Application of Neural Network on Determining Color of Traffic Lights

LBYCP29 – Project Proposal

Gervin Guevarra, Allen Koizumi, Nicholas Moreno, Charleston Uy
Department of Electronics and Communications Engineering
Gokongwei College of Engineering, De La Salle University
Manila, Philippines

***Abstract*—This project proposes the application of neural network on determining the color of traffic lights. Several pictures of stop lights would be presented to a learning algorithm after which a neural network would be created that could identify at least 2 colors of traffic lights – red, and green**

Keywords—Neural network, traffic light, color, learning algorithm

I. INTRODUCTION

In the totality of the lab, we have been experimenting several topics that would lead us into developing neural networks, albeit very robust ones. Thus it is only fitting that the project for this course would be applying neural networks on real world data. For this application, we have chosen the color of traffic lights, and would have the neural network determine if it is red, yellow, or green. This application in a deeper sense may pave a way for semi-autonomous behavior on vehicles since it would then be able to react properly to a given traffic light.

II. OBJECTIVES

The experiment aims to achieve the following objectives

- To obtain a 90% accuracy on the data sets given
- To compile a data set that consists of traffic lights with different colors
- To implement a neural network that is able to determine the color of a traffic light

III. METHODOLOGY

In preliminary gathering of the training data, Google Images were used. Searching for traffic lights seen in the Philippines, a total approximate of pictures gathered were 122 consisting of Red and Green traffic lights. The Yellow traffic light was considerably hard to capture since only a few to none were able to come out of the search. The pictures retrieved from the internet was then cropped to the exact size of the traffic light with a rectangular cropping tool that had a ratio of 1:1. The now cropped photos were then down sampled to 35x35 pixels each to incorporate to the neural network.

For the testing data, actual photos of traffic lights were gathered, going around Metro Manila and capturing most traffic lights. An approximate of 264 pictures were captured consisting of Red, Yellow and Green traffic lights. These also were cropped to the size of the traffic light with a rectangular cropping tool with a ratio of 1:1. They were also down sampled using Adobe Photoshop® to 35x35 pixels each to be injected to the neural network.

Some hindrances arose when obtaining the traffic lights particularly the Yellow traffic light. It was considerably hard to obtain the Yellow traffic light on the internet as there are only a few captured while it was considerably difficult to capture the actual Yellow traffic light as it would only appear after a Green traffic light. Due to time constraints, it is decided not to include the yellow traffic lights in the neural network as there is insufficient data gathered. There were also obstacles in converting the collected images into ".idx3-ubyte" data format prior to the integration to the neural network. The *.idx3-ubyte data format is chosen as this is the same data type used by the MNIST database of handwritten digits.

A library called "MNISTEN" was found on the open source community. This library allows the automatic downsampling of multiple images into a 32x32 image and conversion of these images into a single idx3-ubyte data format – the same format being used by the MNIST database. However, this proved to be time consuming and the installation of the said library requires a lot of dependencies, even involving OpenCV and others.

The solution was to push through using the prepared collection of 35x35 images shown in Figures 1 and 2. Instead of going through all the tedious process of coding the neural network, the group decided to use a MATLAB® toolbox "Neural Network Toolbox". This toolbox provides a user-friendly GUI/wizard that allows the user to build a neural network in just a few clicks.

Figure 1. A part of the collection of Green traffic lights and Red traffic lights that were originally intended for use as training data. The image is not to scale. Actual size for traffic light is 35x35 pixels.

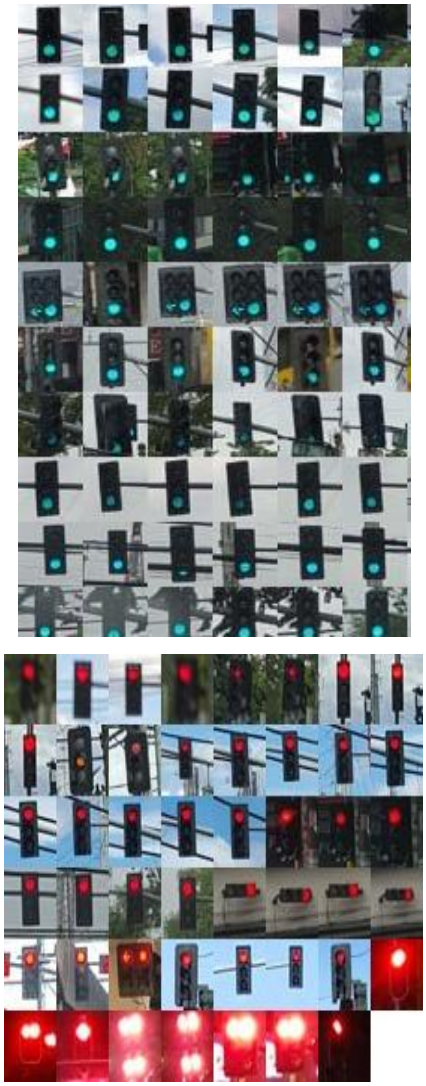

Figure 2. A part of the collection of Green traffic lights and Red traffic lights that were originally intended for use as testing data. The image is not to scale. Actual size for traffic light is 35x35 pixels.

IV. DATA AND RESULTS

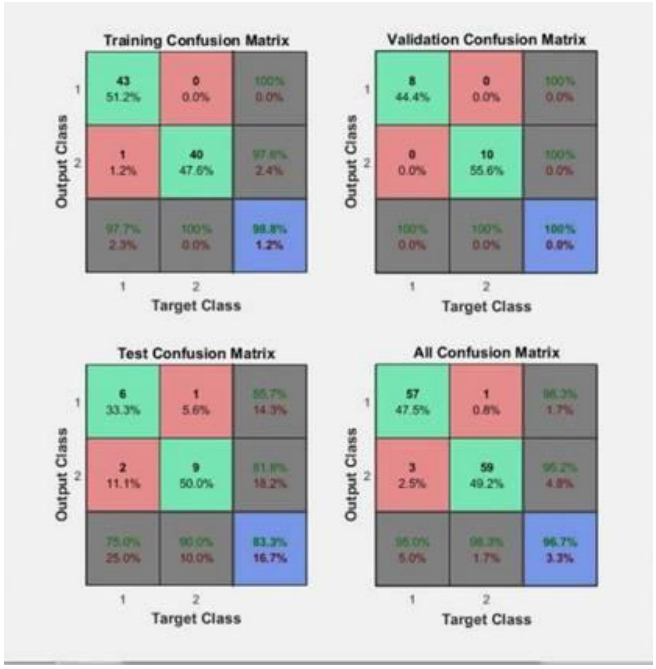

Figure 3. Confusion matrix for the 1st run

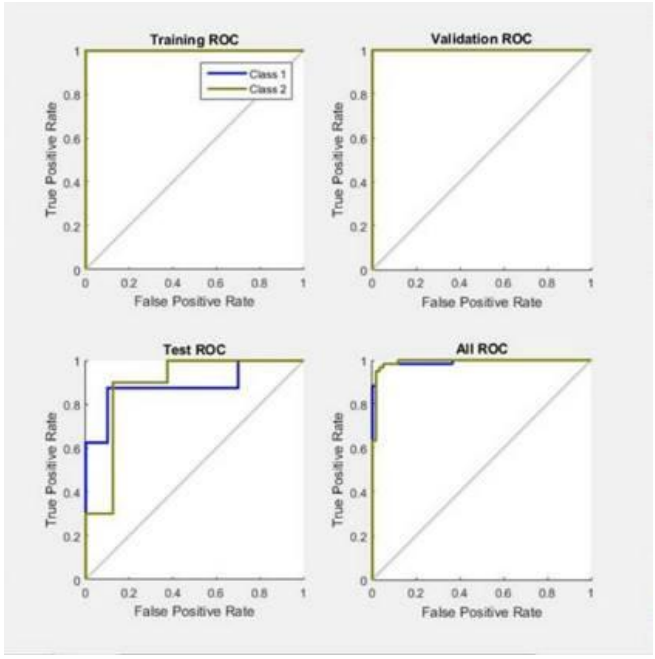

Figure 4. Receiver Operating Characteristic for the 1st run

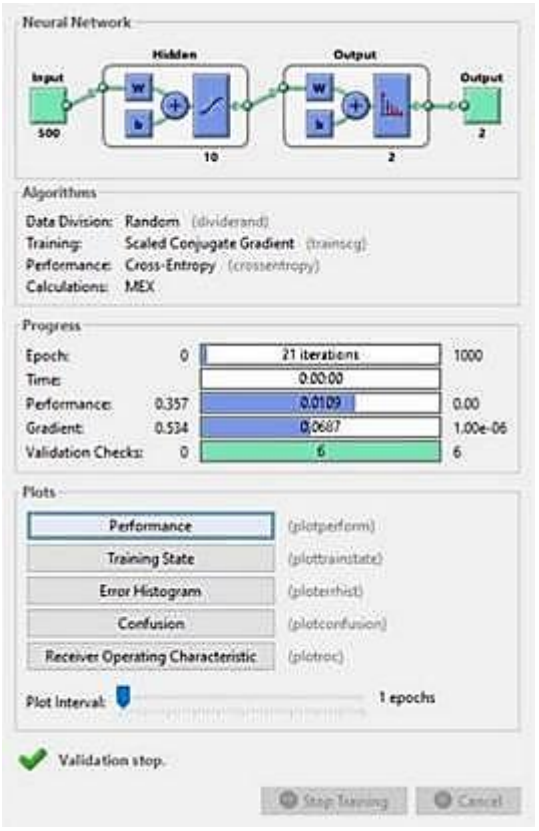

Figure 5. Details of the training done using the Neural Network Toolbox for the 1st run

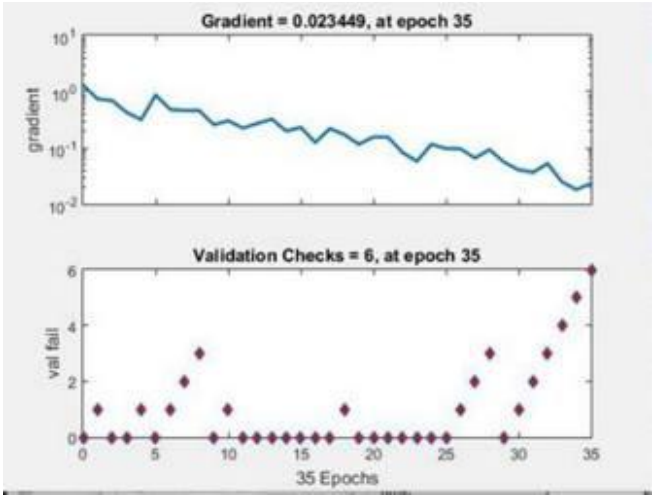

Figure 6. Validation reuslts for the 1st run

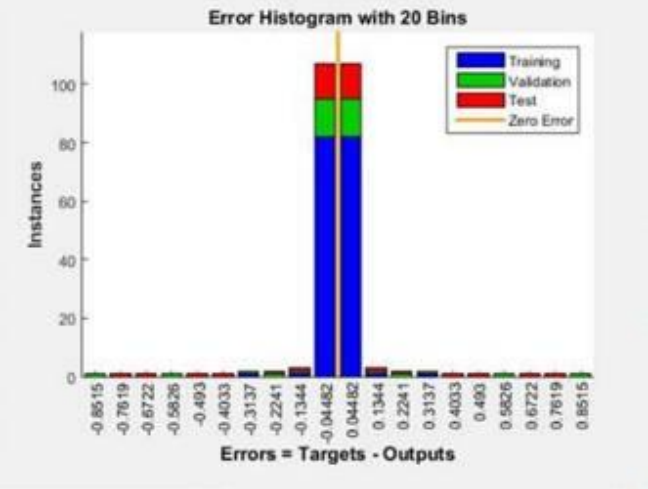

Figure 7. Error Histogram for the 1st run

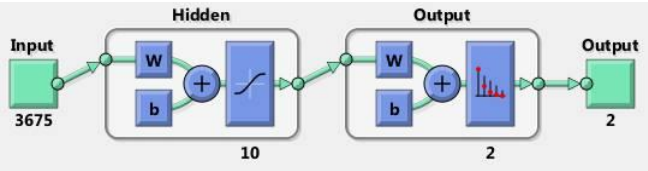

Figure 8. Generated neural network for the 2nd run.

Figure 9. Confusion Matrix for the 2nd run

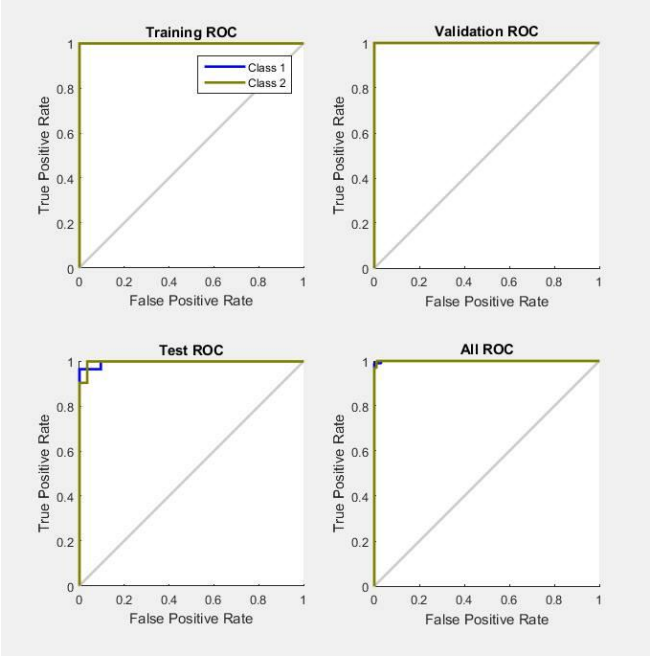

Figure 10. Receiver Operating Characteristic for the 2nd run

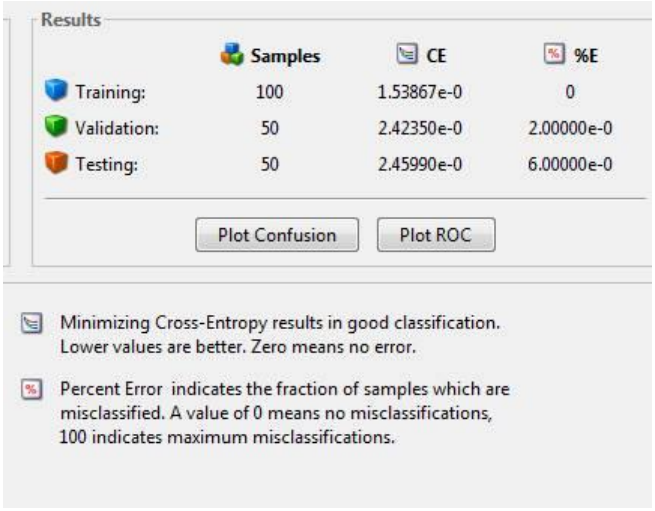

Figure 11. Error calculations for the 2nd run

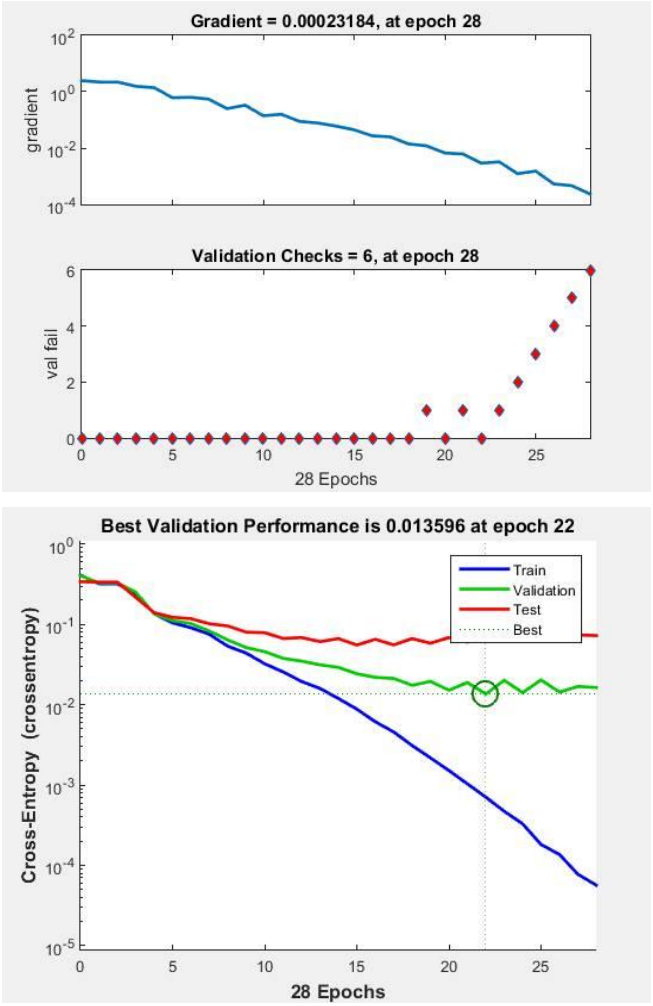

Figure 12. Validation results for the 2nd run

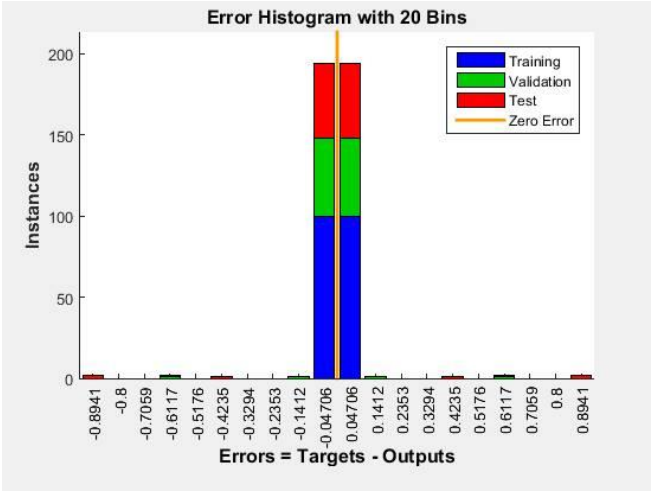

Figure 13. Error Histogram for the 2nd run

V. Analysis and Conclusion

The procedure did not go as easily as expected but it indeed cut off significant time and effort. The problem is that the neural network toolbox cannot work with the originally prepared training data, stating that there are too many features/pixels in the data set. The work around is to convert the image to grayscale. Another problem was that the toolbox takes up too much resource during its operation. Initially, it was ran on a laptop running on 2.2GHz dual-Core CPU with 2GB of RAM, but this proved to be insufficient as the system ran out of memory thus halting the training. There was also an attempt to run it on a 2.7GHz single-core desktop with 4GB RAM but only ended up hanging within a few seconds of training. The last resort was to run it on a computer running on Intel i5-4460 with 8GB. The execution time was roughly less than 10 seconds.

The training was successfully completed with the modified data set and it being run on a relatively high-spec computer. However, the testing was automatically done by the neural network toolbox. From the data set injected to the neural network, the application, by default, picks the 70% for training, 15% for validation and 15% for testing. This ultimately made the actual shots useless as part of the data that are supposed to be solely for training were used for validation and testing. The training returned a high 96% accuracy while the testing recorded 83% - both of which can be considered good (accuracy > 80%). With this result, it can be concluded the neural network is a success.

However, the group wanted to reassure the accuracy of the results. The Red traffic lights that were gathered from Google Images and those that were gathered in actual, were all mixed in a single folder. Same procedure was done for the Green Traffic lights. From there, 100 Red traffic lights and 100 green traffic lights were chosen randomly. Moreover, the images were no longer converted to grayscale thus requiring greater processing power. A second run was done with this newly formed data set. This time the computer used has an i7-2600 CPU at 3.40GHz and an 8GB RAM. The results are shown on Figures 9 to 13. The 2nd run produce more conclusive results and an even higher accuracy.

VI. Recommendations

Although considered as a success in its own accord, the results of this project cannot guarantee its accuracy in a variety of traffic lights as the data set used can still be considered abridged, thereby limiting the neural network's learning potential. Most of the traffic lights present on the data set are more or less of the same type or shape. There are other shapes of traffic lights that were not included in the data set, such as those that has 4 sides, each side facing the road at an intersection. There are also those that has a different color (i.e. body color of traffic light is yellow). Traffic lights in operation at night were also not present on the data set. To

have a greater accuracy on a greater scope, a huge database must also be used.

In addition, the resources should be effectively managed to use MATLAB's Neural Network Tool. It might still be a good idea to rather hard code the neural network than utilize a built-in application. This is where engineering decision making comes into play in order to balance the trade-off (implementation simplicity versus resources).

APPENDIX

Preparing the Training data that is injected to the Neural Network

```
% Script for preparing the Training data

% Clear screen and workspace
clc, clear all

% For Red Traffic lights
cd C:/Users/Gervin/Desktop/CP29_Project/Traffic_Lights/Red/red_all/ % Local path of red
traffic lights
redTrain = dir('*.jpg');
for k = 1:length(redTrain)
    filename_red = redTrain(k).name;
    I = imread(filename_red);
    R(:,k) = reshape(I,1,[]); % each entry containing a 35x35-pixel image reshaped into
a vector
end
clear I k

% For Green Traffic lights
cd C:/Users/Gervin/Desktop/CP29_Project/Traffic_Lights/Green/green_all/ % Local path of
green traffic lights
greenTrain = dir('*.jpg');
for k = 1:length(greenTrain)
    filename_green = greenTrain(k).name;
    I = imread(filename_green);
    G(:,k) = reshape(I,1,[]); % each entry containing a 35x35-pixel image reshaped into
a vector
end
clear I k

% Concatenating redTrain and greenTrain
newTrainData = [R G]; % previous training data is few and only in grayscale
```

"Advance Script" Generated through Neural Network Toolbox

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created 02-Dec-2015 18:16:50
%
% This script assumes these variables are defined:
%
%   newTrainData - input data.
%   label - target data.

x = newTrainData;
t = label;

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
```

```
trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.

% Create a Pattern Recognition Network
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 50/100;
net.divideParam.valRatio = 25/100;
net.divideParam.testRatio = 25/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'crossentropy';  % Cross-Entropy

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotconfusion', 'plotroc'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotconfusion(t,y)
```

```
%figure, plotroc(t,y)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end
```

"Simple Script" Generated using Neural Network Toolbox

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created 02-Dec-2015 18:16:47
%
% This script assumes these variables are defined:
%
%   newTrainData - input data.
%   label - target data.

x = newTrainData;
t = label;

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.

% Create a Pattern Recognition Network
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 50/100;
net.divideParam.valRatio = 25/100;
net.divideParam.testRatio = 25/100;

% Train the Network
```

```
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotconfusion(t,y)
%figure, plotroc(t,y)
```