The True Dress Code Monitor

CPELEC1 – Project

Ryan Joshua Liwag
ryan_liwag@dlsu.edu.ph
11220392

Anfernee Rapio
anfernee_rapio@dlsu.edu.ph
11217707

Department of Electronics and Communications Engineering
Gokongwei College of Engineering, De La Salle University
Manila, Philippines

***Abstract*— To be able to monitor students within De La Salle University with regards to the dress code policy by creating a program or application which uses image processing and Artificial Neural Network. The output of the program will show and highlight the student that would violate the dress code.**

***Keywords*—Dress code, Artificial Neural Network, image processing, data and results, error**

I. Introduction

Dress code policy is a rule being implemented among schools and universities with regards to the outfit of the students. With that being said there are also failures on how it is being implemented among the students. For example, other students are still being able to pass through the security even if they are already violating the dress code policy. Also, other security guards fail to implement the policy as they forgot the necessary details and information about it. Other security guards also fail to do it as there are high traffic of students in the entrance therefore giving them a hard time on checking the proper outfit of the students. To be able to properly implement the policy, there must be additional help to the guards for them to monitor it completely. With this problem in the university, we are going to create a program wherein it monitors the outfit among students to see if they are properly implementing the policy. This program also monitors if the student isn't wearing ID. The program will need to implement the Artificial Neural Network for it to learn to monitor dress codes efficiently.

II. Objectives

- To be able to apply Artificial Neural Network on a program or application
- To be able to apply Artificial Neural Network on a specific problem in De La Salle University

III. methodology

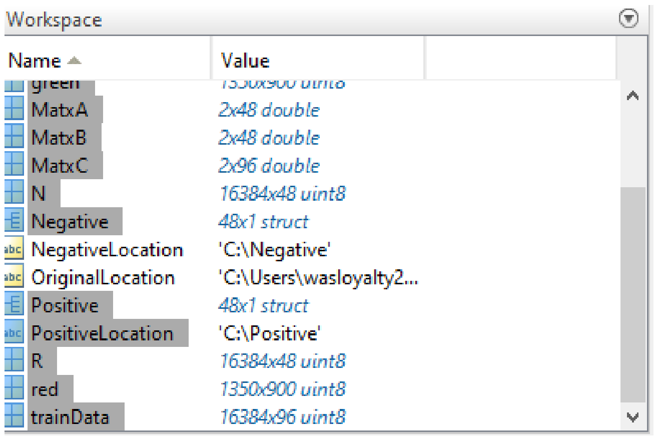
For the policy to be implemented correctly on De La Salle University, a program would be created to help the security guards of the institution to monitor the outfit of the students. To be able to create the program necessary for the solution of the problem, MATLAB programming language is going to be used. Also, for the program to be created and coded properly it is recommended to use operating systems that use Linux For the input of the data, a sample photo of students are needed to scan through if there would be a violation of the policy. For the artificial neural network part, we are going to apply machine intelligence on the program so that it would monitor students with the absence of the user. With these requirements getting accomplished, the program would be efficient enough for it to monitor the outfit of the entering students in the university even if there would be absence of the user.

IV. Data and Results

For the program to work properly, training data is required for the machine to learn a task. In this project, the training data is the acquired image samples of multiple person wearing a proper and improper uniform. People that wear proper uniform are labeled as positive while the others are labeled as negative. A matrix is then created to be able to compare the amount of images taken as sample. After getting the matrix, the sample images are then converted into greyscale and would have its resolution minimized, this allows the program to easily train and for it to quickly analyze the images. To get the training data, the user needs to have the MATLAB IDE opened , change the current directory to the folder where the Load_train_data.m is and should type the following code below or run the program

- >>Load_train_data;

After running the code, the program would then prompt the user to select where the main directory of the MATLAB files are or the folder which contains the Load_train_data.m file. After that, the program asks the user to specify the folder that contains the positive images and then right after that another dialog appears which would ask where the negative images are. Then right away, the program would process the given data and would get it ready for the training. The sample image below would show the generated parameters after running the code above.

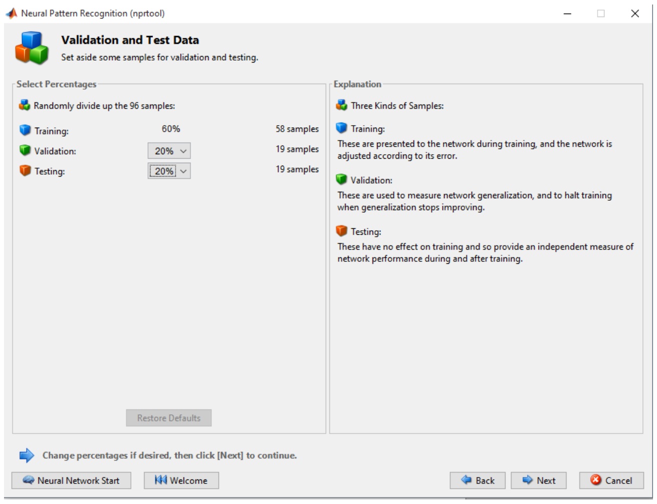


Workspace

Name	Value
green	1350x900 uint8
MatxA	2x48 double
MatxB	2x48 double
MatxC	2x96 double
N	16384x48 uint8
Negative	48x1 struct
NegativeLocation	'C:\Negative'
OriginalLocation	'C:\Users\wasloyalty2...
Positive	48x1 struct
PositiveLocation	'C:\Positive'
R	16384x48 uint8
red	1350x900 uint8
trainData	16384x96 uint8

Figure 1- The variables produced by running the Load_train_data

These variables would then be required, specifically the MatxC and the trainData, to test the data later on. In MATLAB there is a built-in function where in it utilizes ANN to train the machine on a given task by using the training data the user will provide. To run the said function, the user will need to type "nnstart" in the command window. This brings the user to the Neural Network Start window. Then by entering the parameters, inputs, and outputs that would be required the function would then display the graphs, tables and results.



Neural Pattern Recognition (nprtool)

Validation and Test Data

Set aside some samples for validation and testing.

Select Percentages

Randomly divide up the 96 samples:

Training:	60%	58 samples
Validation:	20%	19 samples
Testing:	20%	19 samples

Explanation

Three Kinds of Samples:

Training:
These are presented to the network during training, and the network is adjusted according to its error.

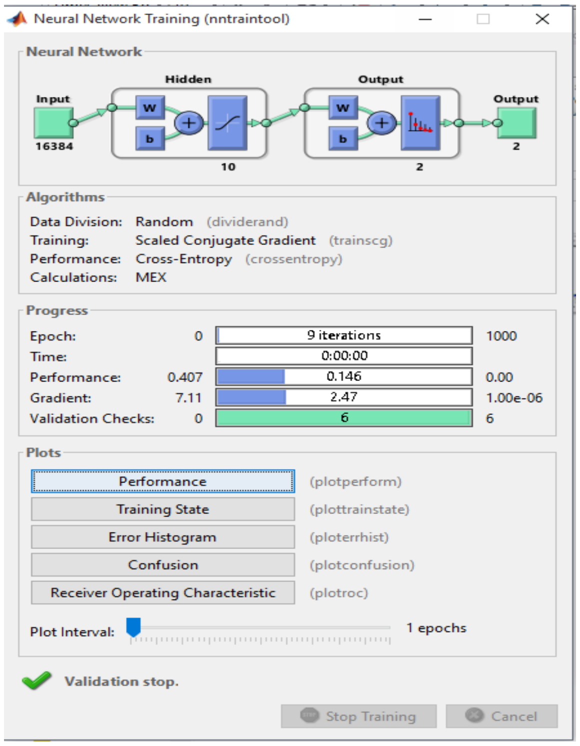
Validation:
These are used to measure network generalization, and to halt training when generalization stops improving.

Testing:
These have no effect on training and so provide an independent measure of network performance during and after training.

Change percentages if desired, then click [Next] to continue.

Figure 2- Dividing the training data into training, validation and testing by means of percentage from the total data input.

Running the first train, the image next is the yielded result



Neural Network Training (nntraintool)

Algorithms

Data Division:	Random (dividerand)
Training:	Scaled Conjugate Gradient (trainscg)
Performance:	Cross-Entropy (crossentropy)
Calculations:	MEX

Progress

Epoch:	0	9 iterations	1000
Time:		0:00:00	
Performance:	0.407	0.146	0.00
Gradient:	7.11	2.47	1.00e-06
Validation Checks:	0	6	6

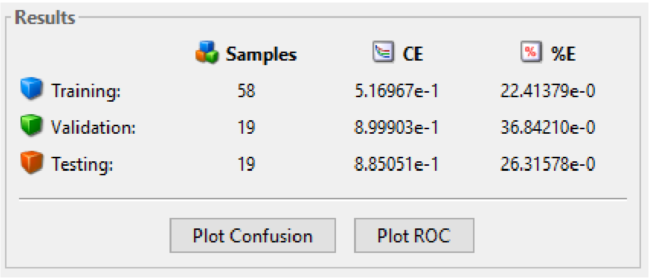
Plots

Performance	(plotperform)
Training State	(plottrainstate)
Error Histogram	(ploterrhist)
Confusion	(plotconfusion)
Receiver Operating Characteristic	(plotroc)

Plot Interval: 1 epochs

Validation stop.

Figure 3- The Neural Network Training



Results

	Samples	CE	%E
Training:	58	5.16967e-1	22.41379e-0
Validation:	19	8.99903e-1	36.84210e-0
Testing:	19	8.85051e-1	26.31578e-0

Plot Confusion
Plot ROC

Figure 4- The result

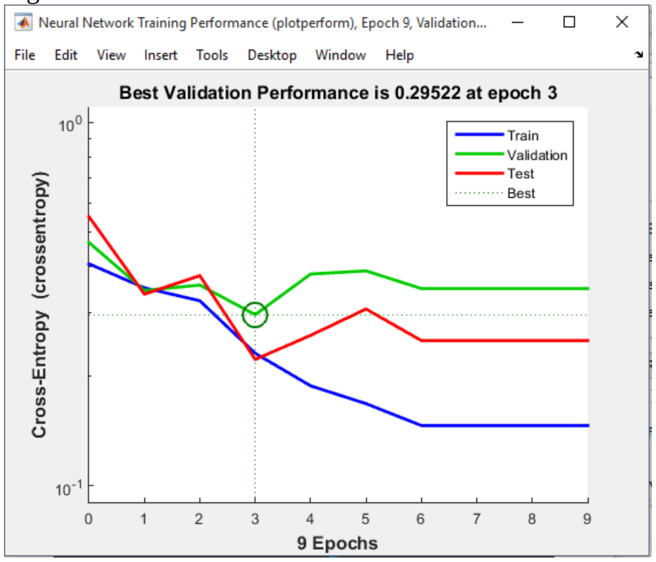


Figure 5- The performance chart or graph

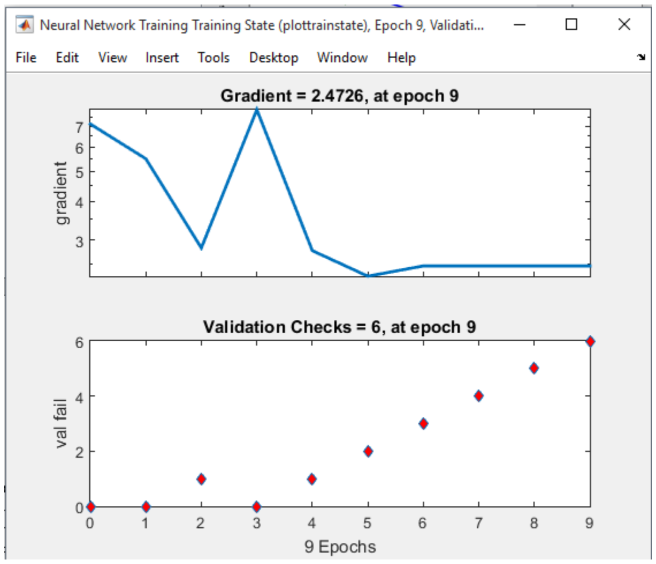


Figure6- The train state

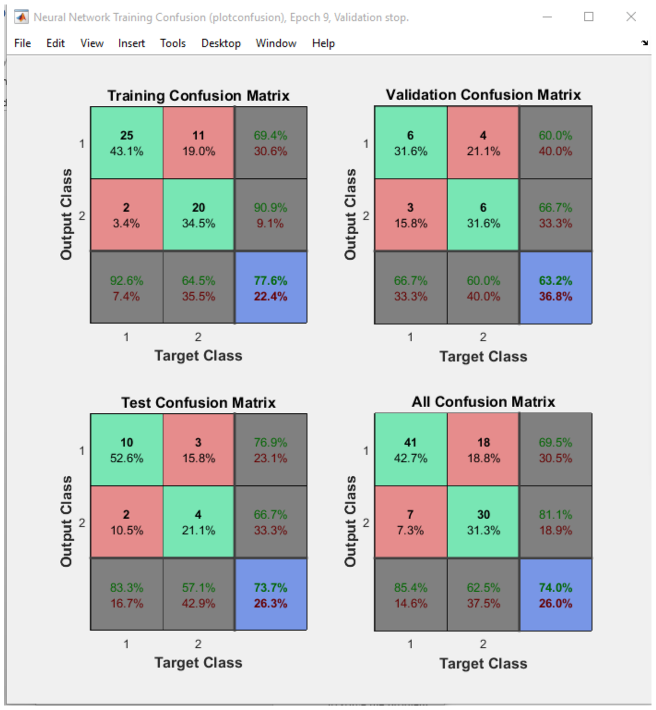


Figure 7- The confusion matrix

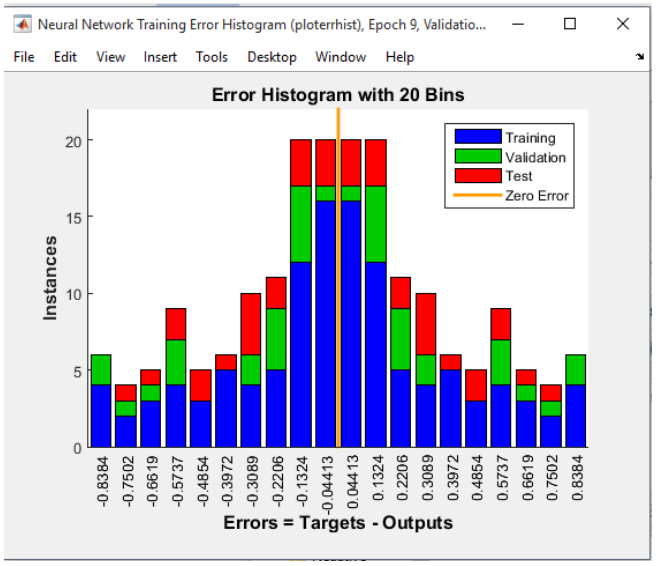
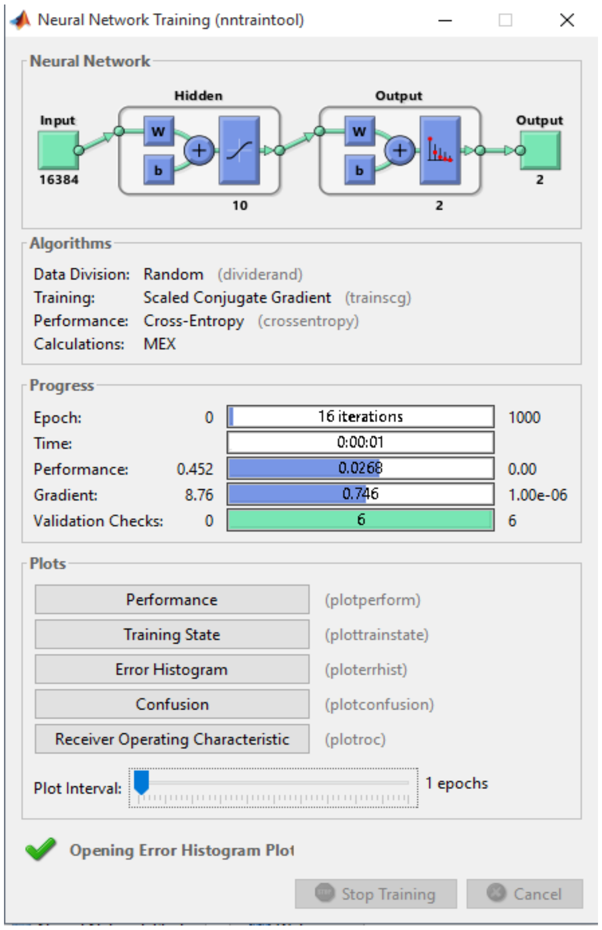


Figure 8- The error historgram

The images before are the results from the first training done to the program. To be able for the program to work properly a second training is recommended. The images below are the results of the second training


Neural Network Training (nntraintool)
Neural Network
Input
Hidden
Output
Output
16384
10
2
2

Algorithms

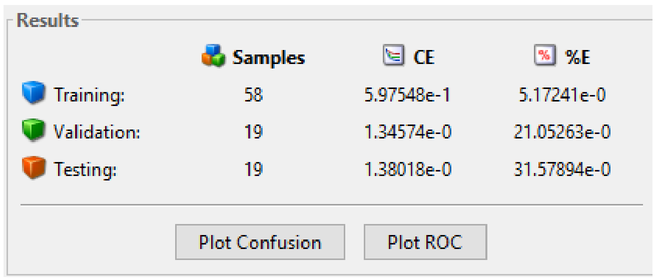
Data Division:	Random (dividerand)
Training:	Scaled Conjugate Gradient (trainscg)
Performance:	Cross-Entropy (crossentropy)
Calculations:	MEX

Progress

Epoch:	0	16 iterations	1000
Time:		0:00:01	
Performance:	0.452	0.0268	0.00
Gradient:	8.76	0.746	1.00e-06
Validation Checks:	0	6	6

Plots
Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Confusion (plotconfusion)
Receiver Operating Characteristic (plotroc)
Plot Interval: 1 epochs
Opening Error Histogram Plot
Stop Training
Cancel

Figure 9- Second Neural Network Training



Results

	Samples	CE	%E
Training:	58	5.97548e-1	5.17241e-0
Validation:	19	1.34574e-0	21.05263e-0
Testing:	19	1.38018e-0	31.57894e-0

Figure 10- The second training, validation and testing result

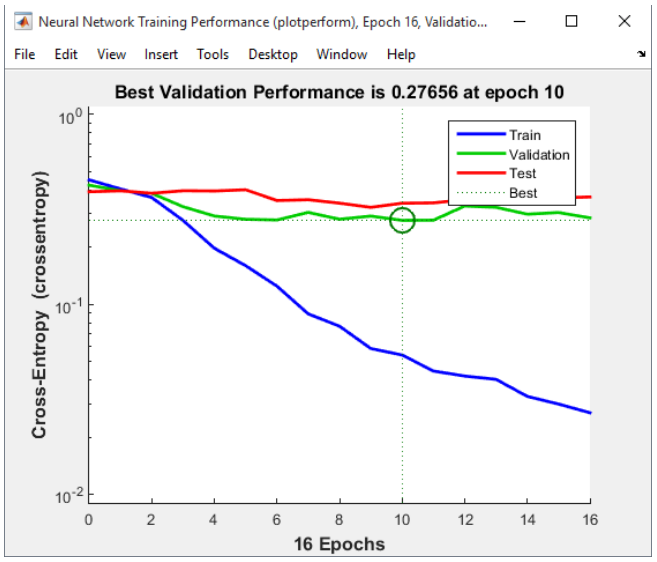


Figure 11- The second performance chart or graph

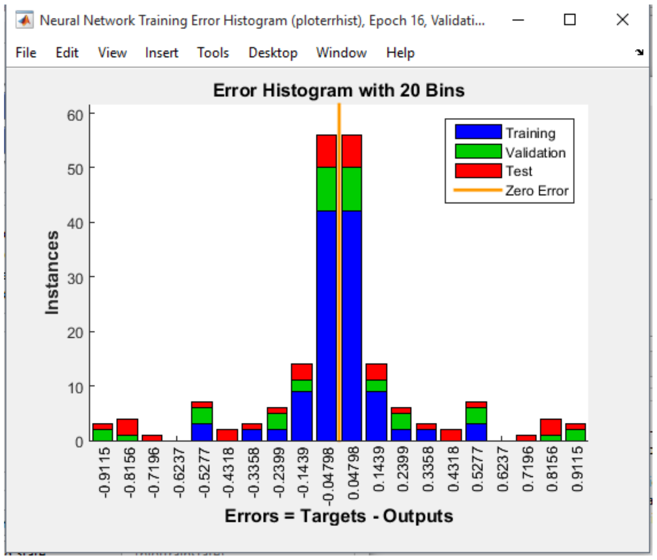
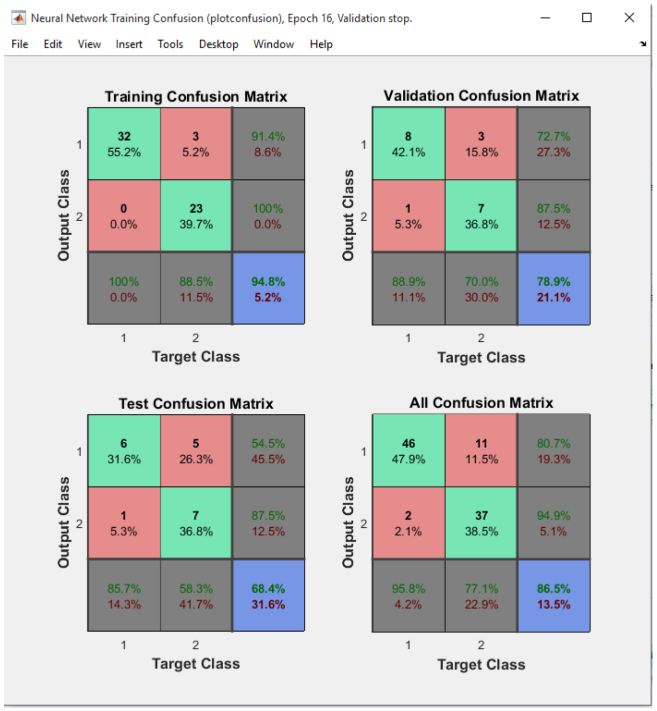


Figure 12- The second error histogram



Training Confusion Matrix

Output Class \ Target Class	1	2	
1	32 55.2%	3 5.2%	91.4% 8.6%
2	0 0.0%	23 39.7%	100% 0.0%
	100% 0.0%	88.5% 11.5%	94.8% 5.2%

Validation Confusion Matrix

Output Class \ Target Class	1	2	
1	8 42.1%	3 15.8%	72.7% 27.3%
2	1 5.3%	7 36.8%	87.5% 12.5%
	88.9% 11.1%	70.0% 30.0%	78.9% 21.1%

Test Confusion Matrix

Output Class \ Target Class	1	2	
1	6 31.6%	5 26.3%	54.5% 45.5%
2	1 5.3%	7 36.8%	87.5% 12.5%
	85.7% 14.3%	58.3% 41.7%	68.4% 31.6%

All Confusion Matrix

Output Class \ Target Class	1	2	
1	46 47.9%	11 11.5%	80.7% 19.3%
2	2 2.1%	37 38.5%	94.9% 5.1%
	95.8% 4.2%	77.1% 22.9%	86.5% 13.5%

Figure13- The second confusion matrix

On the results above we can see that there is a significant amount of change that occur on the tables, graphs and curves. This yielded into a better result on the training part although every retraining done would give the computer more load as it uses more workload.

V. Analysis and Conclusion

The learning procedure of the program did not go perfectly as planned but it contributed significantly to the learning of the program. Analysis of the image with colors would give problems to the learning so to compensate with it the images were turned into black and white or greyscale. The resolution of the images are also decreased for the learning process to be more efficient. As the retraining process continues, the IDE would use more resources on the computer therefore increasing its workload and decrease the free memory bandwidth.

To assure the effectiveness, easiness and efficiency of the training, the group properly designed the user interface of the program to separate and gather the training data needed. The images that show off proper and improper outfit were filtered and result into better training.

With the data and results giving a detailed information on how the program works, we conclude that it would help the security personnel to implement the policy in De La Salle University. With the help of training data the program was able to learn the process that would need in order to solve the problem.

VI. BIBLIOGRAPHY

[1] "Neural Network Toolbox," [Online]. Available: http://www.mathworks.com/products/neural-network/.

[2] "train," [Online]. Available: http://www.mathworks.com/help/nnet/ref/train.html. [Accessed 29 November 2015].

APPENDIX

Source Code for gathering the training data

```
% Script for creating the Training data
% Clear screen and workspace
clc, clear all
OriginalLocation = uigetdir('C:\ ', 'Select the Reference Folder (Where the train data
matlab file is)');
%global MatxC
%global trainData
%Positive Images
%cd C:\Positive
PositiveLocation = uigetdir('C:\ ', 'Enter Location of Positive Images');
cd(PositiveLocation);
% Local path
%^Change the directory to the folder with the training samples
%Positive means the proper images
Positive = dir('*.jpg');
for k = 1:length(Positive)
    filename_red = Positive(k).name;
    I = imread(filename_red);
    red = I(:,:,1); % Red channel
    green = I(:,:,2); % Green channel
    blue = I(:,:,3); % Blue channel
    I = rgb2gray(I); % convert to grayscale
    P = imresize(I, [128 128]);
    R(:,k) = reshape(P,1,[]); %
end
clear P I k
% NEGATIVE IMAGES
NegativeLocation = uigetdir('C:\ ', 'Enter Location of Negative Images');
cd(NegativeLocation)
%^Change the directory to the folder with the training samples
%Negative means the improper images
Negative = dir('*.jpg');
for C = 1:length(Negative)
    File_Negative = Negative(C).name;
    I = imread(File_Negative);
    red = I(:,:,1); % Red channel
    green = I(:,:,2); % Green channel
    blue = I(:,:,3); % Blue channel
    I = rgb2gray(I);
    P = imresize(I, [128 128]);
    N(:,C) = reshape(P,1,[]);
end
clear P I k
MatxA = [ones(1,length(Positive)) ; zeros(1,length(Positive))];
MatxB = [zeros(1,length(Negative)) ; ones(1,length(Negative))];
MatxC = [MatxA MatxB];
cd(OriginalLocation)
trainData = [R N];
%CPELEC1Project;
%CPELEC1ProjectAdvance;
```

Source Code for the simple script generated

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created 03-Dec-2015 12:03:52
%
% This script assumes these variables are defined:
%
%   trainData - input data.
%   MatxC - target data.

x = trainData;
t = MatxC;

% 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 = 90/100;
net.divideParam.valRatio = 5/100;
net.divideParam.testRatio = 5/100;

% 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)
```

Source Code for the advanced script generated

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created 03-Dec-2015 12:03:56
%
% This script assumes these variables are defined:
%
%   trainData - input data.
%   MatxC - target data.

x = trainData;
t = MatxC;

% 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 = 90/100;
net.divideParam.valRatio = 5/100;
net.divideParam.testRatio = 5/100;

% 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);
```

```matlab
% 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
```