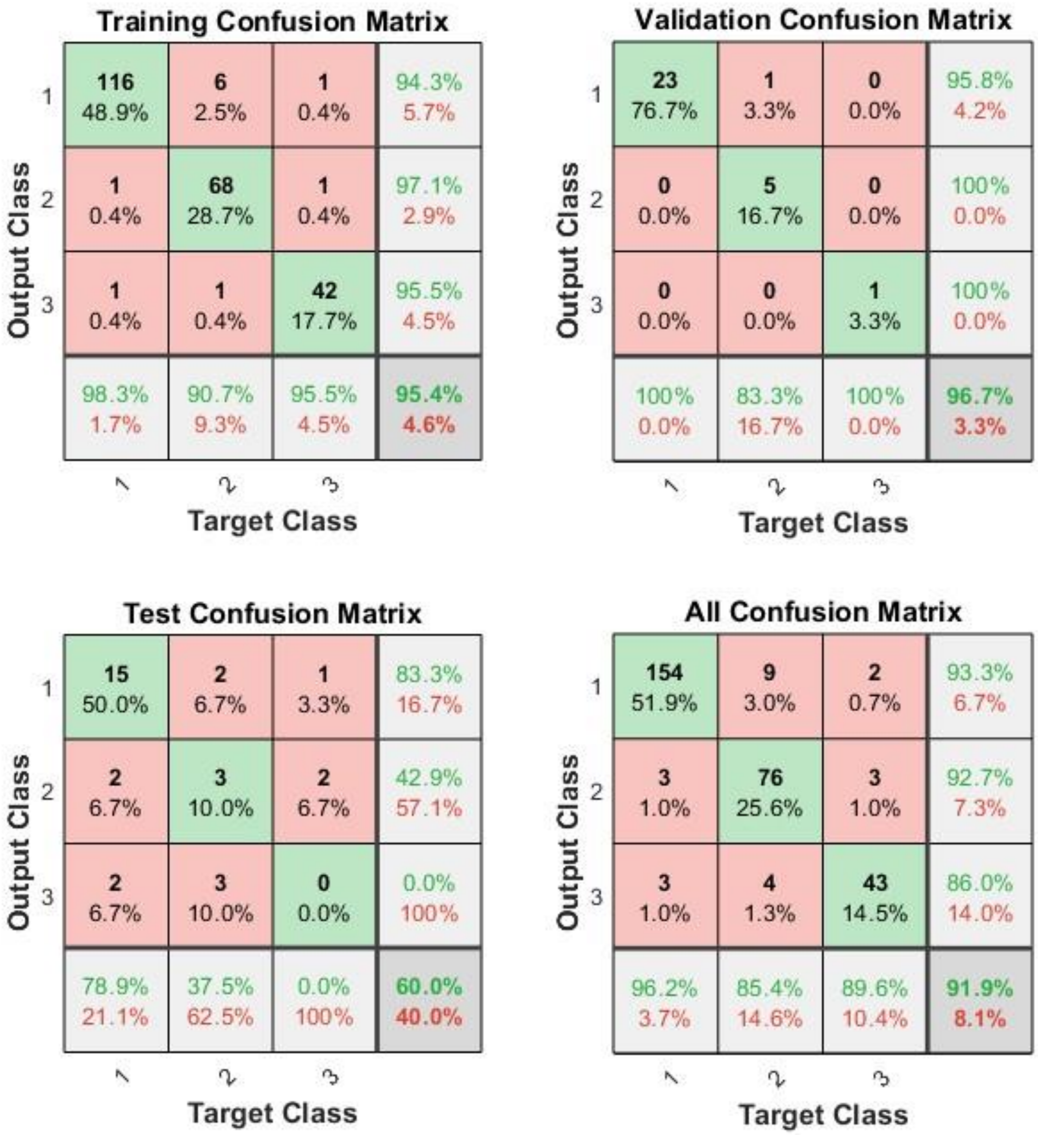
The results are as below:



Training Batch	Training Function	Accuracy
1	trainlm'	81.8%
2		87.9%
3		85.5%
4		70.7%
5		91.9%
Training Batch	Training Function	Accuracy
1	traingd	66.3%
2		73.4%
3		74.1%
4		71.7%
5		67.7%
Training Batch	Training Function	Accuracy
1	traingdx	74.4%
2		72.4%
3		60.6%
4		70.7%
5		70.4%

Table 14: Results for testcase 5

The best model performance was obtained using the ‘trainlm’ training function again.



Training Confusion Matrix

Output Class \ Target Class	1	2	3	
1	116 48.9%	6 2.5%	1 0.4%	94.3% 5.7%
2	1 0.4%	68 28.7%	1 0.4%	97.1% 2.9%
3	1 0.4%	1 0.4%	42 17.7%	95.5% 4.5%
	98.3% 1.7%	90.7% 9.3%	95.5% 4.5%	95.4% 4.6%

Validation Confusion Matrix

Output Class \ Target Class	1	2	3	
1	23 76.7%	1 3.3%	0 0.0%	95.8% 4.2%
2	0 0.0%	5 16.7%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	1 3.3%	100% 0.0%
	100% 0.0%	83.3% 16.7%	100% 0.0%	96.7% 3.3%

Test Confusion Matrix

Output Class \ Target Class	1	2	3	
1	15 50.0%	2 6.7%	1 3.3%	83.3% 16.7%
2	2 6.7%	3 10.0%	2 6.7%	42.9% 57.1%
3	2 6.7%	3 10.0%	0 0.0%	0.0% 100%
	78.9% 21.1%	37.5% 62.5%	0.0% 100%	60.0% 40.0%

All Confusion Matrix

Output Class \ Target Class	1	2	3	
1	154 51.9%	9 3.0%	2 0.7%	93.3% 6.7%
2	3 1.0%	76 25.6%	3 1.0%	92.7% 7.3%
3	3 1.0%	4 1.3%	43 14.5%	86.0% 14.0%
	96.2% 3.7%	85.4% 14.6%	89.6% 10.4%	91.9% 8.1%

Figure 24: Confusion matrix testcase 5

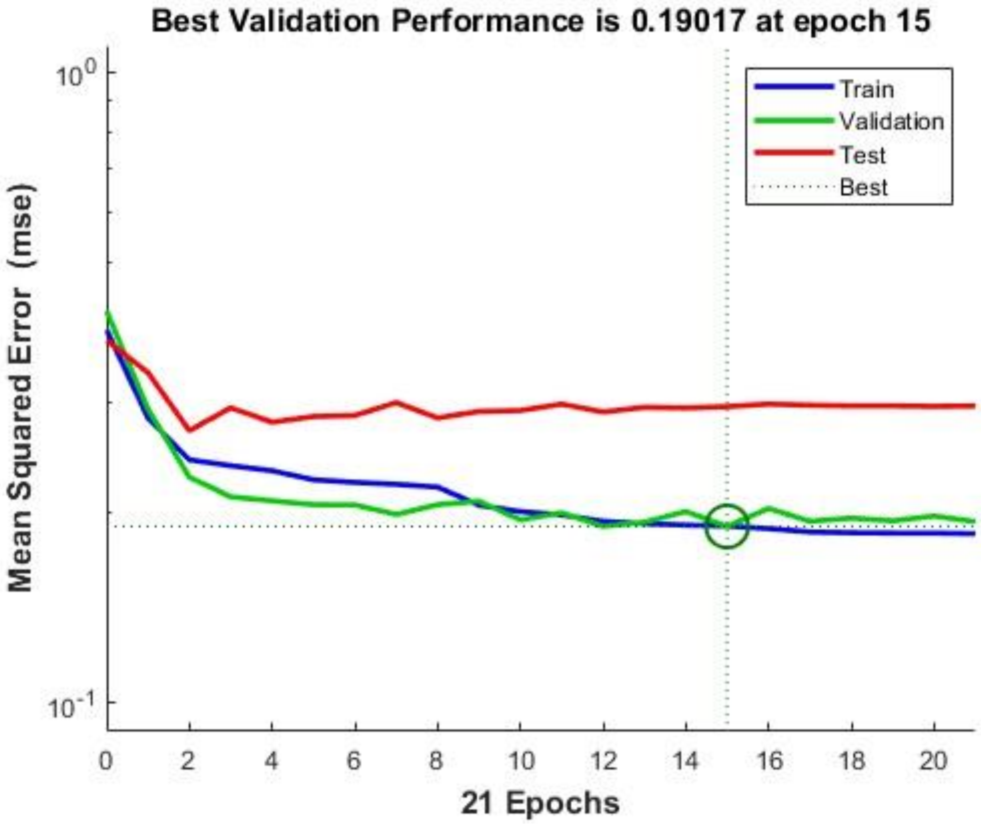


Figure 25: Performance curve testcase 5

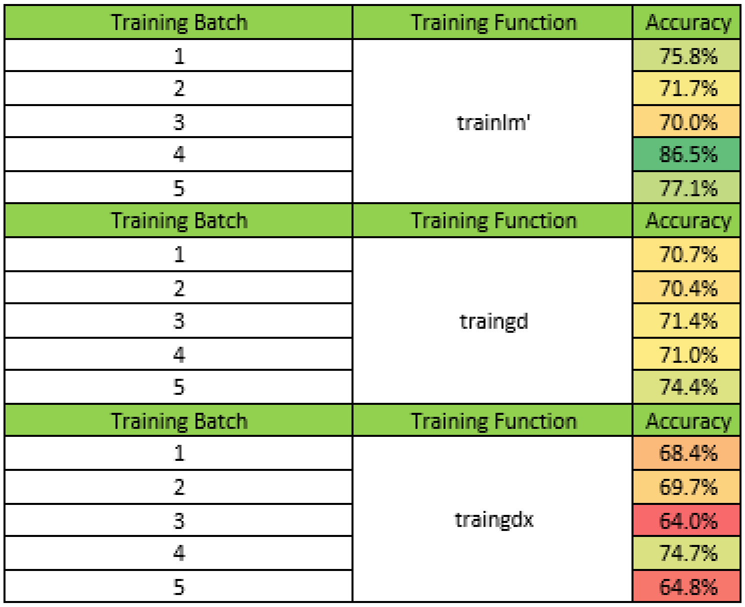
Test Case 6:

In this case, we change the output layer activation function to something else and check the model performance. The hyperparameters used are as below:

Input No	13
Hidden Layer No	1
Hidden Layer Activation Function	Sigmoid
No of Neurons	20
No of Outputs	3
Output Activation Function	tansig
Normalized data	Yes

Table 15: Hyperparameters for testcase 6

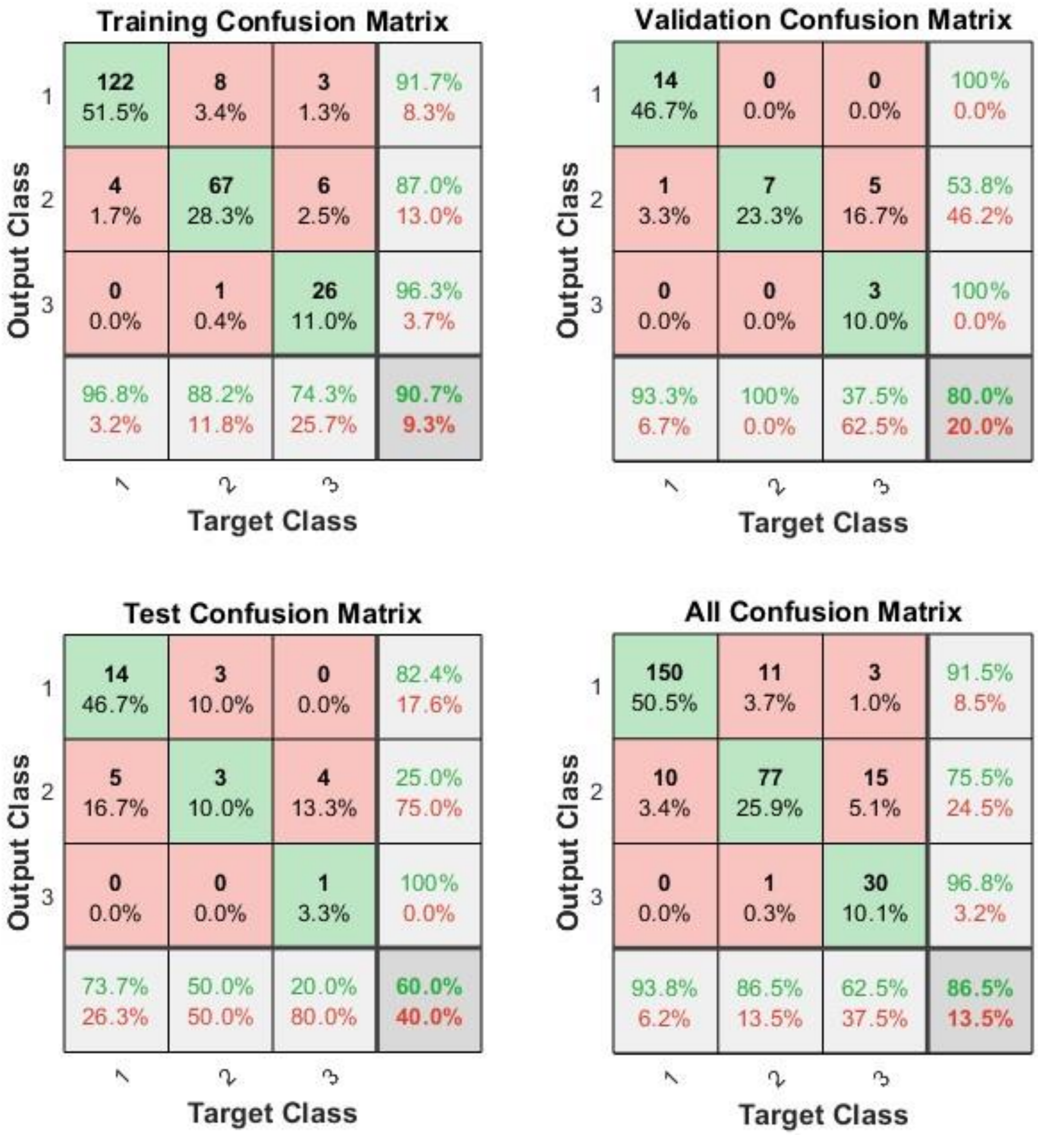
The overall results of the model performance are as below:



Training Batch	Training Function	Accuracy
1	trainlm'	75.8%
2		71.7%
3		70.0%
4		86.5%
5		77.1%
Training Batch	Training Function	Accuracy
1	traingd	70.7%
2		70.4%
3		71.4%
4		71.0%
5		74.4%
Training Batch	Training Function	Accuracy
1	traingdx	68.4%
2		69.7%
3		64.0%
4		74.7%
5		64.8%

Table 16: Results testcase 6

Again, the best performance is the trainlm function.



Training Confusion Matrix

Output Class	1	2	3	
1	122 51.5%	8 3.4%	3 1.3%	91.7% 8.3%
2	4 1.7%	67 28.3%	6 2.5%	87.0% 13.0%
3	0 0.0%	1 0.4%	26 11.0%	96.3% 3.7%
	96.8% 3.2%	88.2% 11.8%	74.3% 25.7%	90.7% 9.3%

Target Class

Validation Confusion Matrix

Output Class	1	2	3	
1	14 46.7%	0 0.0%	0 0.0%	100% 0.0%
2	1 3.3%	7 23.3%	5 16.7%	53.8% 46.2%
3	0 0.0%	0 0.0%	3 10.0%	100% 0.0%
	93.3% 6.7%	100% 0.0%	37.5% 62.5%	80.0% 20.0%

Target Class

Test Confusion Matrix

Output Class	1	2	3	
1	14 46.7%	3 10.0%	0 0.0%	82.4% 17.6%
2	5 16.7%	3 10.0%	4 13.3%	25.0% 75.0%
3	0 0.0%	0 0.0%	1 3.3%	100% 0.0%
	73.7% 26.3%	50.0% 50.0%	20.0% 80.0%	60.0% 40.0%

Target Class

All Confusion Matrix

Output Class	1	2	3	
1	150 50.5%	11 3.7%	3 1.0%	91.5% 8.5%
2	10 3.4%	77 25.9%	15 5.1%	75.5% 24.5%
3	0 0.0%	1 0.3%	30 10.1%	96.8% 3.2%
	93.8% 6.2%	86.5% 13.5%	62.5% 37.5%	86.5% 13.5%

Target Class

Figure 26: Confusion matrix testcase 6

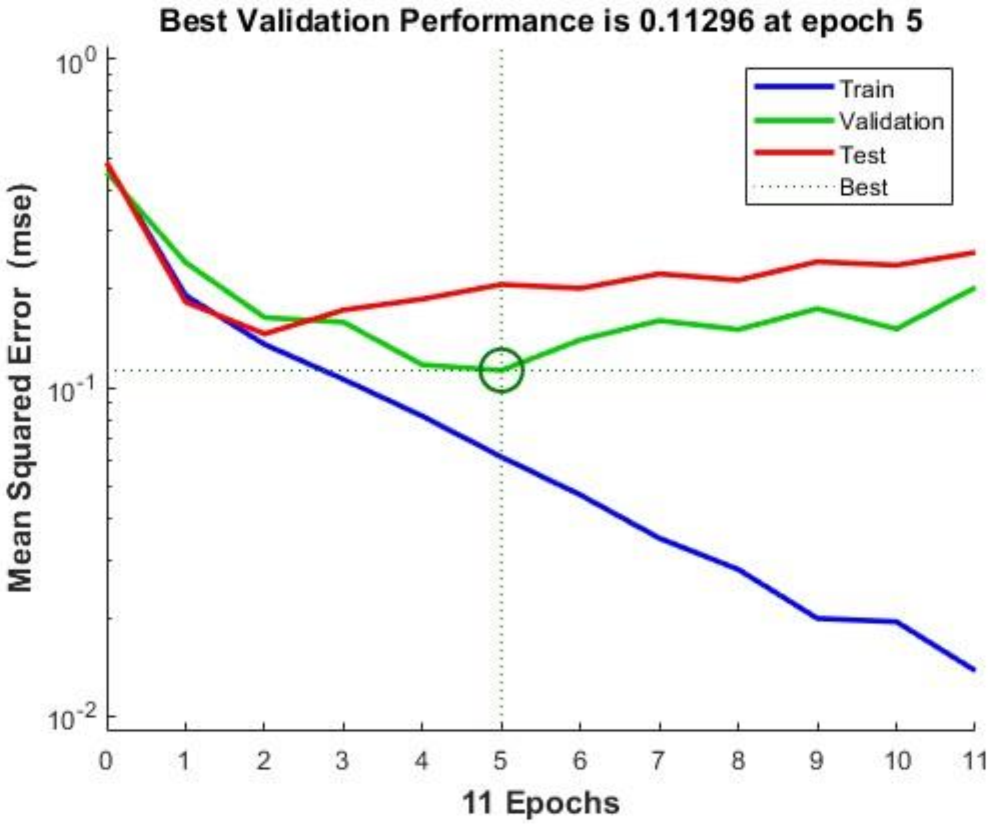
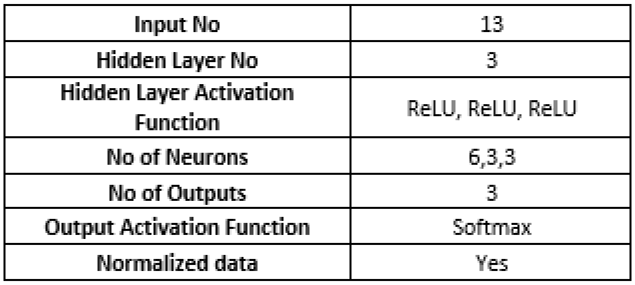


Figure 27: Performance curve testcase 6

Test Case 7:

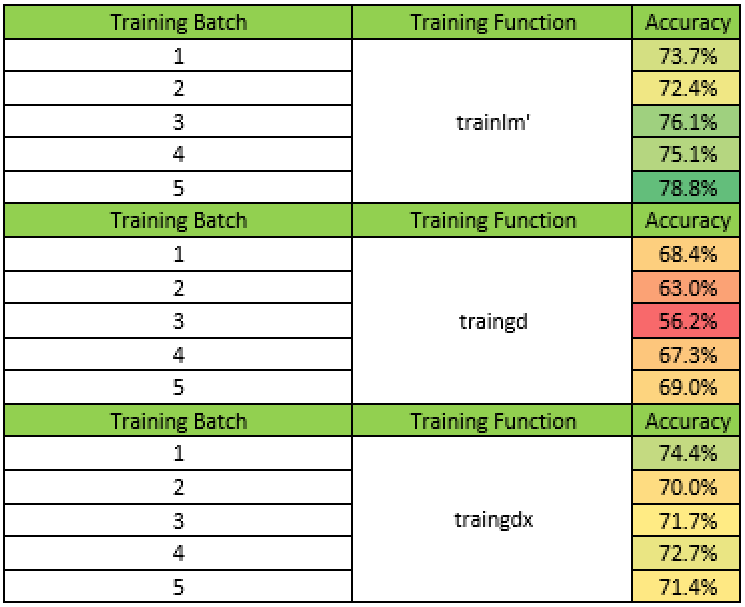
Here, we change the network architecture and increase the number of hidden layers. We used the ReLU activation function for the hidden layers and vary the activation function of output layer. The hyperparameters for this case are as below:



Input No	13
Hidden Layer No	3
Hidden Layer Activation Function	ReLU, ReLU, ReLU
No of Neurons	6,3,3
No of Outputs	3
Output Activation Function	Softmax
Normalized data	Yes

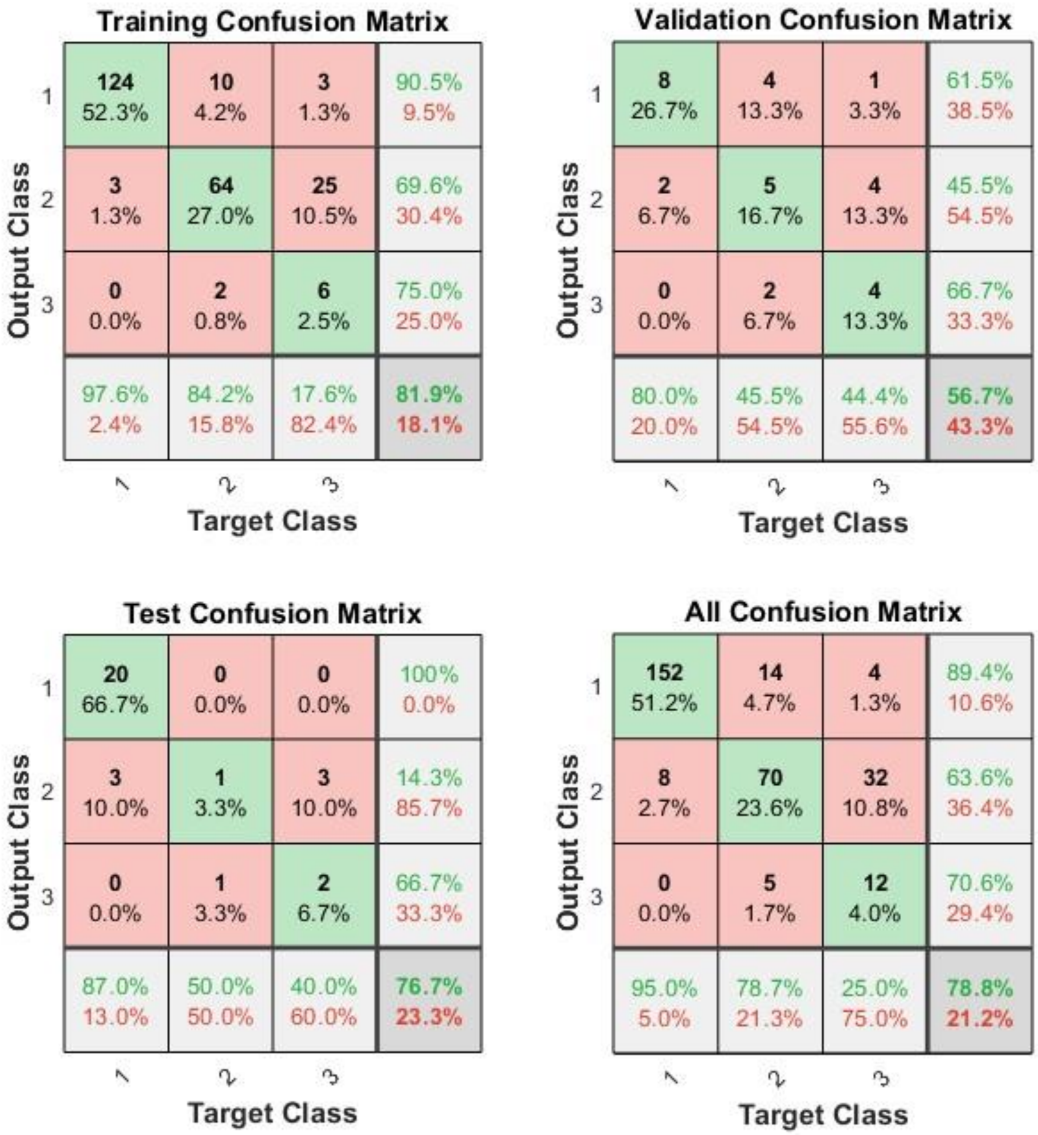
Table 17: Hyperparameters for testcase 7

The results are as below:



Training Batch	Training Function	Accuracy
1	trainlm'	73.7%
2		72.4%
3		76.1%
4		75.1%
5		78.8%
Training Batch	Training Function	Accuracy
1	traingd	68.4%
2		63.0%
3		56.2%
4		67.3%
5		69.0%
Training Batch	Training Function	Accuracy
1	traingdx	74.4%
2		70.0%
3		71.7%
4		72.7%
5		71.4%

Table 18: Results testcase 7



Training Confusion Matrix

Output Class \ Target Class	1	2	3	
1	124 52.3%	10 4.2%	3 1.3%	90.5% 9.5%
2	3 1.3%	64 27.0%	25 10.5%	69.6% 30.4%
3	0 0.0%	2 0.8%	6 2.5%	75.0% 25.0%
	97.6% 2.4%	84.2% 15.8%	17.6% 82.4%	81.9% 18.1%

Validation Confusion Matrix

Output Class \ Target Class	1	2	3	
1	8 26.7%	4 13.3%	1 3.3%	61.5% 38.5%
2	2 6.7%	5 16.7%	4 13.3%	45.5% 54.5%
3	0 0.0%	2 6.7%	4 13.3%	66.7% 33.3%
	80.0% 20.0%	45.5% 54.5%	44.4% 55.6%	56.7% 43.3%

Test Confusion Matrix

Output Class \ Target Class	1	2	3	
1	20 66.7%	0 0.0%	0 0.0%	100% 0.0%
2	3 10.0%	1 3.3%	3 10.0%	14.3% 85.7%
3	0 0.0%	1 3.3%	2 6.7%	66.7% 33.3%
	87.0% 13.0%	50.0% 50.0%	40.0% 60.0%	76.7% 23.3%

All Confusion Matrix

Output Class \ Target Class	1	2	3	
1	152 51.2%	14 4.7%	4 1.3%	89.4% 10.6%
2	8 2.7%	70 23.6%	32 10.8%	63.6% 36.4%
3	0 0.0%	5 1.7%	12 4.0%	70.6% 29.4%
	95.0% 5.0%	78.7% 21.3%	25.0% 75.0%	78.8% 21.2%

Figure 28: Confusion matrix testcase 7

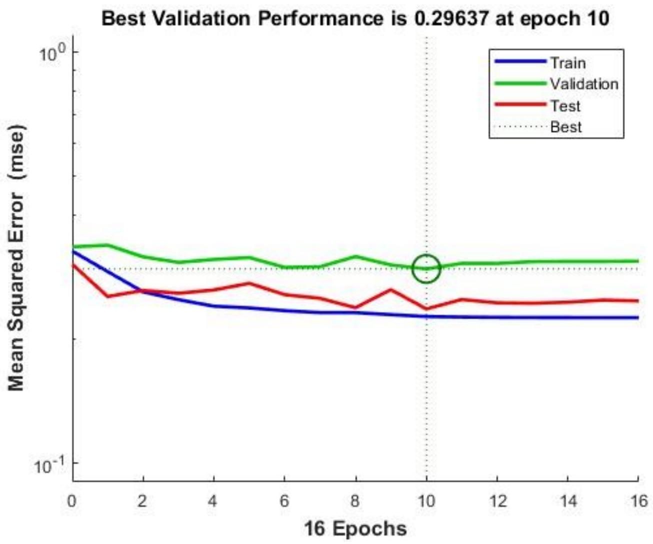
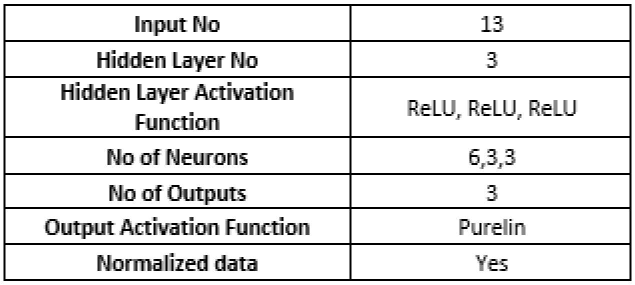


Figure 29: Performance curve testcase 7

Test Case 8:

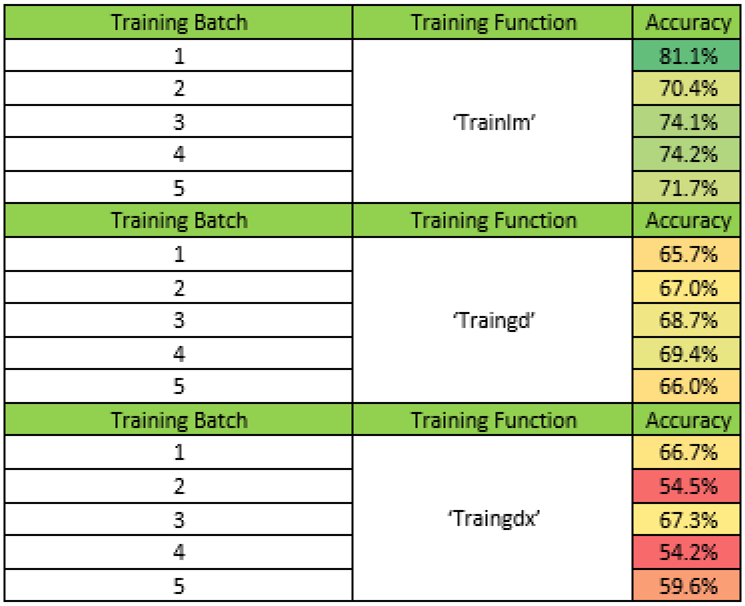
In this case, I change the output layer activation function to 'purelin' and check the output. The hyperparameter settings are as below:



Input No	13
Hidden Layer No	3
Hidden Layer Activation Function	ReLU, ReLU, ReLU
No of Neurons	6,3,3
No of Outputs	3
Output Activation Function	Purelin
Normalized data	Yes

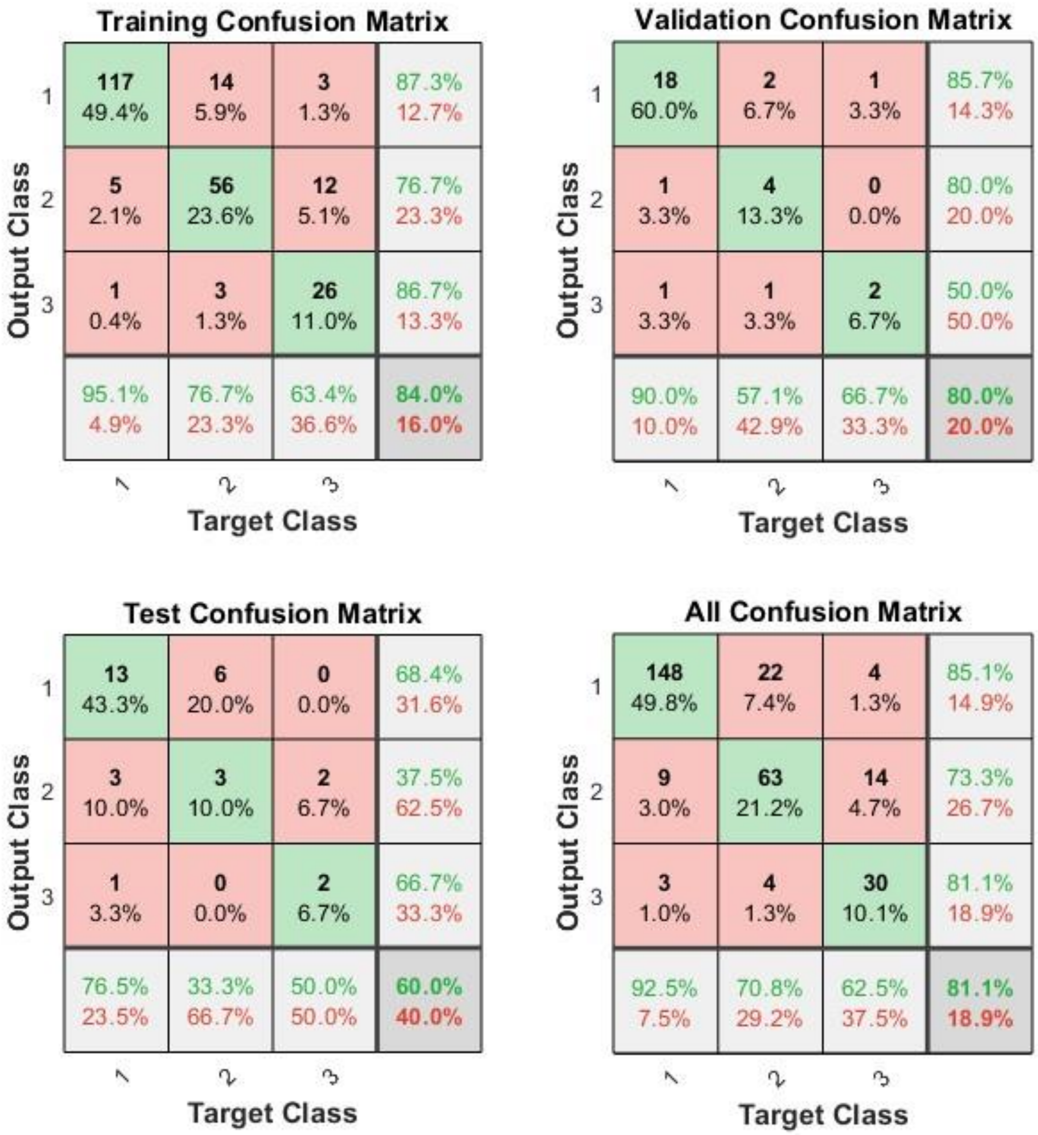
Table 19: Hyperparameters for testcase 8

The results are as below:



Training Batch	Training Function	Accuracy
1	‘Trainlm’	81.1%
2		70.4%
3		74.1%
4		74.2%
5		71.7%
Training Batch	Training Function	Accuracy
1	‘Traingd’	65.7%
2		67.0%
3		68.7%
4		69.4%
5		66.0%
Training Batch	Training Function	Accuracy
1	‘Traingdx’	66.7%
2		54.5%
3		67.3%
4		54.2%
5		59.6%

Table 20: Results testcase 8



Training Confusion Matrix

	1	2	3	
1	117 49.4%	14 5.9%	3 1.3%	87.3% 12.7%
2	5 2.1%	56 23.6%	12 5.1%	76.7% 23.3%
3	1 0.4%	3 1.3%	26 11.0%	86.7% 13.3%
	95.1% 4.9%	76.7% 23.3%	63.4% 36.6%	84.0% 16.0%

Validation Confusion Matrix

	1	2	3	
1	18 60.0%	2 6.7%	1 3.3%	85.7% 14.3%
2	1 3.3%	4 13.3%	0 0.0%	80.0% 20.0%
3	1 3.3%	1 3.3%	2 6.7%	50.0% 50.0%
	90.0% 10.0%	57.1% 42.9%	66.7% 33.3%	80.0% 20.0%

Test Confusion Matrix

	1	2	3	
1	13 43.3%	6 20.0%	0 0.0%	68.4% 31.6%
2	3 10.0%	3 10.0%	2 6.7%	37.5% 62.5%
3	1 3.3%	0 0.0%	2 6.7%	66.7% 33.3%
	76.5% 23.5%	33.3% 66.7%	50.0% 50.0%	60.0% 40.0%

All Confusion Matrix

	1	2	3	
1	148 49.8%	22 7.4%	4 1.3%	85.1% 14.9%
2	9 3.0%	63 21.2%	14 4.7%	73.3% 26.7%
3	3 1.0%	4 1.3%	30 10.1%	81.1% 18.9%
	92.5% 7.5%	70.8% 29.2%	62.5% 37.5%	81.1% 18.9%

Figure 30: Confusion matrix testcase 8

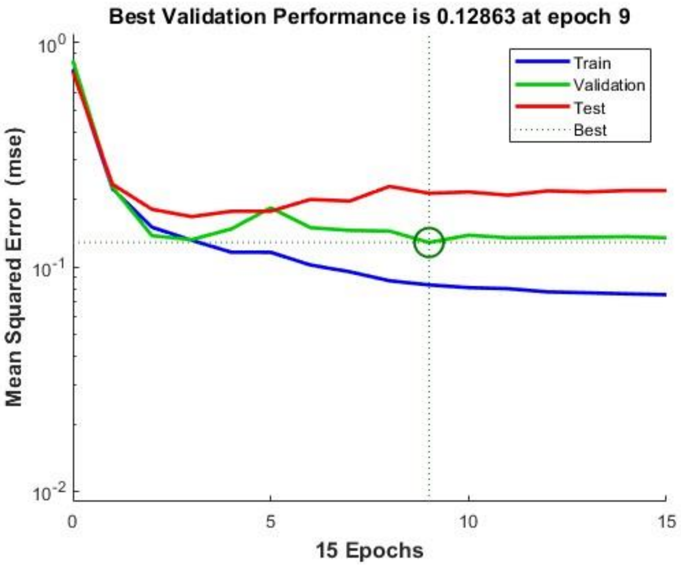


Figure 31: Performance curve testcase 8

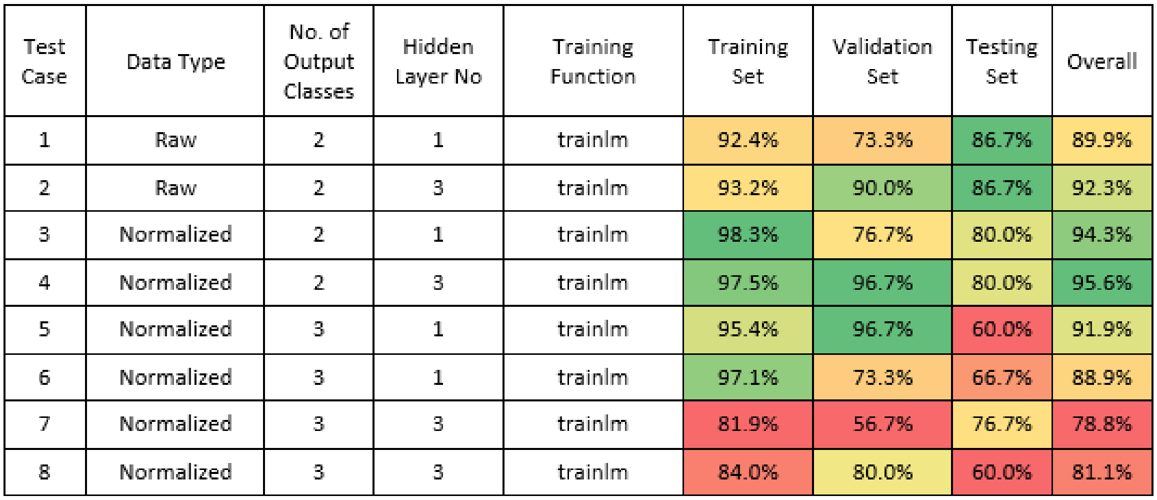
Discussions and Conclusions.

In this report, we have looked at the design of a neural network to address the problem of classifying the kind of heart disease a patient has. We used the Cleveland heart disease dataset which has 297 instances and 14 attributes. We have also explored different architecture types to app

My results show that the best performance is obtained using a neural network of 3 hidden layers with 13 input features, 1 output layer and two output targets of 'normal' and 'diseased'. For the case with 3 output targets, we saw from the exploratory data analysis that the data was not balanced, and our results show the effects of this. We can see from the table below that in testcase 5 – 7, the accuracy of the testing and validation set shows bias, misclassification, and poor generalization because of this imbalance.

My results also show that the best performance is gotten using the 'trainlm' function which uses Levenberg-Marquardt optimization method as against the 'traingd' and 'traingdx' function.

We can conclude that with the right architecture and better data, neural nets can be designed to predict the occurrence of heart diseases in people, provide insights for patients and assist with improving the healthcare for several patient who may be at risk..



Test Case	Data Type	No. of Output Classes	Hidden Layer No	Training Function	Training Set	Validation Set	Testing Set	Overall
1	Raw	2	1	trainlm	92.4%	73.3%	86.7%	89.9%
2	Raw	2	3	trainlm	93.2%	90.0%	86.7%	92.3%
3	Normalized	2	1	trainlm	98.3%	76.7%	80.0%	94.3%
4	Normalized	2	3	trainlm	97.5%	96.7%	80.0%	95.6%
5	Normalized	3	1	trainlm	95.4%	96.7%	60.0%	91.9%
6	Normalized	3	1	trainlm	97.1%	73.3%	66.7%	88.9%
7	Normalized	3	3	trainlm	81.9%	56.7%	76.7%	78.8%
8	Normalized	3	3	trainlm	84.0%	80.0%	60.0%	81.1%

Figure 32: Summary of results

References

1) Rajeswari, D. and Thangavel, K., 2020. The performance of data normalization techniques on heart disease datasets. *International Journal of Advanced Research in Engineering and Technology*, *11*(12), pp.2350-2357.
2) Ajam, N., 2015. Heart diseases diagnoses using artificial neural network. *IISTE Network and Complex Systems*, *5*(4).
3) Antar, R.K., A Lotaibi, S.T. and Al Ghamdi, M., 2021. Heart Attack Prediction using Neural Network and Different Online Learning Methods. ("Heart Attack Prediction using Neural Network and Different Online ...") *International Journal of Computer Science & Network Security*, *21*(6), pp.77-88.
4) Abdolrasol, M.G., Hussain, S.S., Ustun, T.S., Sarker, M.R., Hannan, M.A., Mohamed, R., Ali, J.A., Mekhilef, S. and Milad, A., 2021. Artificial neural networks-based optimization techniques: A review. *Electronics*, *10*(21), p.2689.
5) Olczak, J., Pavlopoulos, J., Prijs, J., Ijpma, F.F., Doornberg, J.N., Lundström, C., Hedlund, J. and Gordon, M., 2021. Presenting artificial intelligence, deep learning, and machine learning studies to clinicians and healthcare stakeholders: an introductory reference with a guideline and a Clinical AI Research (CAIR) checklist proposal. *Acta orthopaedica*, *92*(5), pp.513-525.
6) Blum, A.L. and Langley, P., 1997. Selection of relevant features and examples in machine learning. *Artificial intelligence*, *97*(1-2), pp.245-271.
7) Stevens, L.M., Mortazavi, B.J., Deo, R.C., Curtis, L. and Kao, D.P., 2020. Recommendations for reporting machine learning analyses in clinical research. *Circulation: Cardiovascular Quality and Outcomes*, *13*(10), p.e006556.
8) British Heart Foundation (2021). How Your Heart Works. [online] Bhf.org.uk. Available at: https://www.bhf.org.uk/informationsupport/how-a-healthy-heart-works.
9) World Health Organization (2018). World health statistics 2018 : monitoring health for the SDGs, sustainable development goals. Geneve: World Health Organization, Cop.
10) Narkhede, S. (2018). Understanding Confusion Matrix. [online] Medium. Available at: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.
11) Gupta, A. (2020). Feature Selection Techniques in Machine Learning. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/.

Appendix

Section 1 Code

```
clear all

%%
load cleveland_heart_disease_dataset_labelled.mat
%%
T = table(x,t);
B = table2array(T);
Table1 = array2table(B);
Table2 = renamevars(Table1,
{'B1','B2','B3','B4','B5','B6','B7','B8','B9','B10','B11','B12','B13','B14'},
...
    {'Age','Sex','CP','Trestbps','Chol','fbs','restecg', ...
    'thalach','exang','oldpeak','slope','ca','thal','target'});
%% change class 2 to 1
Table3 = Table2;
Table3.target(Table3.target == 2) = 1;
%% convert table to array

data = table2array(Table3);
data = normalize(data,'range');
sorted_d = sortrows(data,14);
%%
patterns = sorted_d(:,1:13)';
targets = sorted_d(:,14)';
%%
net = feedforwardnet([20]);
net.trainFcn = 'trainlm';
[trainInd,valInd,testInd] = dividerand(297,0.8,0.1,0.1);

net.layers{1}.transferFcn = 'logsig';
%net.layers{2}.transferFcn = 'logsig';
%net.layers{3}.transferFcn = 'logsig';
net.layers{end}.transferFcn = 'purelin';

net.performFcn = 'mse';
net.plotFcns = {'plotperform','plottrainstate','ploterrhist',
'plotregression',
    'plotconfusion', 'plotroc'};

net.trainParam.epochs = 1000;

[net,tr] = train(net, patterns, targets);
```

```
%% output
predict = sim(net,patterns);

%% plot confusion
figure;
plotconfusion(targets, predict)
```

Section 2 Code

```
%% CourseWork Section 2: 3 outputs.

clear all

%%
 load cleveland_heart_disease_dataset_labelled.mat

%% Load data
patterns = x;
targets = t';

%% Normalize data set

%pt = normalize(patterns,'range');
pt = (patterns - min(patterns)) ./ (max(patterns) - min(patterns));
pt_norm = pt';

%% create new table - combine two inputs and output together
newtag = [pt,t];
%% separate data based on output in column 14.
sorted_d = sortrows(newtag,14);


%% select data
zeros_array = sorted_d(1:160,1:13);
ones_array = sorted_d(161:249,1:13);
twos_array = sorted_d(250:end,1:13);
target = sorted_d(:,14);
%% create complete data but sorted

data = cat(1,zeros_array,ones_array,twos_array);
data = data';
data_targets = target;



%% Replace data in column 14 with the correct labels then change the labels
```

```
% to something that can be used for hot encoding

% Convert targets to categorical array
data_target = categorical(data_targets, [0 1 2], {'normal' 'mild heart
disease' 'severe heart disease'});
%test_set_target = categorical(test_set_target, [0 1 2], {'normal' 'mild
heart disease' 'severe heart disease'});
d_target = data_target;
d_tar = onehotencode(d_target,2);
tar = d_tar';

%% design network using the train data.
net = feedforwardnet([20]);

% Set training algorithm
net.trainFcn = 'trainlm';

%
%net.divideFcn = 'dividetrain';

% Set activation function for hidden layers
% Set activation function for hidden layers
net.layers{1}.transferFcn = 'logsig';
% net.layers{2}.transferFcn = 'tansig';
% net.layers{3}.transferFcn = 'logsig';
net.layers{end}.transferFcn = 'tansig';

%set training fuction and ratios
[trainInd,valInd,testInd] = dividerand(297,0.8,0.1,0.1);

% net.divideFcn = 'dividerand';
% net.divideParam.trainRatio = 0.8;
% net.divideParam.valRatio = 0.1;
% net.divideParam.testRatio = 0.1;

% Choose an evaluation metrics (mae, mse)
net.performFcn = 'mse';
net.plotFcns = {'plotperform','plottrainstate','ploterrhist',
'plotregression', 'plotroc', 'plotconfusion'};

% Set number of epochs for training
net.trainParam.epochs = 1000;
%% Train network
[net,tr] = train(net, data, tar);

%% Test network next part

predict = sim(net,data);
%% Plot confusion
figure;
```

```
plotconfusion(tar, predict)
```

Exploratory data analysis code:

```
%% Exploratory data analysis code

clear all

%%

load cleveland_heart_disease_dataset_labelled.mat

%% create table from the data
T = table(x,t);
B = table2array(T);
Table1 = array2table(B);
Table2 = renamevars(Table1,
{'B1','B2','B3','B4','B5','B6','B7','B8','B9','B10','B11','B12','B13','B14'},
...
    {'Age','Sex','CP','Trestbps','Chol','fbs','restecg', ...
    'thalach','exang','oldpeak','slope','ca','thal','target'});


%% plots
%% distribution of Sec
c = categorical(Table2.Sex,[0 1],{'Female','Male'});
histogram(c)
%histogram(c,{'Female','Male'},"FaceColor",{'r','g'})
%%
%distribution of age
histogram(Table2.Age)
%%
%distribution of sex
C = categorical(Table2.target,[0 1 2],{'No Heart Disease','Mild Heart
Disease','Severe Heart Disease'});
histogram(C)
%% correlation coefficient of the data
R = corrplot(Table2,Type="Pearson",TestR="on");


%% Scatterplots
scatter(Table2,"target",{'CP','exang','ca','thal'});
%%
%%Normalize data
norm_T = normalize(Table2,'range');

%%
figure
corrplot(Table2(:,1:end),'testR','on')
title('Correlation plot')
```

```
%%
%bivariate histogram
c = [Table2.target; Table2.Sex];
bar(c)
colorbar

%%
% Create a histogram of age
figure
histogram(Table2.Age)
title('Age Distribution')
% xlabel('Age')
% ylabel('Frequency')
%%
% Create a scatter plot of age vs. cholesterol
figure
scatter(Table2.target, Table2.Age)
title('Age vs. Cholesterol')
xlabel('Age')
ylabel('Cholesterol')
%%
% Create a bar plot of the frequency of each chest pain type
figure
cp_counts = countcats(categorical(Table2.CP));
bar(cp_counts)
title('Chest Pain Type Frequencies')
xlabel('Chest Pain Type')
ylabel('Frequency')
xticklabels({'Typical Angina','Atypical Angina','Non-Anginal
Pain','Asymptomatic'})
%%
% Create a bar plot of the frequency of each resting ECG type
figure
cp_counts = countcats(categorical(Table2.restecg));
bar(cp_counts)
title('Rest ECG Frequencies')
xlabel('Rest ECG')
ylabel('Frequency')
xticklabels({'Normal','Having ST-T wave abnormality','Left ventricular
hypertrophy'})
xtickangle(45)
%%
c = [Table2.restecg Table2.target];
histogram(c)
colorbar
%%
figure
histogram(Table2.CP,"Data",Table2.target)
```

```
%%
figure
histogram(Table2.target,"Data",Table2.restecg)
xticks([0,1,2])
xticklabels({'Normal','Having ST-T wave abnormality','Left ventricular
hypertrophy'})
% xlabel('Rest ECG')
% ylabel('Frequency')
%%
figure
histogram(Table2.target,"Data",Table2.thalach,"","auto")


%%
% Histogram of thalac by target type
figure;
histogram(Table2.thalach(Table2.target==0), 'BinWidth', 5, 'FaceColor',
'blue');
hold on;
histogram(Table2.thalach(Table2.target==1),  'BinWidth', 5, 'FaceColor',
'red');
histogram(Table2.thalach(Table2.target==2), 'BinWidth', 5, 'FaceColor',
'green');
legend('Normal', 'Mild','Severe' );
title('Histogram of Thalac by Target Type');
%%
figure;
histogram(Table2.Age(Table2.target==0), 'BinWidth', 5, 'FaceColor', 'blue');
hold on;
histogram(Table2.Age(Table2.target==1), 'BinWidth', 5, 'FaceColor', 'red');
histogram(Table2.Age(Table2.target==2),  'BinWidth', 5, 'FaceColor',
'green');
legend('Normal', 'Mild','Severe' );
title('Histogram of Age by Target Type');

%%
figure;
histogram(Table2.Trestbps(Table2.target==0), 'BinWidth', 5, 'FaceColor',
'blue');
hold on;
histogram(Table2.Trestbps(Table2.target==1), 'BinWidth', 5, 'FaceColor',
'red');
histogram(Table2.Trestbps(Table2.target==2),  'BinWidth', 5, 'FaceColor',
'green');
legend('Normal', 'Mild','Severe' );
title('Histogram of Trestbps by Target Type');

%%
% figure;
histogram(Table2.Chol(Table2.target==0), 'BinWidth', 5, 'FaceColor', 'blue');
```

```
hold on;
histogram(Table2.Chol(Table2.target==1), 'BinWidth', 5, 'FaceColor', 'red');
histogram(Table2.Chol(Table2.target==2),  'BinWidth', 5, 'FaceColor',
'green');
legend('Normal', 'Mild','Severe' );
title('Histogram of Cholesterol by Target Type');

%% generate data subplots

subplot(2,2,1);
histogram(Table2.thalach(Table2.target==0), 'BinWidth', 5, 'FaceColor',
"#77AC30");
hold on;
histogram(Table2.thalach(Table2.target==1),  'BinWidth', 5, 'FaceColor',
"#A2142F");
histogram(Table2.thalach(Table2.target==2), 'BinWidth', 5, 'FaceColor',
"#D95319");
legend('Normal', 'Mild','Severe' );
title('Histogram of Thalac by Target Type');


subplot(2,2,2);
histogram(Table2.Age(Table2.target==0), 'BinWidth', 5, 'FaceColor',
"#77AC30");
hold on;
histogram(Table2.Age(Table2.target==1), 'BinWidth', 5, 'FaceColor',
"#A2142F");
histogram(Table2.Age(Table2.target==2),  'BinWidth', 5, 'FaceColor',
"#D95319");
legend('Normal', 'Mild','Severe' );
title('Histogram of Age by Target Type');

subplot(2,2,3);
histogram(Table2.Trestbps(Table2.target==0), 'BinWidth', 5, 'FaceColor',
"#77AC30");
hold on;
histogram(Table2.Trestbps(Table2.target==1), 'BinWidth', 5, 'FaceColor',
"#A2142F");
histogram(Table2.Trestbps(Table2.target==2),  'BinWidth', 5, 'FaceColor',
"#D95319");
legend('Normal', 'Mild','Severe' );
title('Histogram of Trestbps by Target Type');

subplot(2,2,4);
% figure;
histogram(Table2.Chol(Table2.target==0), 'BinWidth', 5, 'FaceColor',
"#77AC30");
hold on;
histogram(Table2.Chol(Table2.target==1), 'BinWidth', 5, 'FaceColor',
"#A2142F");
```

```
histogram(Table2.Chol(Table2.target==2),  'BinWidth', 5, 'FaceColor',
"#D95319");
legend('Normal', 'Mild','Severe' );
title('Histogram of Cholesterol by Target Type');


%% combine data and draw new plots.

%create a new table
Table3 = Table2;

% change variable 2 in table3.target to 1 making the class type now "with
% disease' and 'without disease'
Table3.target(Table3.target == 2) = 1;
%%
subplot(2,2,1);
histogram(Table3.thalach(Table3.target==0), 'BinWidth', 5, 'FaceColor',
"#77AC30");
hold on;
histogram(Table3.thalach(Table3.target==1),  'BinWidth', 5, 'FaceColor',
"#A2142F");
%histogram(Table2.thalach(Table3.target==2), 'BinWidth', 5, 'FaceColor',
"#D95319");
legend('Normal Heart', 'Diseased Heart' );
title('Histogram of Thalac by Target Type');


subplot(2,2,2);
histogram(Table3.Age(Table3.target==0), 'BinWidth', 5, 'FaceColor',
"#77AC30");
hold on;
histogram(Table3.Age(Table3.target==1), 'BinWidth', 5, 'FaceColor',
"#A2142F");
%histogram(Table2.Age(Table2.target==2),  'BinWidth', 5, 'FaceColor',
"#D95319");
legend('Normal Heart', 'Diseased Heart' );
title('Histogram of Age by Target Type');

subplot(2,2,3);
histogram(Table3.Trestbps(Table3.target==0), 'BinWidth', 5, 'FaceColor',
"#77AC30");
hold on;
histogram(Table3.Trestbps(Table3.target==1), 'BinWidth', 5, 'FaceColor',
"#A2142F");
%histogram(Table2.Trestbps(Table2.target==2),  'BinWidth', 5, 'FaceColor',
"#D95319");
legend('Normal Heart', 'Diseased Heart' );
title('Histogram of Trestbps by Target Type');

subplot(2,2,4);
```

```
% figure;
histogram(Table3.Chol(Table3.target==0), 'BinWidth', 5, 'FaceColor',
"#77AC30");
hold on;
histogram(Table3.Chol(Table3.target==1), 'BinWidth', 5, 'FaceColor',
"#A2142F");
%histogram(Table2.Chol(Table2.target==2),  'BinWidth', 5, 'FaceColor',
"#D95319");
legend('Normal Heart', 'Diseased Heart' );
title('Histogram of Cholesterol by Target Type');


%% scatterplot
figure
gscatter(Table2.Age, Table2.Trestbps, Table2.target)

%% scatterplot
figure
gscatter(Table3.Age, Table3.Trestbps, Table3.target,group,'br','xo')

%%
%% scatterplot
figure
gscatter(Table2.Age, Table2.thalach,
Table2.target,'','sd*','','on','Age','Cholesterol')
title('Scatterplot of Age against thalac grouped by targets')
grid on
grid minor
legend

%% scatterplot
figure
gscatter(Table2.Age, Table2.oldpeak,
Table2.target,'','sd*','','on','Age','Oldpeak')
title('Scatterplot of Age against oldpeak grouped by targets')
grid on
grid minor
legend

%%
subplot(2,2,1)
gscatter(Table2.Age, Table2.thalach,
Table2.target,'','sd*','','on','Age','thalac')
title('Scatterplot of Age against thalac grouped by targets')
grid on
grid minor
legend

subplot(2,2,2)
```

```
gscatter(Table2.Age, Table2.oldpeak,
Table2.target,'','sd*','','on','Age','Oldpeak')
title('Scatterplot of Age against oldpeak grouped by targets')
grid on
grid minor
legend

subplot(2,2,3)
gscatter(Table2.Age, Table2.Trestbps,
Table2.target,'','sd*','','on','Age','Trestbps')
title('Scatterplot of Age against Trestbps grouped by targets')
grid on
grid minor
legend

subplot(2,2,4)
gscatter(Table2.Age, Table2.Chol,
Table2.target,'','sd*','','on','Age','Chol')
title('Scatterplot of Age against Cholesterol grouped by targets')
grid on
grid minor
legend
```