Car Number Plate Recognition Using Neural Networks

49275 Neural Networks & Fuzzy Logic

Alex Horn
12461354

Table of Contents

Introduction

Background

Around the world, the large number of cars driven regularly has led to an increase in the need for automated systems capable of instantly recognizing and identifying cars by their existing number plates. By expanding the capability already provided by the existing number plate registration system, this real-time identification is capable of aiding in a range of applications including security, toll collection, and vehicle registration status monitoring. Due to the breadth of application and the need for precise, accurate results, artificial neural networks provide a solution to this need for reliable number plate recognition.

In this project we have investigated the feasibility of identifying number plates with a two layer artificial neural network, using error back propagation training.

Literature Search

Due to the growing interest in neural networks, and the significant need for vehicle identification, a fair amount of research has been conducted into the design and implementation of neural networks car number plate identification systems. This is in addition to the far greater amount of research into neural networks themselves, and their use in other applications.

These all follow a similar overall process, as shown in Fig 1.

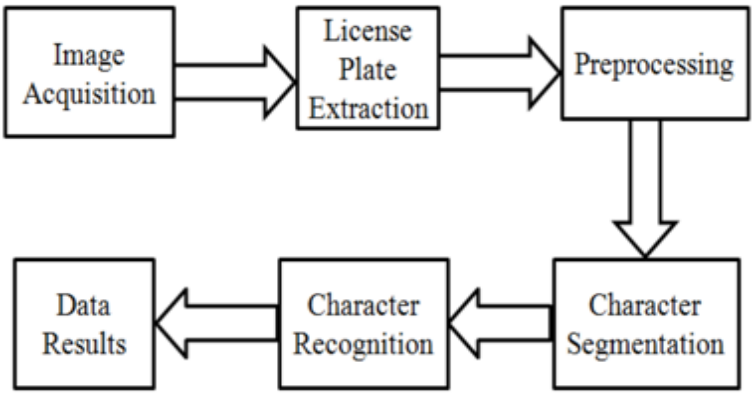


Figure 1 - Typical Stages of Licence Plate Recognition System (Laxmi & Rohil, 2014)

Each of these steps is necessary for recognition of number plates, though there may be some variation in their implementation. For example, Laxmi & Rohil (2014) used a high resolution digital camera to capture 1600x1200 pixel images, while real-world applications may come from a variety of sources, including fixed and mobile cameras, with varying degrees of consistency across captured images.

The image processing and neural network steps have even more room for flexibility, with their design making up the more significant part of neural network implementation. Kocer & Cevik (2010) process images using a combination of contrast enhancement, blob detection, and classification of letters and numbers based on plate location, while Akoum et al (2009) identify

characters by searching for the strong contrast of letters on number plates, a feature always present to improve legibility.

Implementation of the neural network also varied between papers. Ventzas & Karras also divided their data into letters and numbers, and utilized the Radon transform in a feed forward back propagation neural network, while Bhushan & Singh (2013) made use of multithresholding with their neural network to achieve a recognition rate of 98.4%.

In all papers, the researchers were able to achieve successful license plate identification using neural networks, suggesting that this task should be achievable to a significant degree of success.

Available equipment

No special hardware was required for this project.
However we did need to use calculation software, and elected to use MATLAB for this purpose. This was selected because it was readily available, easy to learn, and well documented with plenty of further information available online. It also offered useful additional packages, including the Image Processing Toolbox and the Computer Vision Toolbox which were used during the image processing stage.
Image acquisition was carried out by manually selecting images from carsales.com.au. One of the requirements of this website is that all listing include a clear photo of the number plate, which provided access to a large library of images to choose from.

Project Specifications

License plates include all 26 letters from A to Z, and all 10 digits from 0 to 9. This makes a total of 36 characters to be identified.
Early advice given to us suggested that we would need to gather at least 100 to 200 images to obtain enough images of each individual letter. It was also advised that we should focus on a smaller set of characters to identify within the total, and that it may be far more feasible to develop a neural network capable of identifying 10 characters rather than the total 36.

Image Processing

In order to proceed to neural learning and classification, it is necessary to process input images to extract the relevant areas and create a set of consistent data. Our aim during this stage was to identify and extract individual letters, in a consistent size and in binary format, which would allow us to create arrays with uniform dimensions. A simple way to do this would be to manually select and crop individual letters from images, however due to the volume of data required, this would be a long and tedious task, and so we aimed to automate this process. This would have the additional benefit of allowing live tests to be conducted.

We used the MATLAB Image Processing and Computer Vision toolboxes to process the images obtained from Carsales. We used the following process to process images and obtain fairly accurate, consistent results. There was a great deal of trial and error involved, particularly in the filtering of boundary boxes, as we needed to determine a balance between strict filtering that would eliminate desired filters, and relaxed filtering that would include false locations.

Image Processing Method

Our procedure for image processing followed these steps:

1. Obtain source image (from Carsales.com.au)

Figure 2 - Example source image

2. Perform initial region detection. This is performed using the MSERRegions function in the MATLAB Computer Vision Toolbox. This is a method of region detection that was found to be quite suitable for identifying the number plate region, but would also include many unwanted regions.

Figure 3 - Regions identified using the MSERRegions function

3. Filter out unwanted regions. This is performed by creating Boundary Boxes for the discovered regions, and then filtering these out using a combination of:

a. Aspect ratio. Number plates are assumed to have a consistent aspect ratio, and any regions with different shape are excluded. Some allowance is made for photo angle.
b. Location within image. From the gathered images, we found that numberplates were almost always located around the middle, and slightly low.
c. Proportion of total image. Number plates only take up a small section of a car image.

Figure 4 - Regions after filtering. Note the border around the number plate

4. Crop out number plate

Figure 5 - Number plate cropped from source

5. Binarize the number plate image, trim borders, and resize to fixed height. This converts the image to pure black and white, which is necessary for neural processing. This is performed by determining the average grey level in the image, and using this as a threshold to exclude areas that are too light. Borders are trimmed to remove unwanted details.
The image is resized to a fixed height of 20 pixels, in order to provide consistent data for the neural processing stage.

Figure 6 - Binarized and trimmed number plate

6. Crop characters from image. This is performed by examining the histogram of the binarized image and identifying the regions that match the expected width of letters. The resulting characters are 18x20 pixels.

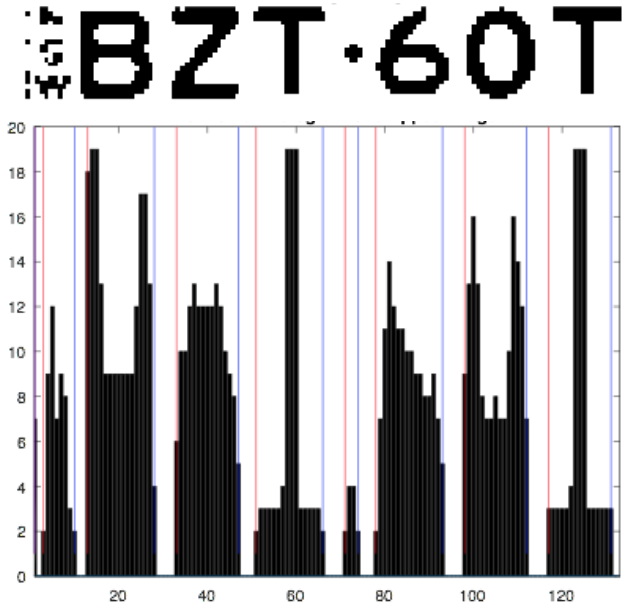


Figure 7 - Histogram of number plate image

Figure 8 - Individual characters cropped

7. Manual identification. The individual letters are identified manually, and are then added to character sets that may be used for training, validation, or testing. These are converted from binary to bipolar, and saved as arrays with length 360.

Training and Validation Data

The total number of letters obtained for training and validation are shown in the table below. For each character, the number of lines refers to the number of samples. This is less than the total amount of available data, as errors present in the image processing stage meant that some characters were not properly extracted from their source images.

Training Data	**Validation Data**
Letter_0.csv : 26 lines	Letter_0.csv : 10 lines
Letter_1.csv : 49 lines	Letter_1.csv : 11 lines
Letter_2.csv : 35 lines	Letter_2.csv : 11 lines
Letter_3.csv : 31 lines	Letter_3.csv : 5 lines
Letter_4.csv : 35 lines	Letter_4.csv : 9 lines
Letter_5.csv : 39 lines	Letter_5.csv : 9 lines
Letter_6.csv : 38 lines	Letter_6.csv : 7 lines
Letter_7.csv : 31 lines	Letter_7.csv : 7 lines
Letter_8.csv : 50 lines	Letter_8.csv : 10 lines
Letter_9.csv : 30 lines	Letter_9.csv : 5 lines
Letter_A.csv : 28 lines	Letter_A.csv : 26 lines
Letter_B.csv : 42 lines	Letter_B.csv : 8 lines
Letter_C.csv : 49 lines	Letter_C.csv : 2 lines
Letter_D.csv : 39 lines	Letter_D.csv : 5 lines
Letter_E.csv : 14 lines	Letter_E.csv : 4 lines
Letter_F.csv : 21 lines	Letter_F.csv : 1 lines
Letter_G.csv : 24 lines	Letter_G.csv : 4 lines
Letter_H.csv : 19 lines	Letter_H.csv : 4 lines
Letter_I.csv : 9 lines	Letter_I.csv : 2 lines
Letter_J.csv : 14 lines	Letter_J.csv : 6 lines
Letter_K.csv : 14 lines	Letter_K.csv : 3 lines
Letter_L.csv : 19 lines	Letter_L.csv : 3 lines
Letter_M.csv : 12 lines	Letter_M.csv : 4 lines
Letter_N.csv : 12 lines	Letter_N.csv : 6 lines
Letter_O.csv : 8 lines	Letter_O.csv : 3 lines
Letter_P.csv : 13 lines	Letter_P.csv : 1 lines
Letter_Q.csv : 11 lines	Letter_Q.csv : 5 lines
Letter_R.csv : 16 lines	Letter_R.csv : 3 lines
Letter_S.csv : 25 lines	Letter_S.csv : 4 lines
Letter_T.csv : 25 lines	Letter_T.csv : 2 lines
Letter_U.csv : 13 lines	Letter_U.csv : 2 lines
Letter_V.csv : 19 lines	Letter_V.csv : 4 lines
Letter_W.csv : 18 lines	Letter_W.csv : 7 lines
Letter_X.csv : 18 lines	Letter_X.csv : 7 lines
Letter_Y.csv : 30 lines	Letter_Y.csv : 8 lines
Letter_Z.csv : 23 lines	Letter_Z.csv : 6 lines

Neural Network System Design

The neural model used for this project is a multilayered network, with 1 input layer, 1 hidden layer, and one output layer. Number plate data, in this case the individual characters within a number plate, are fed to the input layer, which is then passed forward to the hidden layer using the bipolar logistic function.

$$v = \overline{W}'x$$

Hidden layer output signal = $y = f(v\square) = \frac{1-e^{-v}}{1+e^{-v}}$

Where x is the input layer, $\overline{w}$ is the weight vector.

This process is repeated again to pass the hidden layer to the output layer.

During training, the results at the output layer are compared with the desired classification, and the error is passed back to update the weight from the hidden layer into the output layer:

$$Error\ signal\ term\ \delta = \ ½(d-z)(1\ -z^2\ )$$

$$Updated\ Weight\ W^* = W + \eta\ \delta y'$$

Where d=desired output, z=output layer value, and $\eta$is the learning constant

This is then passed back again to update the weight from the input layer into the hidden layer

$$Error\ signal\ term\ \ \bar{\delta} = \frac{dy}{dv}\sum \delta\ dw$$

$$Updated\ weight\ \overline{W^*} = \overline{W} + \eta\ \delta x'$$

This passing back of the error is known as Error Back Propagation, and is a supervised form of neural network training.
It is necessary to repeat these steps many times, as the learning process is incremental. The accuracy of the system increases with repetition, although it is possible to over-train the system to the point that the ability to classify data outside of the training set is reduced. It is therefore necessary to perform validation tests during the training stage with a separate set of data to ensure that this does not occur, and stop training early if this is observed.

The system can be represented with the following diagram.

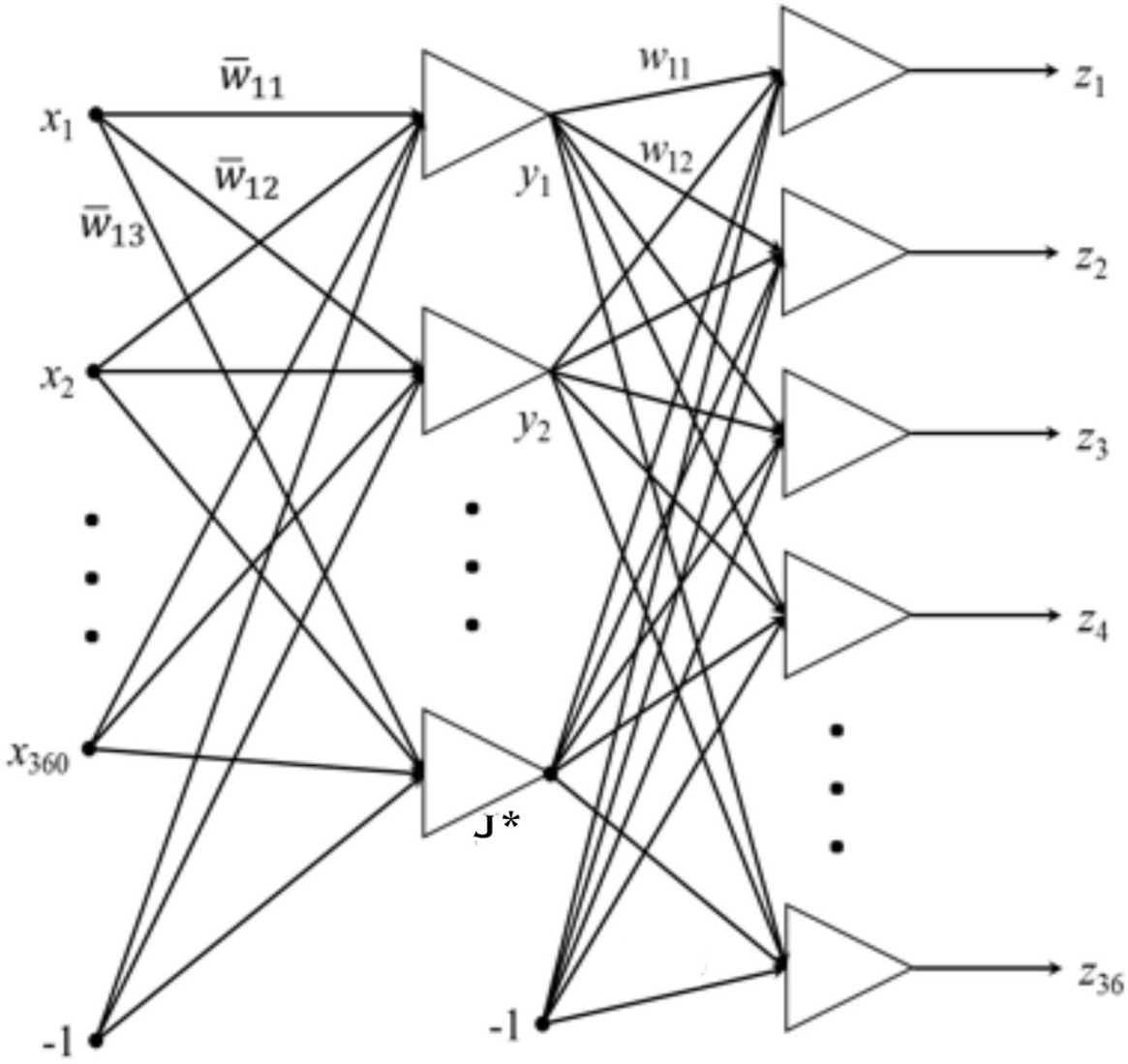


Figure 9 - Layered neural *network*

As shown in this diagram, the input layer has 361 nodes. These are the 18x20 pixels of each character, plus one augmented input node.

The output layer has 36 nodes. These correspond to the 36 desired outputs for classification of every character. It is possible to classify all characters using as low as 6 nodes with distributed representation, where the total binary output of all nodes corresponds to one output. This requires at least 6 nodes, as the number of binary outputs is $2^6 = 64$. However, we chose to use local representation, in which each node corresponds to a single classification.
With K=36 output nodes, the maximum number of separated regions M is given by

$$M = (K^2 + K + 2)/2 = 667.$$

The minimum number of hidden nodes can then be found using $M = 2^{J^*}$, where the number of hidden nodes $J = J^* + 1$. With M=667, this gives a minimum J^*= 10.

Neural Network System Implementation

Number of Output Nodes

As mentioned, it was recommended that we start by developing the network for a reduced number of characters. This is how the system was initially developed, and this was quite successful. However, we had developed the code with scaling in mind, and it was a simple process to expand the system to allow training and classification of the entire character set. Therefore, after some initial work to have the network functioning, all later development was based on the entire character set, using local classification, requiring 36 output nodes.

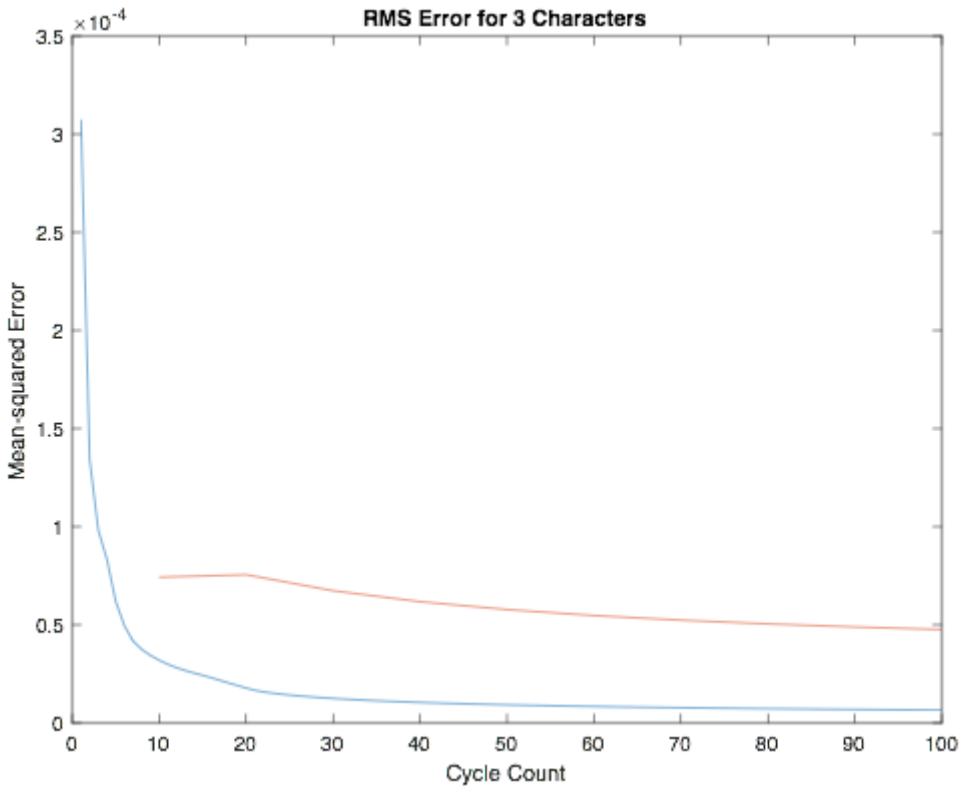


Figure 10 - Error for 3 character classification. n=0.2, 10 nodes
Note: Blue is the training data, red is the validation set.

Number of Hidden Nodes

The number of hidden nodes will affect the accuracy of the output, and it is not necessarily the case that more nodes will increase accuracy. It is possible for this to lead to overtraining, and a reduced ability to correctly classify test data. Increasing the number of nodes also increases the computing power needed by the neural network, so it is preferred to keep this number low if possible.

Initial weight vectors were created for the range of hidden nodes J^*= 10 to 30, and the neural network was trained using the test data and validation set. This was run for 300 cycles of the test data, with validation performed every 10 cycles. Ideally, validation would be performed every cycle, but this was limited to reduce calculation time. A learning constant of 0.2 was used, as this was found by trial and error to provide more accurate results.

Training and validation results were observed over this range of nodes, and the best performance was found at J^*= 17.

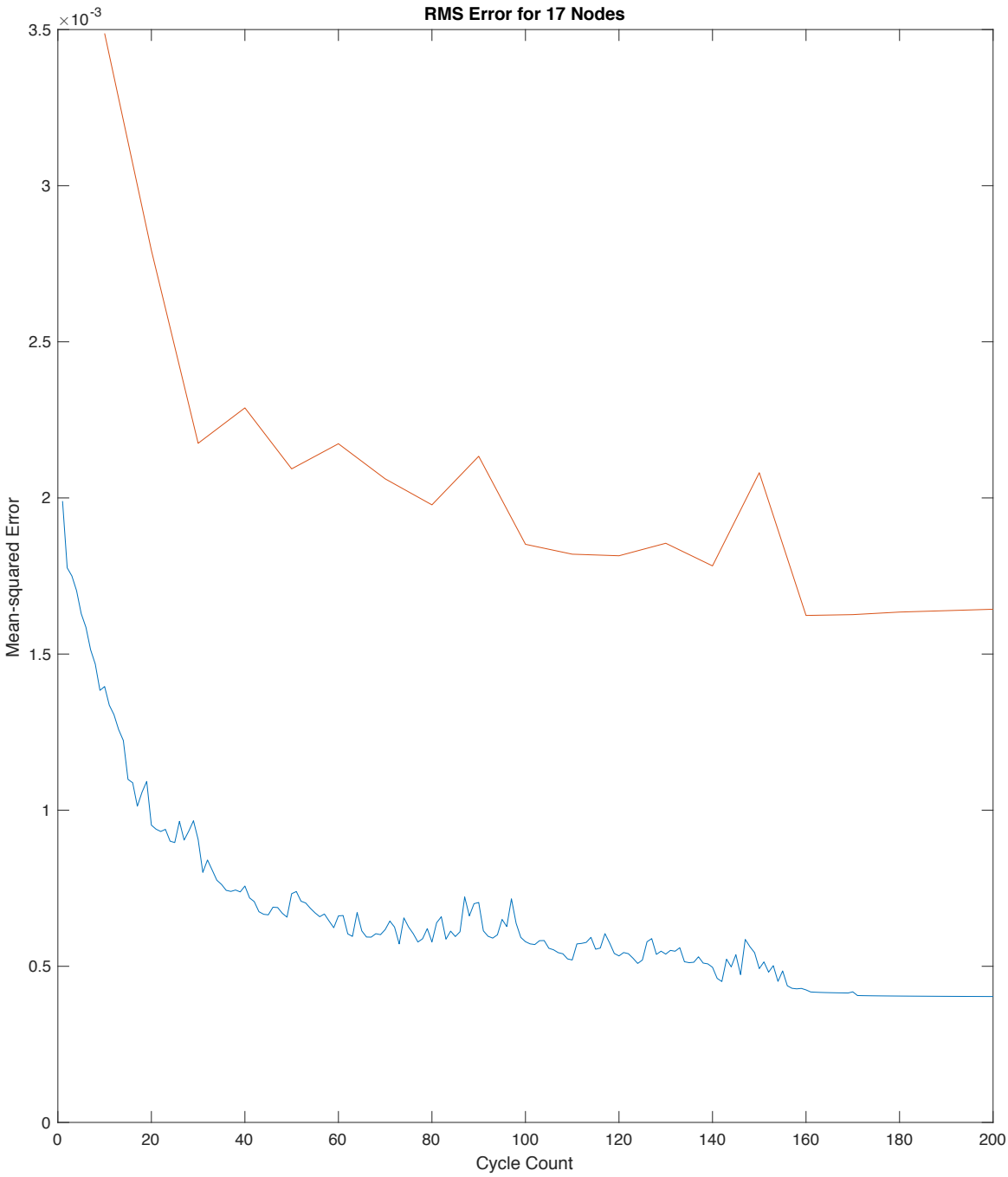


Figure 11 - Error for 17 nodes, n=0.2
Note: Blue is the training data, red is the validation set.

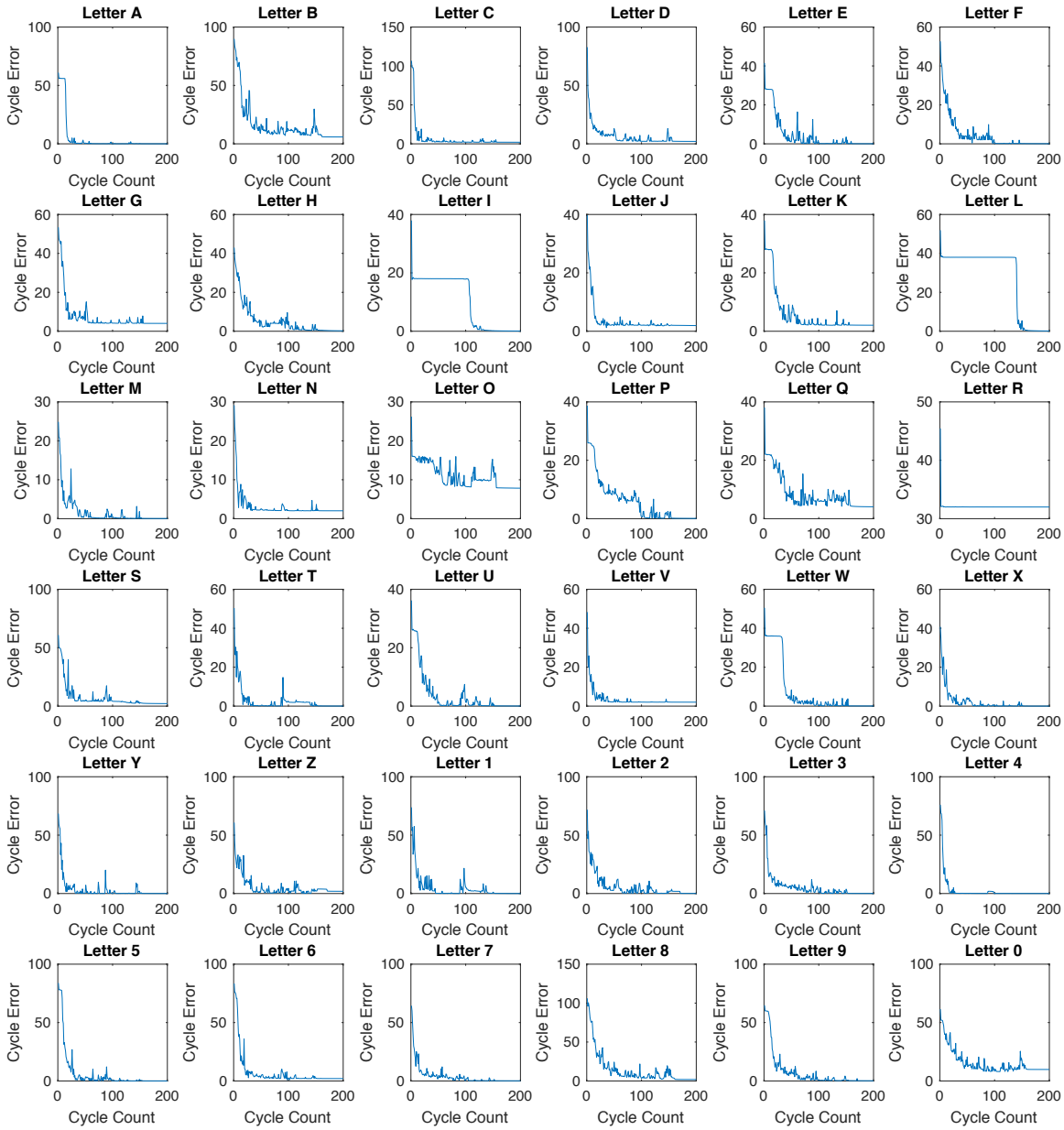


Figure 12 - Cycle Error for each character

Learning Constant

The use of learning constant 0.2 was confirmed by repeating the training with n=0.1 and n=0.3. We can see in the below graphs that the mean square error is worse for the validation tests, than at n=0.2.

N=0.1

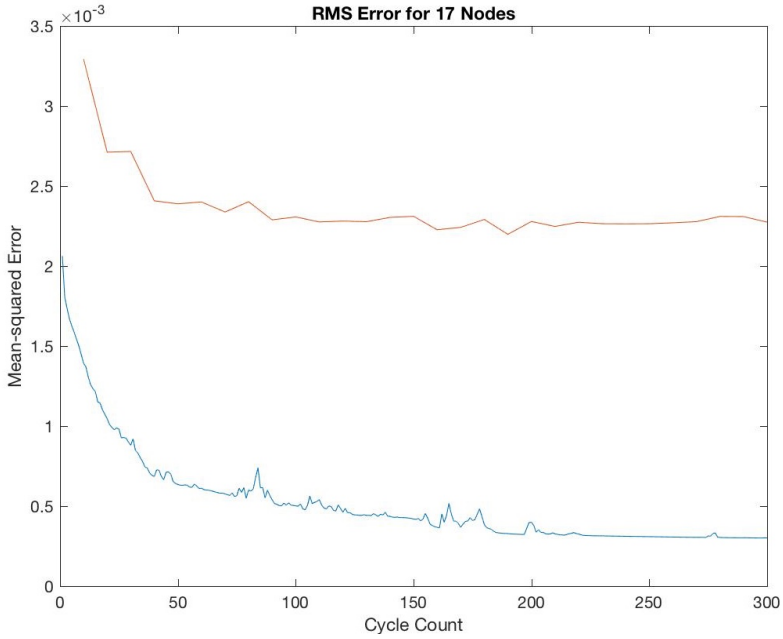


Figure 13 - Error for 17 nodes, n=0.1

Note: Blue is the training data, red is the validation set.

N=0.3

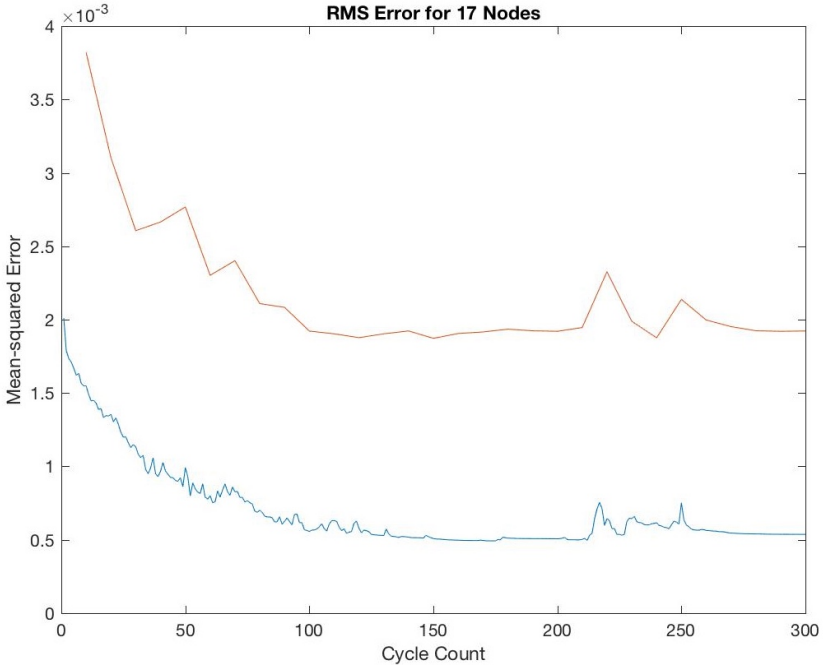


Figure 14 - Error for 17 nodes, n=0.3

Note: Blue is the training data, red is the validation set.

Results

Final Weight Vectors

With our neural network selected as 17 nodes, n=0.2 we can find the final weight vectors for use in testing. From the training results, we can see that the system is fairly stable after 160 cycles, for we can limit the training cycles to this range to reduce calculation and the slight overtraining that occurs.

This gives us the following hidden weights. On the left is the first row of the WB set of weight. On the right is the first row of the W set of weights. We can see that in all case, there is some initial large correction, which settles as the weights become closer to their ideal values.

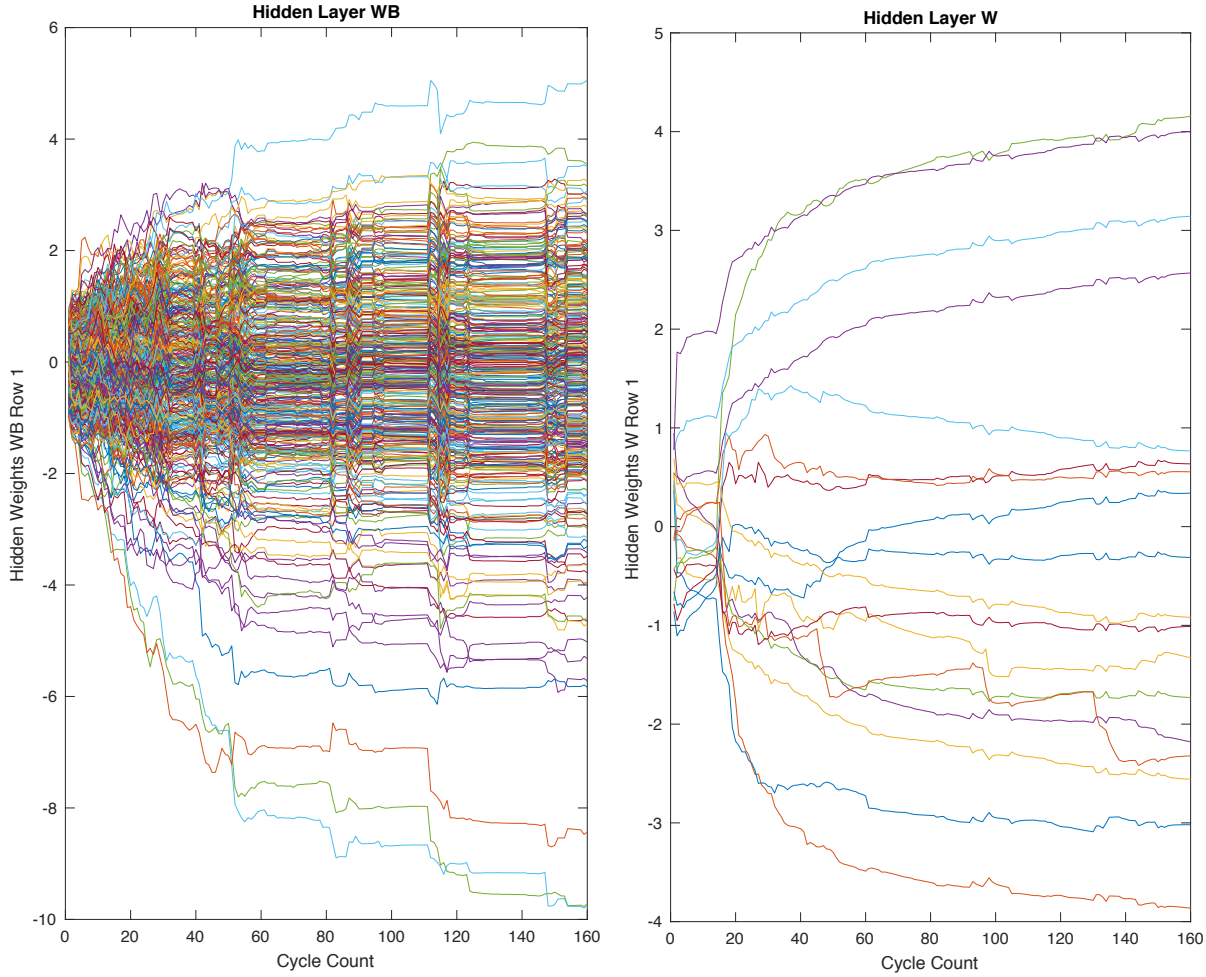


Figure 15 - Neural Network Weights

Testing Implementation

A test function was developed to allow images to be classified by the neural network. This comprises both the Image Processing and Neural Network functions developed earlier, with slight modification to allow image selection and classification output. This does not include error checking, as this is intended to be implemented for automated classification after the network has been suitably validated. Errors were still present during the image processing, which would sometimes result in misidentification of the relevant regions of images.

Where the image processing results in correct identification of character regions however, the neural network is quite successful at classification, though errors are still present, as seen in the bottom row of the sample test results below.

Figure 16 – Sample Test Results. The top image is the detected region, the middle image is the final cropped individual characters, and the bottom box is the classification by the neural network. The upper set is 100% correct, while the bottom row shows errors.

Discussion

From the errors determined above for 17 nodes and learning constant 0.2, we obtain a training RMS error of 0.004, and validation error of 0.0016. The example test results exhibit a 4 failures out of a total 48 characters, and while these images were selected to demonstrate both success and failure of the system, they are quite typical of test cases where the image processing has resulted in clear images. Clearly there is still some error within the system, however the results are largely successful, and the neural network is quite clearly capable of number plate identification.

Complete failure of the neural network only occurs when number plates have not been correctly detected and processed from the original image. It may be possible to improve this by implementing neural networks in the processing of the number plate images. If neural processing was used as an initial pass to help identify relevant regions, this would provide more reliable data to pass through to the neural classifier.

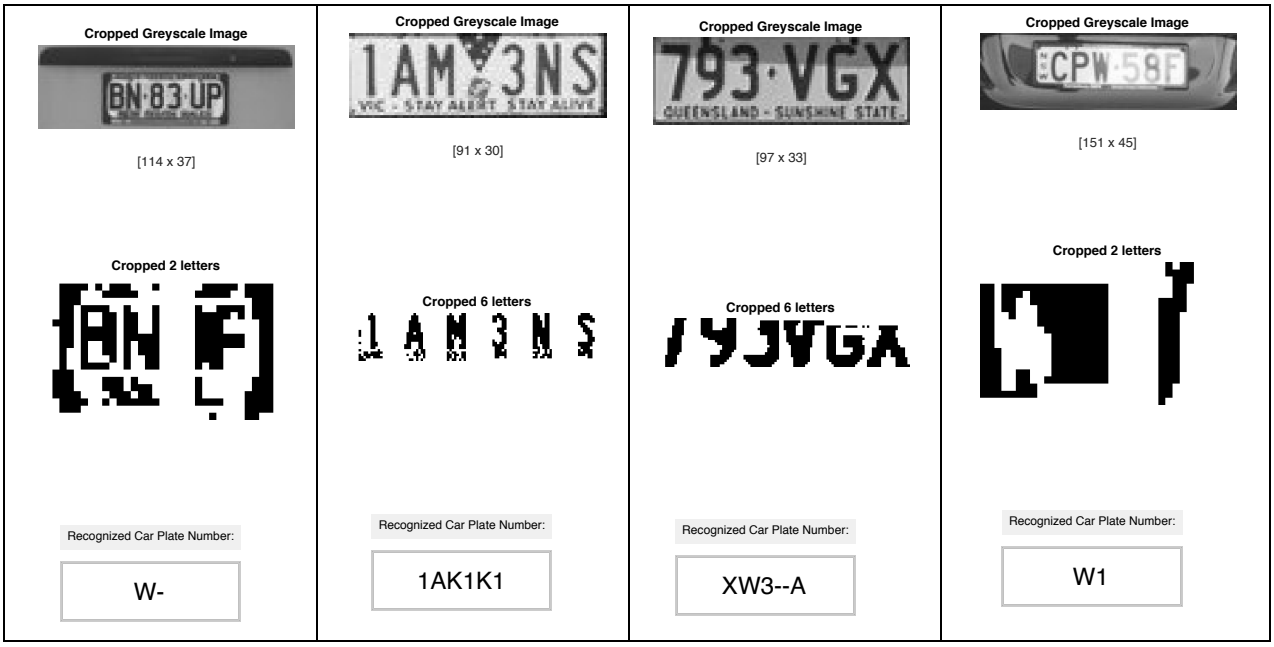


Figure 17 – Failed image processing examples.

Future Work

Classification by the neural network has been largely observed to be successful. Inaccuracies are present however, and further work could be conducted to improve this. The use of more training data, particularly for characters that are currently underrepresented, would help to improve the reliability here. Further developments could be made to the neural network itself, including the use of additional hidden layers, and further refinement of the number of hidden nodes and learning constant. A more developed system could also implement automatic early stopping during validation.

Conclusion

The purpose of this project was to investigate and demonstrate the feasibility of number plate identification using neural networks, and the results have demonstrated that this system is not only feasible, but quite successful. While errors are present in the developed system, it is possible to further develop this in future to create a more accurate and realiable system. Using a larger data set would be useful, and improving the processing of images to create this data set would be essential. This would not only improve the rate of data collection, but make the entire system far more reliable during test implementation, as the majority of errors were caused by limitations in the image processing stage rather than the neural classification.

References

Laxmi, Vijay, and Harish Rohil. "License Plate Recognition System using BackPropagation Neural Network." *International Journal of Computer Applications* 99.8 (2014): 29-37.

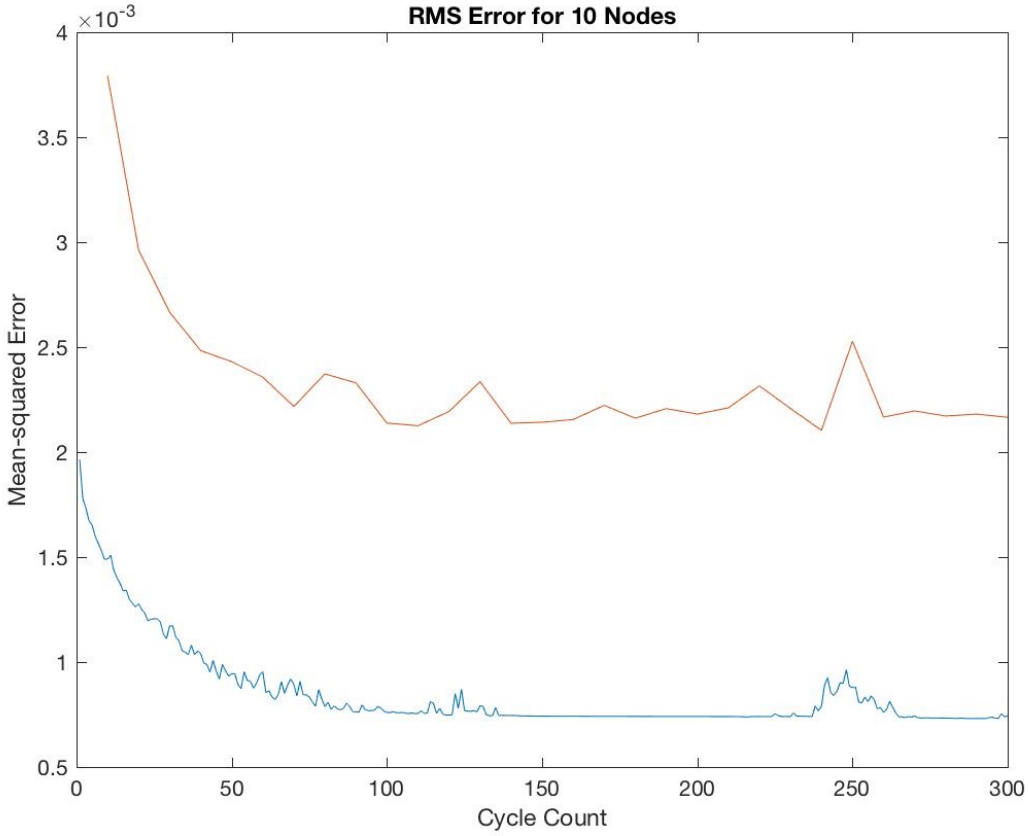
Akoum, A, Bassam Daya, and Pierre Chauvet. "TWO NEURAL NETWORKS FOR LICENSE NUMBER PLATES RECOGNITION." *Journal of Theoretical & Applied Information Technology* 12 (2010).
Kocer, H Erdinc, and K Kursat Cevik. "Artificial neural networks based vehicle license plate recognition." *Procedia Computer Science* 3 (2011): 1033-1037.

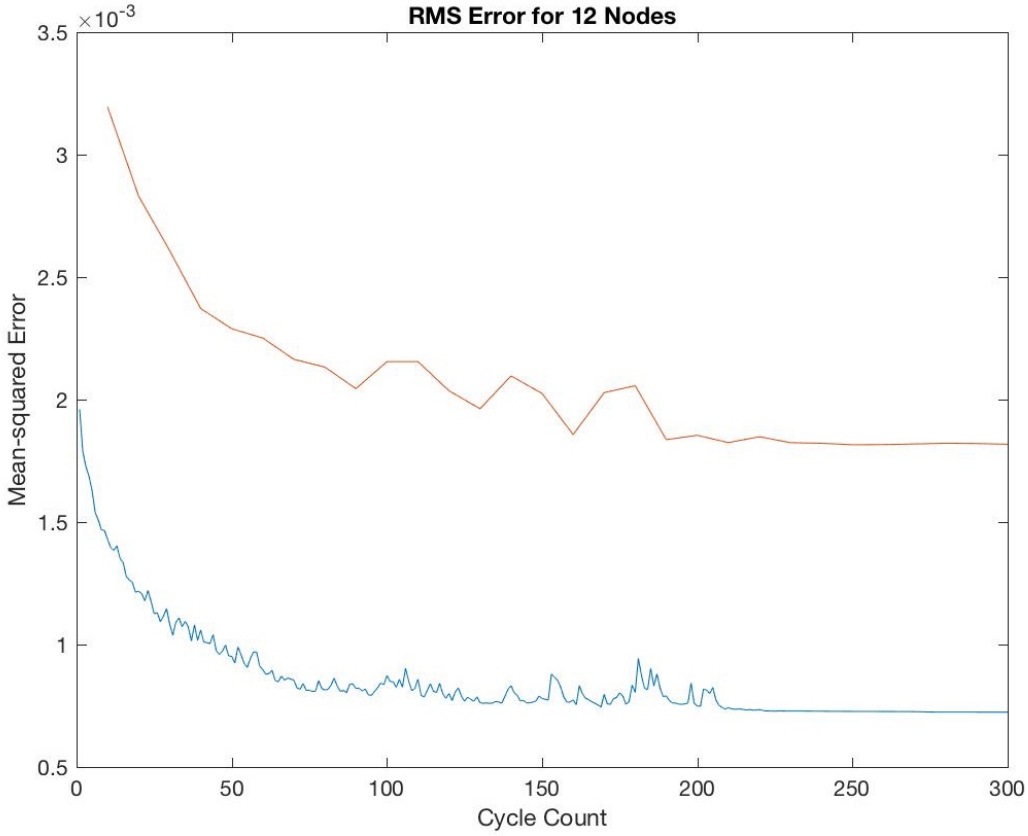
Ventzas, D, and D Karras. "Vehicle's License Plate Recognition System based on a Neural Network Radon Transform Method." *International virtual conference, Advanced Research in Scientific Areas* Oct. 2012: 2097-2104.

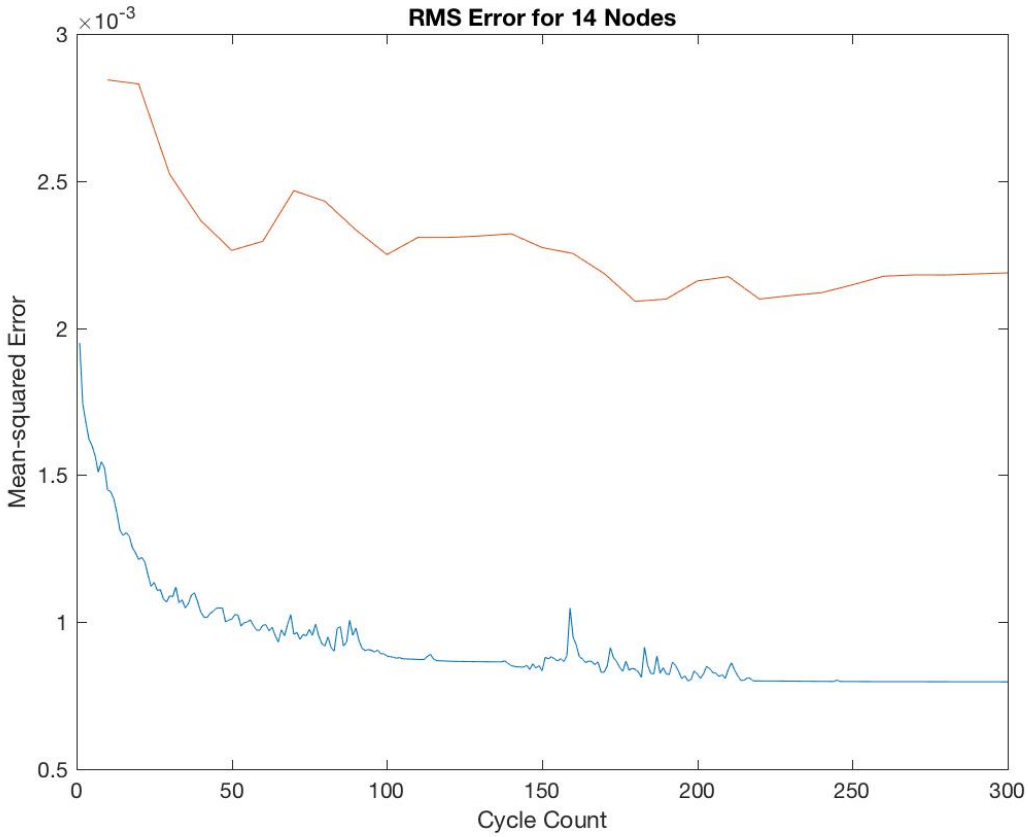
Bhushan, B, Singh, S and Ruchi Singla. "License Plate Recognition System using Neural Networks and Multithresholding Technique." *International Journal of Computer Applications* 84.5 (2013).

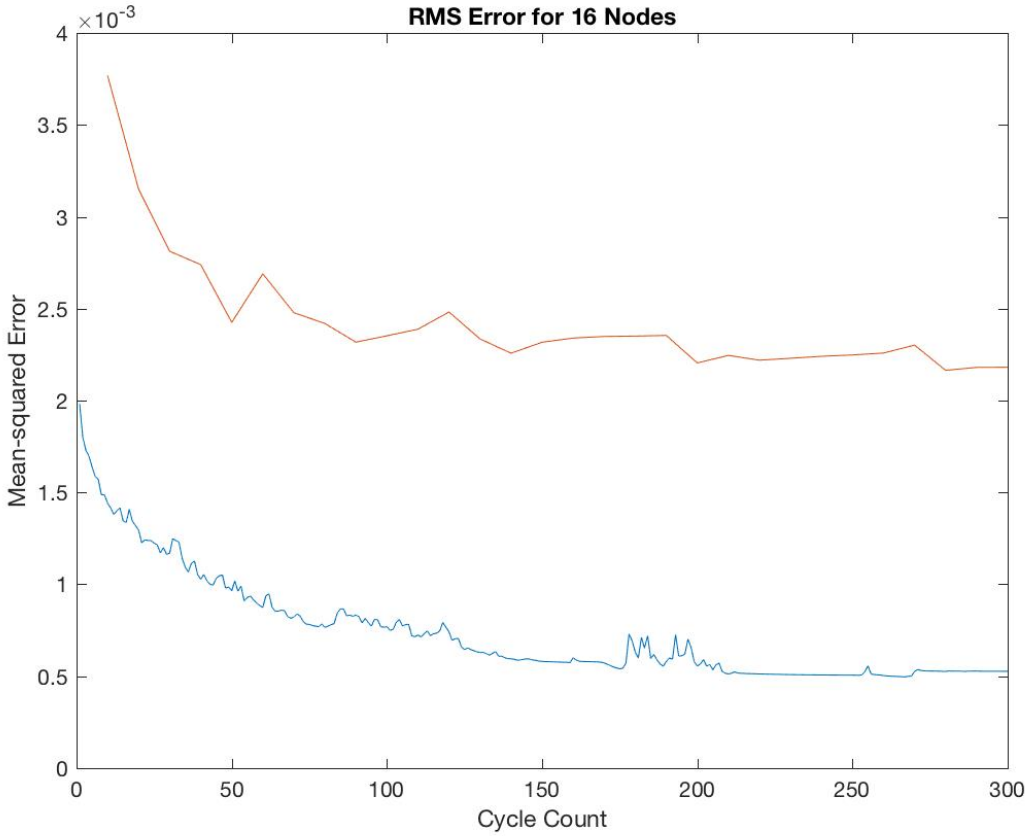
Appendix A - Error for number of hidden nodes

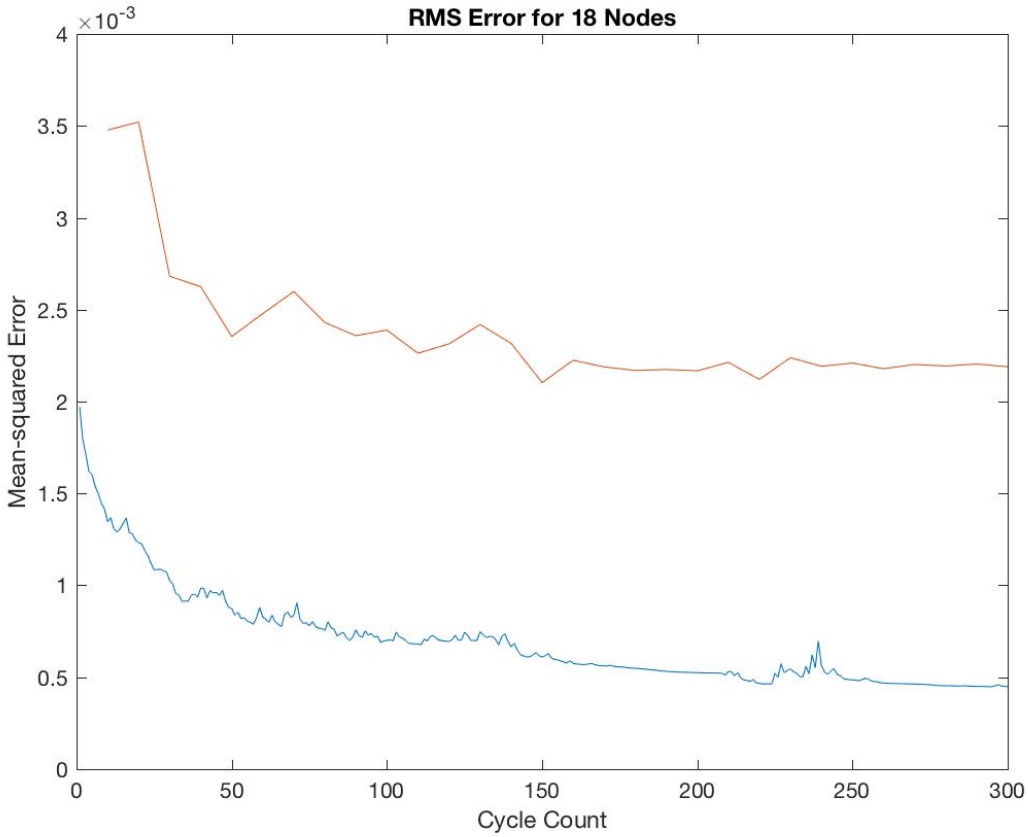
The following images are the error results obtained for varying numbers of hidden nodes. In all cases, the learning constant was n=0.2

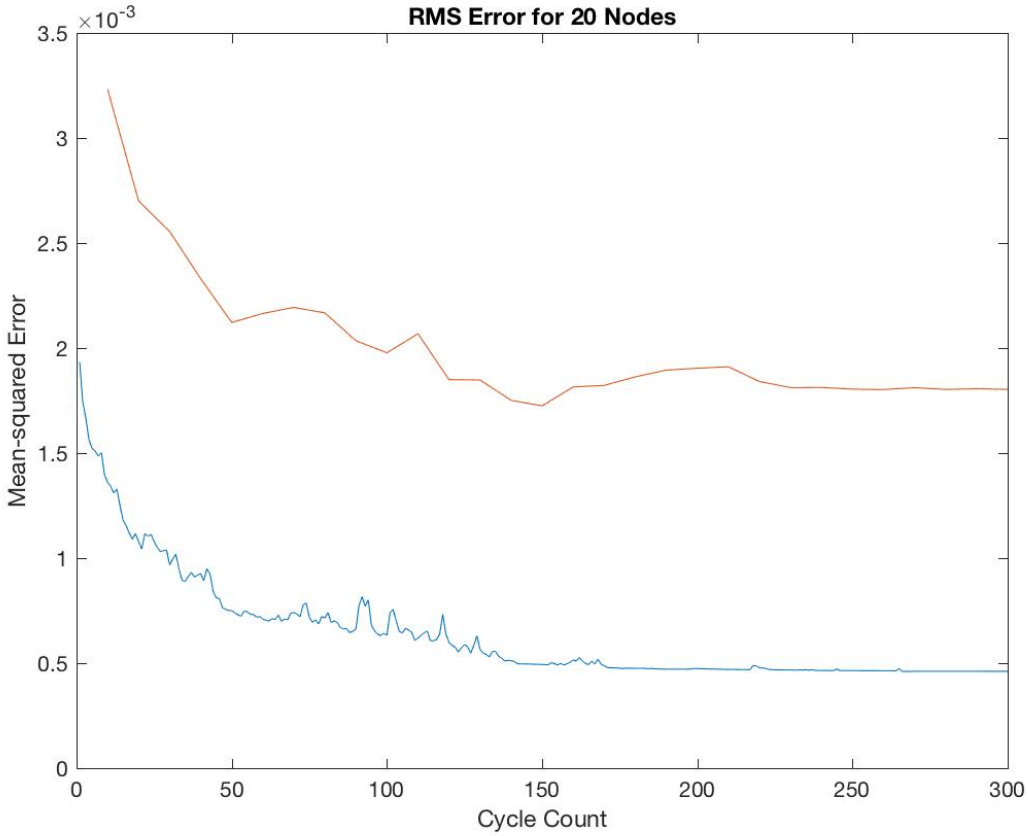


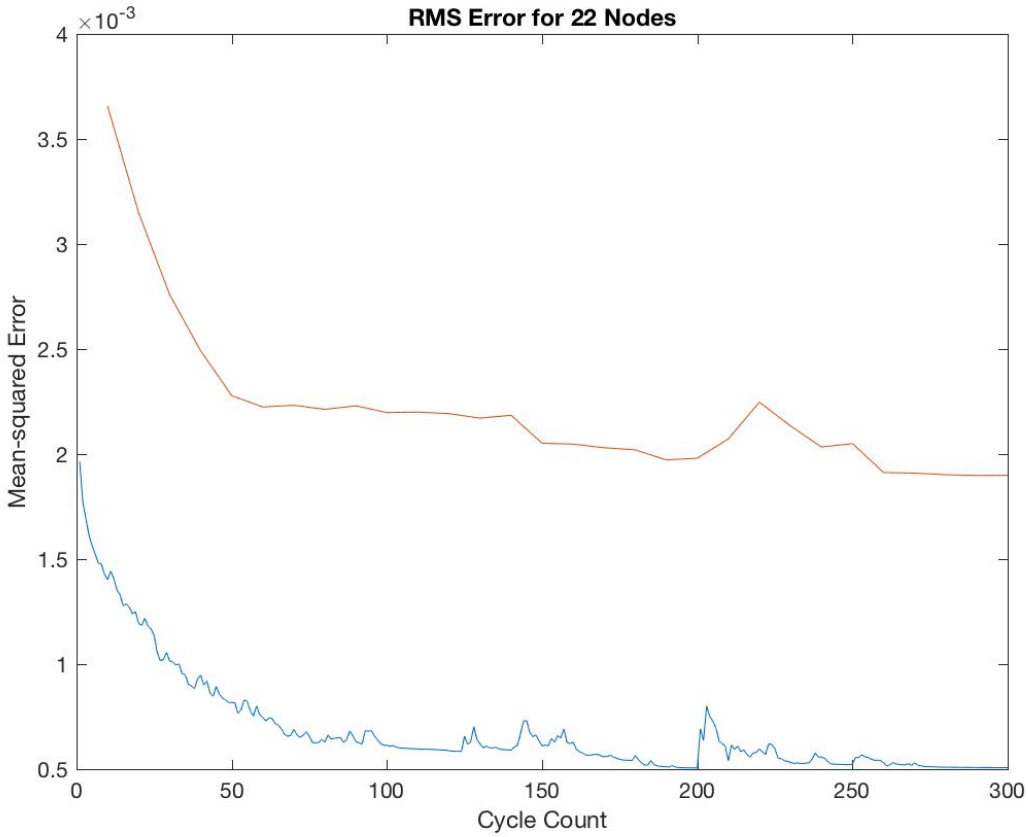


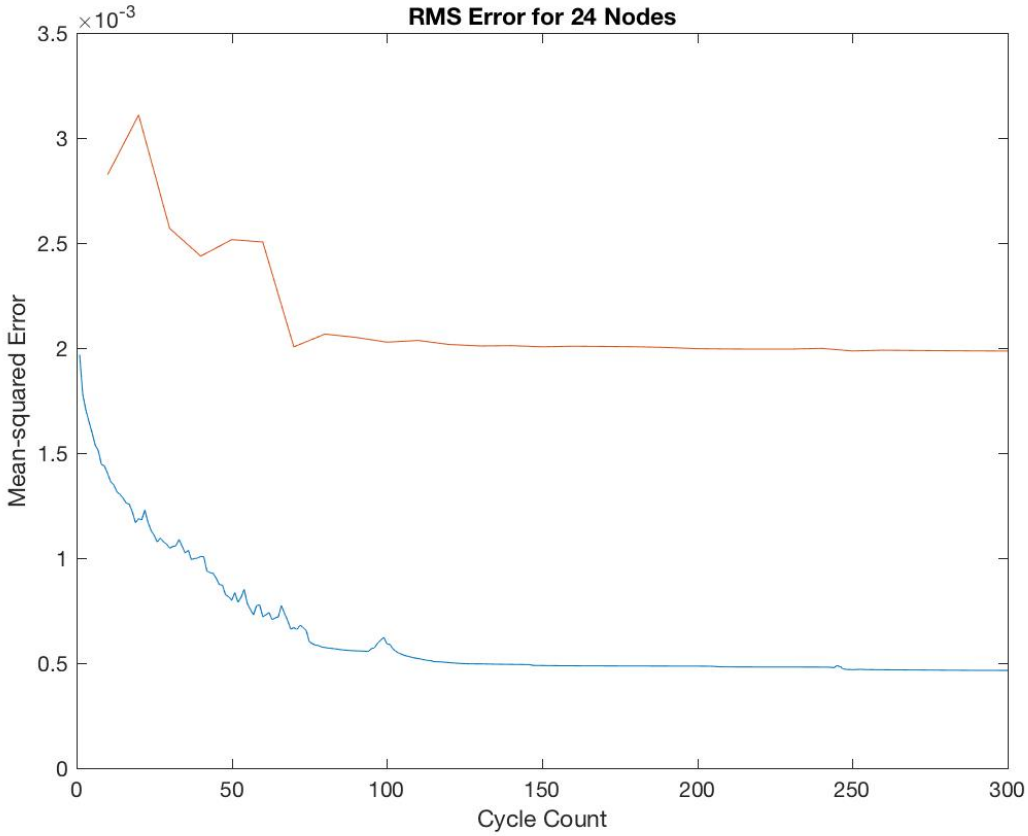

RMS Error for 14 Nodes
×10^-3
Mean-squared Error
Cycle Count
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250
300

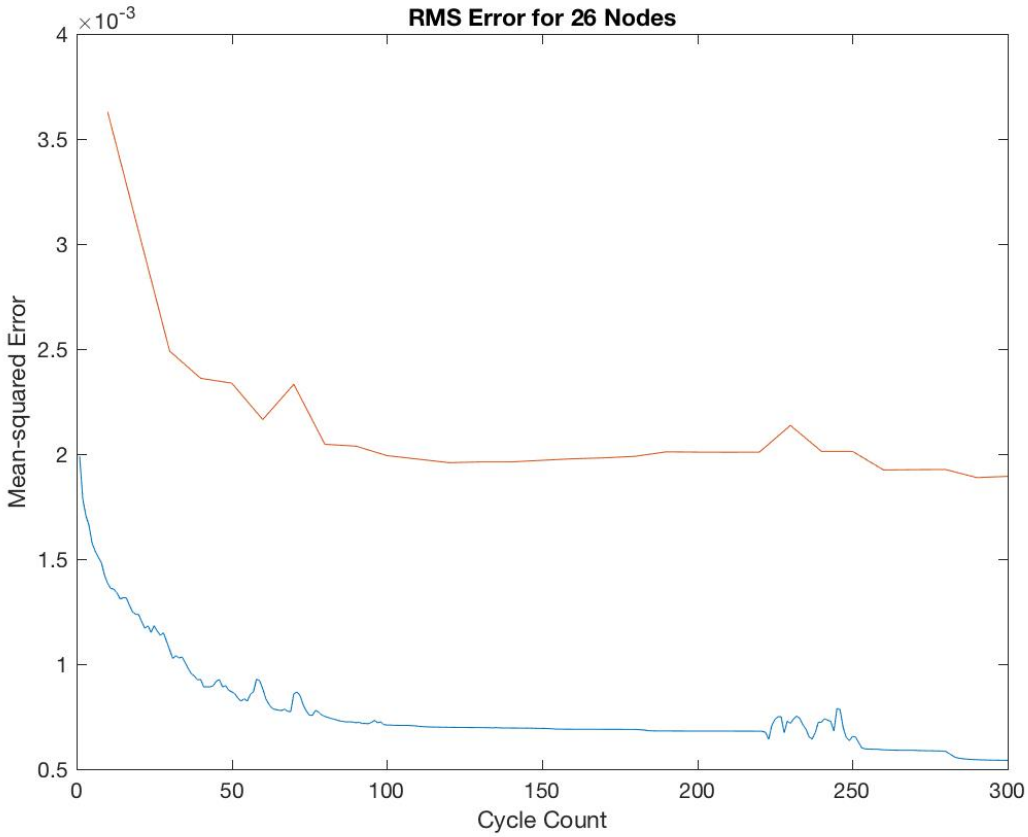

RMS Error for 16 Nodes
×10^-3
Mean-squared Error
Cycle Count
4
3.5
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250
300

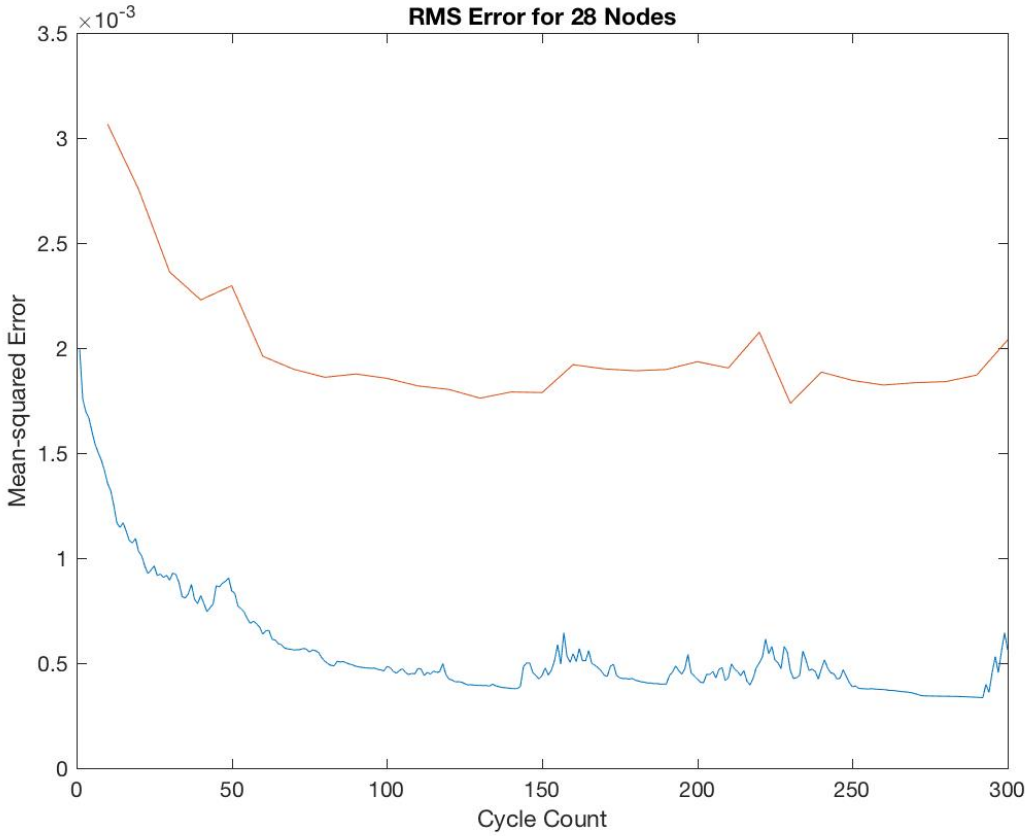

RMS Error for 18 Nodes
×10^-3
Mean-squared Error
Cycle Count
0
0.5
1
1.5
2
2.5
3
3.5
4
0
50
100
150
200
250
300

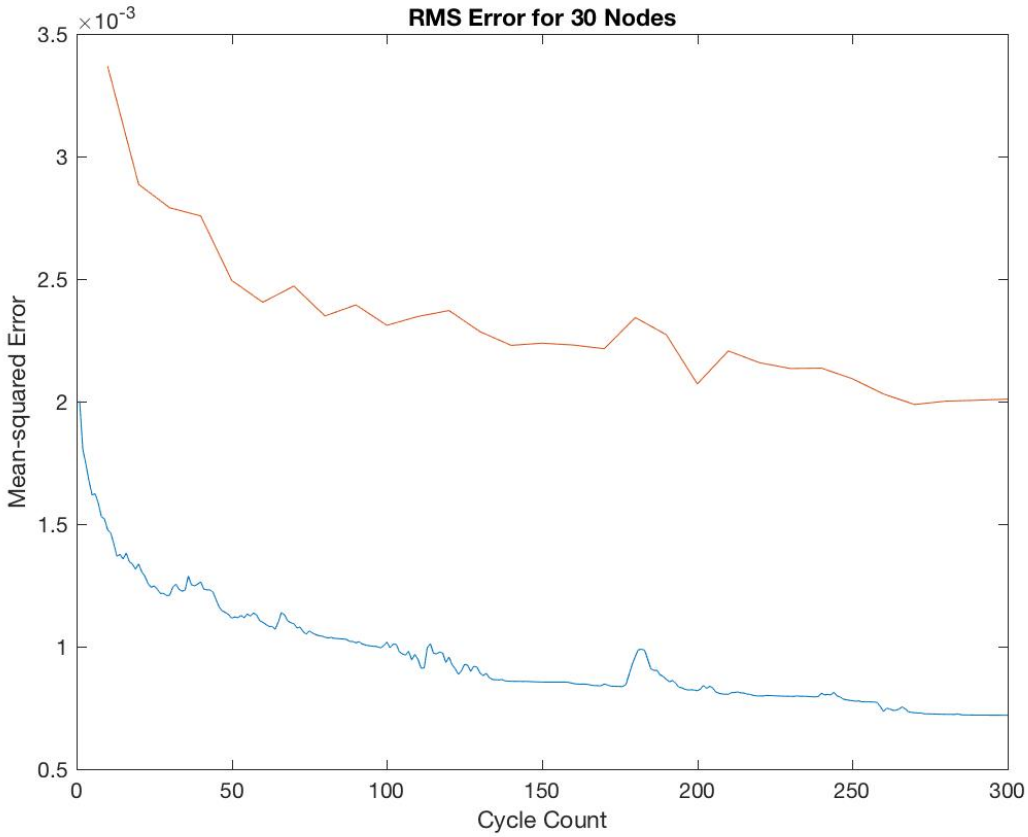

RMS Error for 20 Nodes
×10^-3
Mean-squared Error
Cycle Count
0
0.5
1
1.5
2
2.5
3
3.5
0
50
100
150
200
250
300


RMS Error for 22 Nodes
×10^-3
Mean-squared Error
Cycle Count
4
3.5
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250
300


RMS Error for 24 Nodes
×10^-3
Mean-squared Error
Cycle Count
3.5
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250
300


RMS Error for 26 Nodes
×10^-3
Mean-squared Error
Cycle Count
4
3.5
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250
300


RMS Error for 28 Nodes
×10^-3
Mean-squared Error
Cycle Count
3.5
3
2.5
2
1.5
1
0.5
0
50
100
150
200
250
300


RMS Error for 30 Nodes
×10^-3
3.5
3
2.5
2
1.5
1
0.5
Mean-squared Error
0
50
100
150
200
250
300
Cycle Count

Appendix B – Neural Network Training MATLAB code

```
clear all;
close all;

%set up inputs with augmentation

letters=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q'
'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z' '1' '2' '3' '4' '5' '6' '7' '8' '9' '0'];

testchars=36; %the number of characters we will try to identify from A to 0
(can change this in letters)

for countl=1:testchars;
    rawdata{countl}=csvread(['Training_CSV/Letter_' letters(countl) '.csv']);
    rawdata{countl}=rawdata{countl}';
    rawdata{countl}=rawdata{countl}.*2;
    rawdata{countl}=rawdata{countl}-1;
end

countm=1;    %start of traindata (eg if this is 5, the first 4 of each letter
can be validation)
countp=0;
while countm <= max(cellfun('size', rawdata, 2))
    for countl=1:testchars;
        if size(rawdata{countl},2)>=countm;
            countp=countp+1;
            traindata = rawdata{countl}(:,countm);
            x(:,countp)=[traindata; -1];
            d(:,countp) = zeros(testchars,1)-1;
            d(countl,countp) = 1;
        end
    end
    countm=countm+1;
end

fprintf('Total %d training set ready.\n',countp);

%set up learning conditions
n=0.2;
cycles=160;


samples=size(x,2);

disp('Starting training...');


for nodes=17;
wStr=['InitialW_' num2str(nodes)  'Nodes.csv'] %Weights created from 10 to 30
w=csvread(wStr);
wbStr=['InitialWB_' num2str(nodes)  'Nodes.csv']
wb=csvread(wbStr);


errcur=[];
countk=0;
errcur_sum=[];
valx=[];
```

```
valy=[];
Erms=[];

for counti=1:cycles;
    fprintf('Training cycle %d\n',counti);
    cy_err=zeros(36,1);
    cy_err_sum=0;

    savew(counti,:)= w(1,:);
    savewb(counti,:)= wb(1,:);


    for countj=1:samples;

        countk=countk+1;

        vb= wb*x(:,countj);                %activation vector

        y=((1-exp(-vb))./(1+exp(-vb))); %output to hidden layer
        yaug=[y;-1];                       %augment hidden layer

        v=w*yaug;                          %activation vector for next pass
        z=((1-exp(-v))./(1+exp(-v)));    %output z layer

        dzdv=0.5*(1-z.^2);
        delta=(d(:,countj)-z).*(dzdv);          %find delta

        dydvb=0.5*(1-y.^2);
        gh=w;

        for u=1:length(y)
            deltab(u,1)=delta(:,1)'*gh(:,u)*0.5*(1-y(u,1)^2);
        end

        w=w+n*delta*yaug';                    %update w vector
        wb=wb+n*deltab*(x(:,countj))';        %update wbar

        err=0.5*(d(:,countj)-z).^2;
        cy_err=cy_err+err;                    %find cycle error curve
        cy_err_sum=cy_err_sum+sum(err);

    end
    errcur=[errcur cy_err];
    errcur_sum=[errcur_sum cy_err_sum];
    Erms=[Erms sqrt(2*sum(cy_err_sum))./(899*36)];

    if mod(counti,10) == 0
        csvwrite('Matrix_W.csv',w);
        csvwrite('Matrix_WB.csv',wb);
        validation_rate=Validation();        %Perform Validation test
        valx=[valx counti];
        valy=[valy validation_rate];
    end
end

disp('Training completed.');


figure(1);                                  %Plot error for each character
```

```
for i=1:36
    subplot(6,6,i);
    plot(errcur(i,:));
    title(sprintf('Letter %c',letters(i)));
    xlabel('Cycle Count');
    ylabel('Cycle Error');
end

H=figure(2);
plot(Erms);                                %Plot error for network

titStr=['RMS Error for ' num2str(nodes) ' Nodes']
title(titStr)

xlabel('Cycle Count');
ylabel('Mean-squared Error');
hold on;
plot(valx,valy);
hold off;

figStr=['Fig ' num2str(nodes)  ' Nodes.jpg']    %Save error figures
saveas(H,figStr)
outwStr=['FinalMatrix_W_' num2str(nodes)  'Nodes.csv']  %Save final weights
csvwrite(outwStr,w);
outwbStr=['FinalMatrix_WB_' num2str(nodes)  'Nodes.csv']
csvwrite(outwbStr,wb);

end
```

Appendix B – Neural Network Validation MATLAB code

```
function ret=Validation()
    letters=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P'
'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z' '1' '2' '3' '4' '5' '6' '7' '8' '9'
'0'];
    testchars=36; %the number of characters we will try to identify from A to
0 (can change this in letters)

    for countl=1:testchars;
        rawdata{countl}=csvread(['Validation_CSV/Letter_' letters(countl)
'.csv']);
        rawdata{countl}=rawdata{countl}';
        rawdata{countl}=rawdata{countl}.*2;
        rawdata{countl}=rawdata{countl}-1;
    end

    countm=1;    %start of traindata (eg if this is 5, the first 4 of each
letter can be validation)
    countp=0;
    while countm <= max(cellfun('size', rawdata, 2))
        for countl=1:testchars;
            if size(rawdata{countl},2)>=countm;
                countp=countp+1;
                traindata = rawdata{countl}(:,countm);
                x(:,countp)=[traindata; -1];
                d(:,countp) = zeros(testchars,1)-1;
                d(countl,countp) = 1;
            end
        end
        countm=countm+1;
    end

    fprintf('Total %d validation set ready.\nStarting validation...',countp);

    Ec=0;
    diff=0;

    % Bipolar activation function
    fz=@(v) (1-exp(-v))./(1+exp(-v));
        % Read w and wb matrix from csv files
    w=csvread('Matrix_W.csv');
    wb=csvread('Matrix_WB.csv');

    for countv=1:countp
        vb=wb*x(:,countv);
        y=fz(vb);   %hidden layer
        yaug=[y;-1];
        v=w*yaug;
        z=fz(v);    %output layer
        E=sum(0.5*(d(:,countv)-z).^2);
        Ec=Ec+E;
    end

    Erms=sqrt(2*Ec)./(214*36);

    ret = Erms;
    fprintf(' Finished.\n');%Matched set :
%d(%.2f%%)\n',match,100*match/countp);
end
```

Appendix C – Image Processing and Manual Classification MATLAB code

```
% Making CSVs from car images
clear all;
close all;

% Browse for the image file.
[baseFileName, folder] = uigetfile({'*.jpg','JPG Files (*.jpg)'},...
    'Specify an image file (Multiple files allowed)','MultiSelect','on');
fullImageFileName = fullfile(folder, baseFileName);

if isnumeric(baseFileName);
    return;
elseif ischar(baseFileName);
    baseFileName=cellstr(baseFileName);
end

for idx=baseFileName;

    imStr=char(idx);

    oldFolder=cd(folder);
    colorImage = imread(imStr);
    cd(oldFolder);


    [imHeight, imWidth, dim]=size(colorImage);
    fprintf('Filename: %s (Height = %d, Width = %d)\n', imStr, imHeight,
imWidth);

    I = rgb2gray(colorImage);

    % Detect MSER regions.
    ROIRegion = [int16(imWidth/3), int16(imHeight*3/8), int16(imWidth/3),
int16(imHeight/2)];
    [mserRegions, mserConnComp] = detectMSERFeatures(I,...
                                'RegionAreaRange', [2000 11000],...
                                'MaxAreaVariation', 0.25,...
                                'ROI', ROIRegion);

    % Use regionprops to measure MSER properties
    mserStats = regionprops(mserConnComp, 'BoundingBox', 'Eccentricity', ...
        'Solidity', 'Extent', 'Euler', 'Image');

    % Compute the aspect ratio using bounding box data.
    bbox = vertcat(mserStats.BoundingBox);
    w = bbox(:,3);
    h = bbox(:,4);
    aspectRatio = w./h;

    % Threshold the data to determine which regions to remove. These
thresholds
    % may need to be tuned for other images.
    filterIdx = aspectRatio' > 6.1;
    filterIdx = filterIdx | aspectRatio' < 2.3;
    %filterIdx = filterIdx | [mserStats.Eccentricity] > .995 ;
    filterIdx = filterIdx | [mserStats.Solidity] < 0.46;
    filterIdx = filterIdx | [mserStats.Extent] < 0.43 | [mserStats.Extent] >
0.9;
```

```
    filterIdx = filterIdx | [mserStats.EulerNumber] > -2 |
[mserStats.EulerNumber] < -100;

    % Remove too big boxes if height more than 50px
    filterIdx = filterIdx | (bbox(:,4) > 90)';
    % Remove too big boxes if width more than 26% of width
    filterIdx = filterIdx | (bbox(:,3) > (imWidth*0.26))';
    % Remove if centre of Y coordinates upper than half
    filterIdx = filterIdx | ((bbox(:,2)+(bbox(:,4)/2)) < (imHeight / 2))';
    % Remove if centre of X coordinates too far from centre (7.5%)
    filterIdx = filterIdx | ((abs((bbox(:,1)+(bbox(:,3)/2))-(imWidth / 2))) >
(imWidth*0.075))';

    % Remove regions
    mserStats(filterIdx) = [];
    mserRegions(filterIdx) = [];
    %mserStats.Solidity

    if size(mserStats, 1) > 0;
        % Pick only One which has maximum extent value.
        max_ext_idx=1;
        max_ext_value=0;
        for z=1:size(mserStats, 1)
            if mserStats(z).Extent > max_ext_value
                max_ext_idx=z;
                max_ext_value=mserStats(z).Extent;
            end
        end
        selected_box=mserStats(max_ext_idx).BoundingBox;
        fprintf('Selected Box = %d %d %d %d\n',int16(selected_box));

        % Get bounding boxes for all the regions
        bboxes = vertcat(mserStats.BoundingBox);

        % Show the expanded bounding boxes
        IExpandedBBoxes =
insertShape(colorImage,'Rectangle',bboxes,'LineWidth',3);
        FinalImg =
insertShape(IExpandedBBoxes,'Rectangle',selected_box,'LineWidth',3,'Color','r
ed');

        % Show remaining regions
        subplot(3,3,1);
        imshow(FinalImg);
        title(sprintf('MSER Region of %s', imStr));
        subplot(3,3,2);
        cropImg=imcrop(I,selected_box);
        imshow(cropImg);
        title('Cropped Greyscale Image');
        xlabel(sprintf('[%d x %d]',size(cropImg,2),size(cropImg,1)));
        subplot(3,3,3);
        imhist(cropImg);
        title('Histogram of cropped image');

        % Calculate median value in grey image
        [counts, binLists]=imhist(cropImg);
        max1_bin=0;
        max1_count=0;
        for k=1:256;
            if counts(k) > max1_count;
                max1_count=counts(k);
```

```
                max1_bin=k;
            end
        end
        max2_bin=0;
        max2_count=0;
        for k=256:-1:1;
            if (counts(k) > max2_count) &&...
                    (counts(k) < max1_count) &&...
                    (abs(k-max1_bin) > 40);
                max2_count=counts(k);
                max2_bin=k;
            end
        end
        if max1_bin > max2_bin;
            tempswap=max2_bin;
            max2_bin=max1_bin;
            max1_bin=tempswap;
        end
        threshold=((max1_bin+max2_bin)/2);
        %fprintf('Left Max = %d, Right Max = %d, Threshold = %d(%.2f)\n',
max1_bin, max2_bin, int16(threshold), threshold/256);
        hold on;
        ylim=get(gca,'ylim');
        line([threshold threshold],ylim,'Color','r');
        hold off;
        xlabel(sprintf('Threshold level=%d',int16(threshold)));

        subplot(3,3,4);
        bwCropImg=im2bw(cropImg,threshold/256);
        imshow(bwCropImg);
        title('Binary converted Image');

        % Inverse Image if black is majority
        if (sum(sum(bwCropImg))/prod(size(bwCropImg))) < 0.375;
            bwCropImg=not(bwCropImg);
            %disp('Inverted');
        end

        % Strip Image vertically (first and last 3%)
        xmax=size(bwCropImg,2);
        ymax=size(bwCropImg,1);
        limit_strip=int16(xmax*0.03);
        limit_black=int16(ymax*0.25);       % 75% (1-0.75)
        limit_white=int16(ymax*0.90);       % 90%
        vertical_sum=sum(bwCropImg,1);
        strip_flag=(vertical_sum < limit_black) | (vertical_sum >
limit_white);
        for i=limit_strip:xmax-limit_strip;
            strip_flag(i)=0;
        end

        left_strip=0;
        right_strip=0;
        for i=1:limit_strip
            if i > 3 && strip_flag(i) == 1 &&...
                    strip_flag(i-1) == 0 &&...
                    strip_flag(i-2) == 0 &&...
                    strip_flag(i-3) == 0;
                strip_flag(i) = 0;
            end
            if strip_flag(i) == 1;
                left_strip = i;
```

```
            end
            if i > 3 && strip_flag(xmax-i+1) == 1 &&...
                    strip_flag(xmax-i+2) == 0 &&...
                    strip_flag(xmax-i+3) == 0 &&...
                    strip_flag(xmax-i+4) == 0;
                strip_flag(xmax-i+1) = 0;
            end
            if strip_flag(xmax-i+1) == 1;
                right_strip = i;
            end
        end

        for i=1:left_strip
            strip_flag(i) = 1;
        end
        for i=1:right_strip
            strip_flag(xmax-i+1) = 1;
        end

        bwCropImg(:,strip_flag) = [];

        % Strip Image horizontally
        horizontal_sum=sum(bwCropImg,2);
        centre_y=int16(size(bwCropImg,1)/2);
        adj = int16(centre_y/4);
        max_CropImg = max(horizontal_sum);
        threshold = max(horizontal_sum(centre_y-adj:centre_y+adj));
        adjust_threshold = int16((max_CropImg-threshold)/3);
        strip_flag=boolean(zeros(ymax,1));
        upper_threshold=threshold+adjust_threshold;
        lower_threshold=int16(threshold/3);
        for i=centre_y-3:-1:1;
            if or(horizontal_sum(i) > upper_threshold, horizontal_sum(i) <
lower_threshold);
                break;
            end
        end
        for j=1:i
            strip_flag(j)=1;
        end
        for i=centre_y+3:ymax;
            if or(horizontal_sum(i) > upper_threshold, horizontal_sum(i) <
lower_threshold);
                break;
            end
        end
        for j=i:ymax
            strip_flag(j)=1;
        end
        bwCropImg(strip_flag,:)=[];

        subplot(3,3,5);
        imshow(bwCropImg);
        title('Stripped Image');
        xlabel(sprintf('[%d x %d]',size(bwCropImg,2),size(bwCropImg,1)));

        % Resize Image
        AdjustedWidth=int16(20*size(bwCropImg,2)/size(bwCropImg,1));
        FinalImage=imresize(bwCropImg,[20 AdjustedWidth]);

        % Clear useless solid line on both top and bottom 3 lines
        xmax=size(FinalImage,2);
```

```
ymax=size(FinalImage,1);
for i=[1 2 3 18 19 20];
    FinalImage(i,:)=MyFunc2(FinalImage(i,:));
end

subplot(3,3,6);
imshow(FinalImage);
title('Fixed Height Image');
xlabel(sprintf('[%d x %d]',AdjustedWidth,20));

subplot(3,3,9);
inverted_FinalImage=not(FinalImage);
horizontal_sum=sum(inverted_FinalImage,1);
bar(horizontal_sum);
title('Horizontal Histogram of stripped image');
xlim([1 xmax]);

% Pre-adjustment
temp_strip=0;
original_horizontal_sum=horizontal_sum;
while MyFunc1(horizontal_sum) > 18;
    temp_strip=temp_strip+1;
    horizontal_sum=horizontal_sum-1;
end

cropped_letters=[];
start_part=false;
extend_right=false;
hold on;
plot(xlim,[temp_strip temp_strip]);
for i=1:xmax;
    if horizontal_sum(i) > 0
        if ~start_part;
            start_part=true;
            starting_point=i;
            plot([i i],[1 20],'r');
            extend_right=false;
        end
    else
        if start_part;
            if (original_horizontal_sum(i) > 0) &&...
                    (((i-starting_point) < 6) || extend_right);
                if (i-starting_point) < 18;
                    extend_right=true;
                    continue;
                else
                    extend_right=false;
                end
            end
            start_part=false;
            plot([i-1 i-1],[1 20],'b');

            letter_img=MyFunc3(FinalImage,starting_point,i-1);

            if sum(sum(letter_img)) > 0;
                cropped_letters=[cropped_letters letter_img];
            end
        end
    end
end
if start_part;
    plot([i i],[1 20],'b');
```

```
            letter_img=MyFunc3(FinalImage,starting_point,i);

            if sum(sum(letter_img)) > 0;
                cropped_letters=[cropped_letters letter_img];
            end
        end
        hold off;
        figs=figure(1);
        figs.OuterPosition=[10 50 1000 800];

        subplot(3,3,7);
        imshow(cropped_letters);
        sizeofletters=int8(size(cropped_letters,2)/18);
        title(sprintf('Cropped %d letters',sizeofletters));
        cropped_letters=not(cropped_letters);

        ButtonFlag=false;
        h_Text=uicontrol('Style','text','String','Enter characters of
left:',...
            'Position',[430 210 150 15],'FontSize',10);
        h_Edit=uicontrol('Style','edit','Position',[430 150 150 50],...
            'FontSize',20,'Callback','ButtonFlag=true;');
        h_Button=uicontrol('Style','pushbutton','String','Save to CSV',...
            'Position',[450 100 110 30],'FontSize',10,...
            'Callback','ButtonFlag=true;');

        plate_string=get(h_Edit,'String');
        while not(waitforbuttonpress);
            if ButtonFlag;
                plate_string=get(h_Edit,'String');
                if isempty(plate_string);
                    disp('Input letters on left image.');
                    ButtonFlag=false;
                else
                    if sizeofletters == size(plate_string,2);
                        close(1);
                        break;
                    else
                        disp('Wrong letter counts');
                        ButtonFlag=false;
                    end
                end
            end
        end

        if not(isempty(plate_string)) && (sizeofletters ==
size(plate_string,2));
            plate_string=upper(plate_string);
            fprintf('Input: %s\n',plate_string);

            for i=1:sizeofletters;
                filename=sprintf('Letter_%c.csv',plate_string(i));
                fprintf('Saving file %s...\n',filename);
                fID=fopen(filename,'at+');

                for j=1:20;
                    for k=1:18;
                        fprintf(fID,'%d',cropped_letters(j,(k+(i-1)*18)));
                        if (k<18) || (j<20);
                            fprintf(fID,',');
```

```
                        end
                    end
                end
                fprintf(fID,'\n');
                fclose(fID);
            end
        end
        close all;
    end;
end;
```

Appendix D – Test Image Processing and Classification MATLAB code

Note: This is called as part of another test code, which is very similar to the Image Processing code, and for that reason has not been included here.

```
% Checkletter
function ret=CheckLetter2(input_matrix)
    letters=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P'
'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z' '1' '2' '3' '4' '5' '6' '7' '8' '9'
'0'];

    % Bipolar activation function
    fz=@(v) (1-exp(-v))./(1+exp(-v));

    % Read w and wb matrix from csv files
    w=csvread('Matrix_W.csv');
    wb=csvread('Matrix_WB.csv');

    % Make it bipolar
    input_matrix=input_matrix.*2;
    input_matrix=input_matrix-1;

    % Make inputed matrix as 1 column
    for i=1:20
        for j=1:18
            input((i-1)*18+j,1)=input_matrix(i,j);
        end
    end

    AugmentedInput=[input ; -1];
    vb=wb*AugmentedInput;
    y=fz(vb);   %hidden layer
    yaug=[y;-1];
    v=w*yaug;
    z=fz(v);    %output layer
    positive_count=0;
    for i=1:36
        if z(i) > 0
            positive_count=positive_count+1;
            %fprintf('Positive value : %.4f on %c\n',z(i),letters(i));
            candidates(positive_count)=i;
        end
    end
    if positive_count == 0
        ret=0;
    elseif positive_count < 2
        ret=letters(candidates(1));
    else
        max_value=0;
        candidate_number=1;
        for i=1:positive_count
            if z(candidates(i)) > max_value
                max_value=z(candidates(i));
                candidate_number=candidates(i);
            end
        end
        ret=letters(candidate_number);
    end
end
```