Automated Image Interpretation via ConvNets & Recurrent Networks

Xiongming Dai

SysEng 5212/EE 5370

May 4, 2018

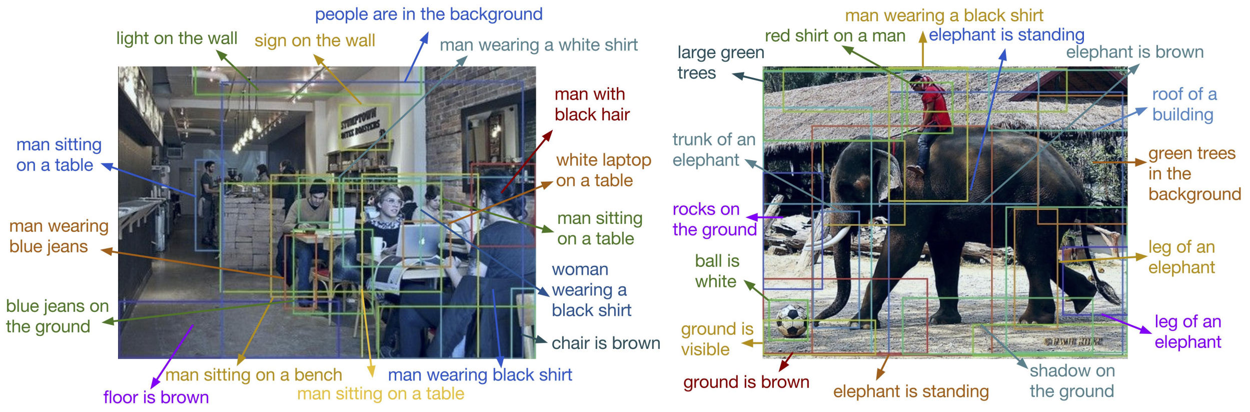


Table of Contents

1. Introduction

Neural networks are employed in a number of "real-world" applications, including pattern recognition, similar to the techniques used in the SysE378 class. I chose to examine the application of neural networks perform license plate recognition.

There are several commercially available license plate recognition systems available today [1] [2]. Figure 1 shows a typical law enforcement application. These systems include fixed and mobile platforms, and are employed in diverse applications including physical security, access control, law enforcement and counter-terrorism [3] [4] [5] [6] [7].

My project attempts to implement a simple backpropagation neural network capable of recognition of "standard" Florida license plates in use today and to develop a Matlab-based application capable of processing an image file and returning the letters/numbers on that license plate.

Figure 1. Automatic License Plate Recognition System (www.platescan.com)

2. Approach

My approach was to develop training and testing datasets of Florida license plates, and then develop and train a Matlab-based Neural Network. Obtaining a complete training set of license plate imagery was time-prohibitive, so I created a library of synthetic imagery created using image creation tools on www.imagechef.com; this web-based application can create various types of imagery, specifically arbitrary license plate images with user-selected text. The images are not perfect replicas of "real" license plates (as will be seen later in the report), and the images had a watermark at the bottom (see Figure 2), but the images were of sufficient fidelity and resolution to support the development and testing of the neural network.

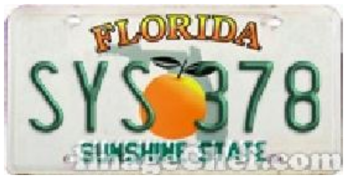


Figure 2. Sample Synthetic License Plate Imagery from www.imagechef.com

I created a training dataset of 36 license plates, each having 6 characters, with a space in the middle (a typical configuration); AAAAAA through ZZZZZZ and 000000 through 999999. This provided a training dataset with all possible characters in all six possible positions, for a total of 216 test images (after segmenting out the individual characters). Each image file was saved as a JPEG image file with the filename being the six characters in the image (e.g., "AAAAAA.JPG"); therefore, the correct value for the image is always the filename itself, without the file extension (made it easy to score the runs).

I also created a testing dataset of 36 license plates with various letter/number combinations (e.g., "NEURAL","NETWRK","SYS378", etc). I did not use these to train the network, but used this dataset (and the training dataset as well) to measure the performance of each network configuration. So the training dataset consists of 216 characters (36 license plates) and the testing dataset consists of 432 characters (72 license plates).

2.1 Image Processing Front End

The image processing "front end" for my license plate recognizer is simple, as I wanted to concentrate on the neural network design. It includes two steps:

1. Convert the color RGB image to gray scale, downsample by 50% in each dimension, and then perform a simple histogram stretch (for contrast enhancement)
2. Extract six (6) characters based on fixed locations; this is illustrated in Figure 3. I "hard coded" the character locations, as they did not vary with the synthetic imagery I used for training. I researched for a smarter adaptive algorithm to extract both the license plate image and the individual characters (see references [8] and [9]), but I considered this beyond the scope of the project, as I wanted to concentrate on the design and testing of the network.

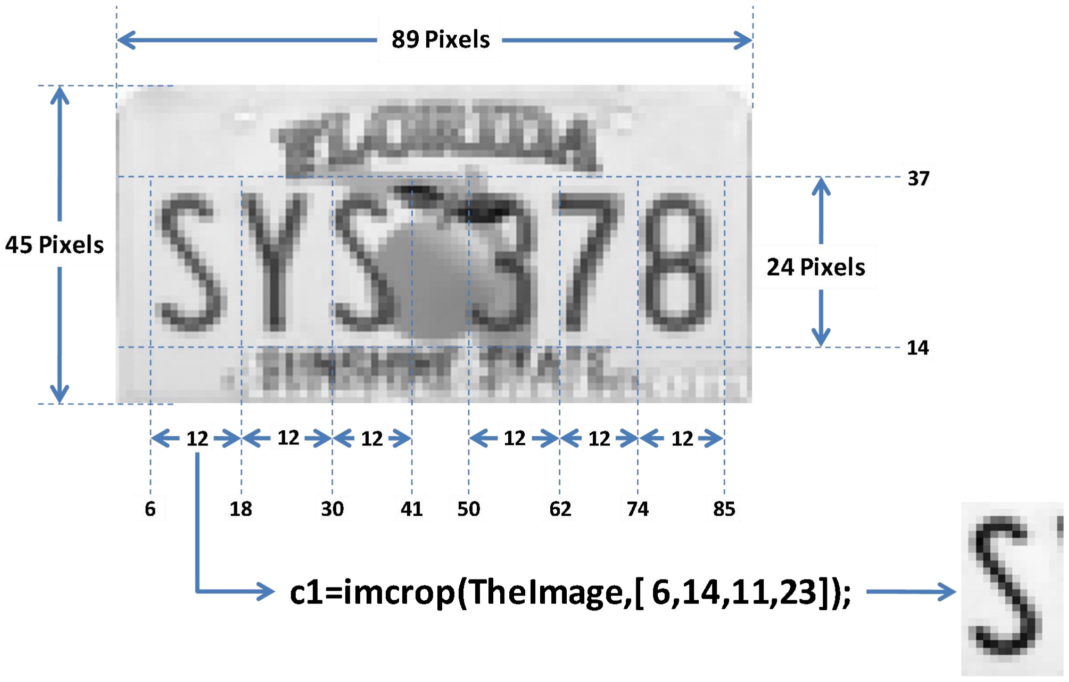
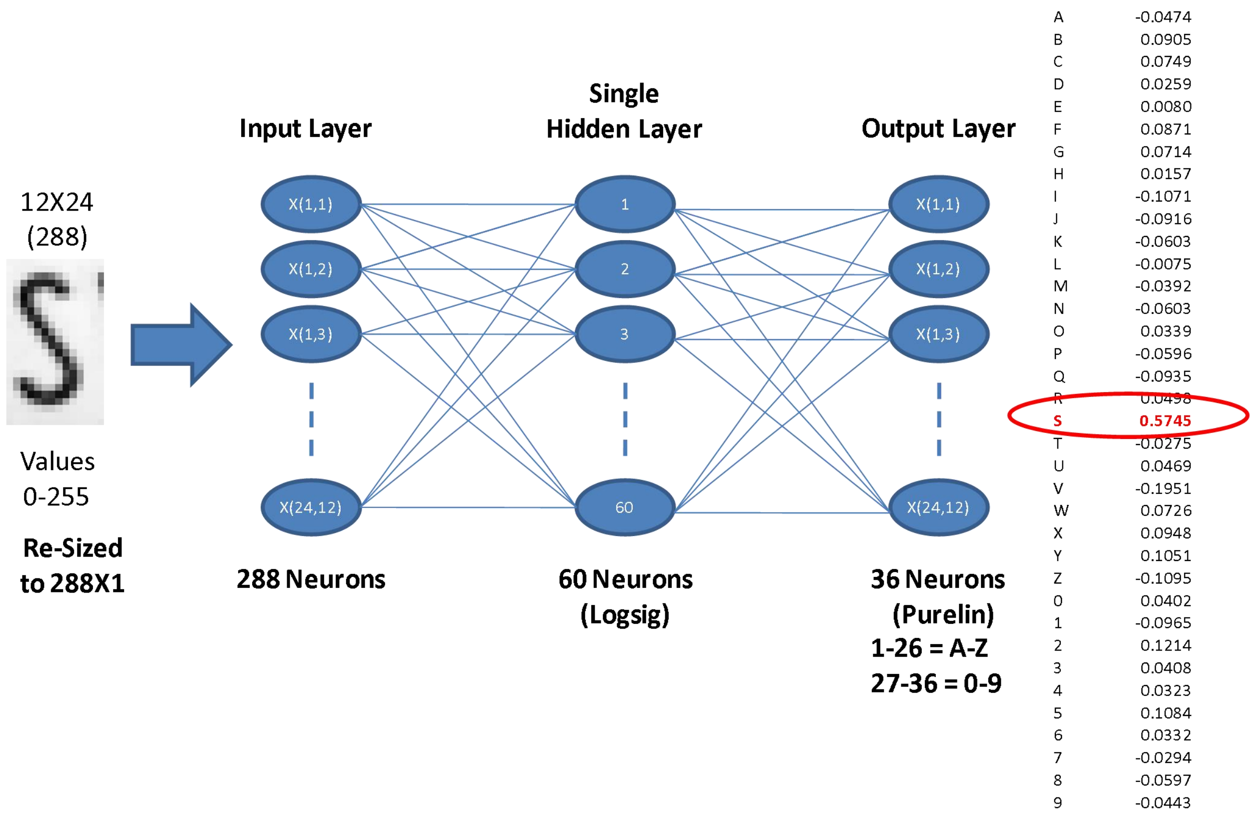


Figure 3. License Plate Character Extraction

2.2 Neural Network Design

The architecture of the selected backpropagation neural network is shown in Figure 4. The image of the segmented character (12 by 24 pixels) was reshaped to a 288 by 1 array for the input layer. I chose 60 neurons for the hidden layer, but experimented with values from 5 to 200 neurons (see Table 2). I also experimented with multiple hidden layers, but it did not provide any additional performance. The output layer has 36 neurons, or "classes", each class representing a letter (A-Z) or a number (0-9). I used *logsig* as the activation function for the hidden layer, and *purelin* for the output layer (I tried *tansig*, but it did not produce as good a result as *logsig*).

For pre-processing, I normalized the input gray scale image values (0-255) to a range of -1...+1 (using the Matlab "mapminmax" function) and then "de-normalized" after running the model; this improved performance dramatically.



A	-0.0474
B	0.0905
C	0.0749
D	0.0259
E	0.0080
F	0.0871
G	0.0714
H	0.0157
I	-0.1071
J	-0.0916
K	-0.0603
L	-0.0075
M	-0.0392
N	-0.0603
O	0.0339
P	-0.0596
Q	-0.0935
R	0.0498
S	**0.5745**
T	-0.0275
U	0.0469
V	-0.1951
W	0.0726
X	0.0948
Y	0.1051
Z	-0.1095
0	0.0402
1	-0.0965
2	0.1214
3	0.0408
4	0.0323
5	0.1084
6	0.0332
7	-0.0294
8	-0.0597
9	-0.0443

Figure 4. Backpropagation Neural Network Architecture

3. Performance Results

I evaluated performance of the network both with the synthetic imagery (from www.imagechef.com) and from digital imagery of four "real" Florida license plates.

After training the network, I developed a short Matlab script (test.m) (with two custom functions) to allow the user to enter a license plate image file name (6 characters) and run the network with the selected image, and then show detailed results from the run; a diagram is shown in Figure 5. The detailed results included the image, the downsampled grayscale image, images of the 6 extracted characters, and a detailed printout of the network results for each of the six characters; see sample in Figure 6. Above each character image are the top 3 characters output from the network, and the output values relative to the top value (that's why its value is always 1.0). This allows insight into characters the network may have trouble with (especially during noise injection, see below). It also could be used to indicate ambiguity in the network; if the top 2 values were quite close, I might want to output "?" rather than a letter or a number to indicate this.

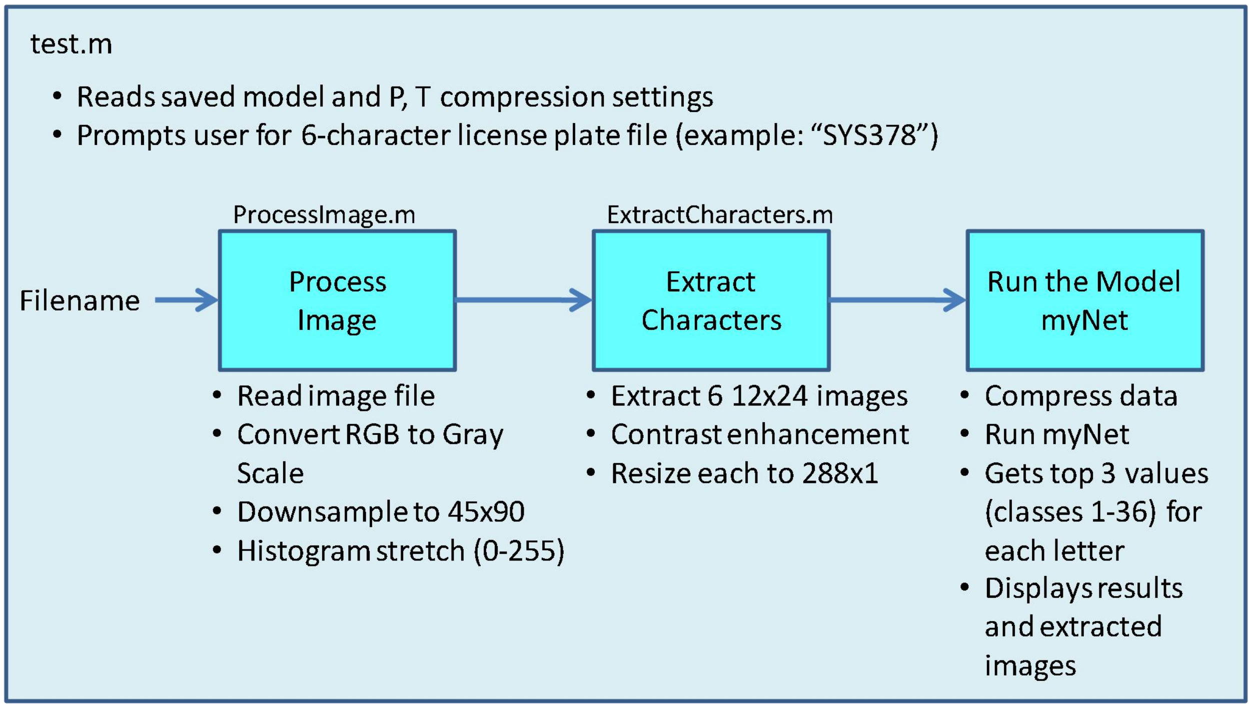

test.m
• Reads saved model and P, T compression settings
• Prompts user for 6-character license plate file (example: "SYS378")
Filename
ProcessImage.m
Process Image
• Read image file
• Convert RGB to Gray Scale
• Downsample to 45x90
• Histogram stretch (0-255)
ExtractCharacters.m
Extract Characters
• Extract 6 12x24 images
• Contrast enhancement
• Resize each to 288x1
Run the Model myNet
• Compress data
• Run myNet
• Gets top 3 values (classes 1-36) for each letter
• Displays results and extracted images

Figure 5. Testing Phase Functional Flow Diagram

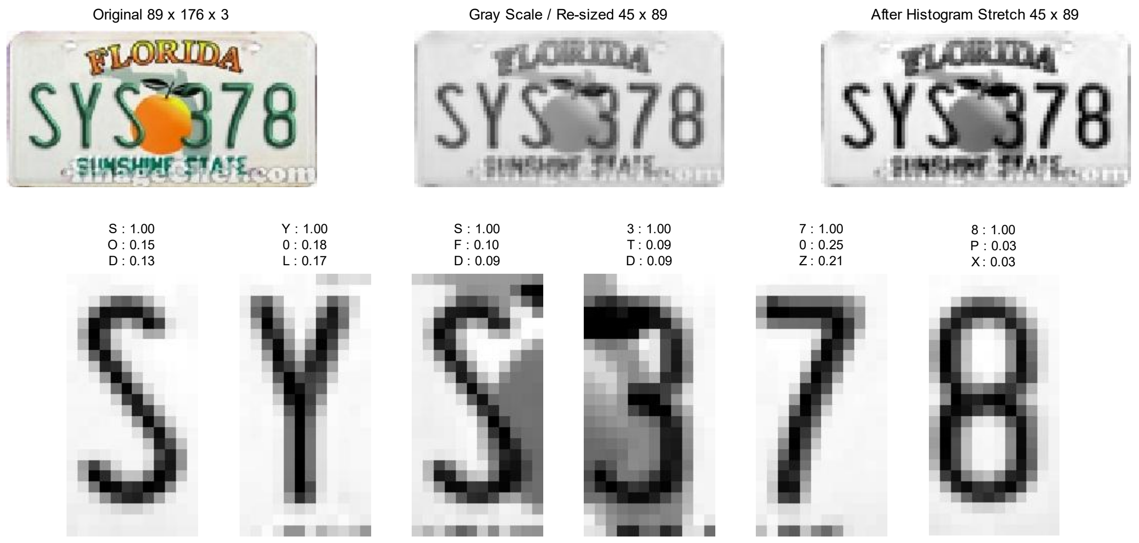


Figure 6. Sample Output from Testing

3.1 Performance Evaluation

I evaluated 5 potential backpropagation training algorithms, as summarized in Table 1. I used a maximum number of epochs of 2,000, and a training goal value of 0.0001. I could not get the Levenberg-Marquardt algorithm to work in Matlab due to memory issues (even after reducing the number of hidden neurons). Overall, the conjugate gradient (traincgf) was the overall winner, providing the best performance with the smallest number of neurons (I measured training times for each, but was not considered in the overall trade study). A summary of the performance of the conjugate gradient algorithm with different numbers of neurons in the hidden layer is summarized in Table 2; I selected 60 neurons, as it performed well and was the best balance of performance and complexity. As shown, performance tends to drop off as the number of neurons in the hidden layer increased above 100 or so; I presume this is due to overtraining.

Table 1. Sumary of 5 Backpropagation Learning Algorithms Evaluated

Algorithm	**# Neurons Hidden Layer**	**Training Time**	**Comments**
Batch Gradient Descent (traingd)	100	81 sec	Did not reach training goal; Poor performance (47% correct)
Gradient Descent w/Momentum (traingdm)	20 - 100	15 - 39 sec	Did not reach training goal; (0.019) Good performance (97 - 99.5% correct) Learning rate = 0.9; Momentum constant = 0.1;
Levenberg-Marquardt (trainlm)	100	N/A	Out of memory
Variable Learning Rate (traingdx)	20 -100	15 - 37 sec	Did not reach training goal; Performance = 90-100% correct Learning rate = 0.9; Learning rate increment value = 1.10
Conjugate Gradient (Fletcher-Reeves Update, traincgf)	5 - 200	22 - 82 sec	Best overall performance; selected 60 neurons in hidden layer

Table 2. Algorithm Performance as a Function of Number of Neurons in Hidden Layer

# Neurons	**Training Time (sec)**	**% Epochs**	**Final Training Value**	**% Correct**
5	22.5	2,000	0.0999	7.8%
10	21.8	2,000	0.0836	38.2%
20	27.3	2,000	0.0460	90.7%
30	31.3	2,000	0.0240	97.7%
40	35.8	2,000	0.0091	99.3%
50	33.8	1,543	0.0001	100.0%
60	42.2	1,317	0.0001	100.0%
70	49.7	1,408	0.0001	100.0%
80	39.6	1,066	0.0001	100.0%
90	39.2	1,102	0.0001	100.0%
100	45.9	1,026	0.0001	99.8%
200	82.9	981	0.0001	99.3%

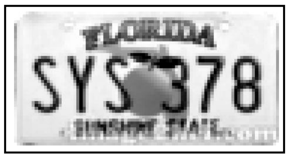
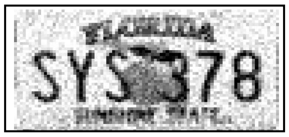
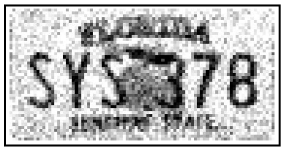
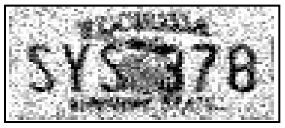
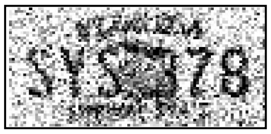
3.2 Performance with Noisy Imagery

I experimented with adding “white noise” by using the Matlab function

imnoise(Img,'gaussian‘,mean,var)

I used various values of noise variance with a zero mean, as summarized in Table 3. As shown, network performance was quite sensitive to noise, as expected; I would need to add some sort of noise filter to remove these artifacts in a "real" system.

Table 3. Performance Results with Noisy Images

Noise Variance	Sample Image	% Characters Correct	License Plates Correct
0		100%	72/72
0.01		87.7%	26/72
0.02		70.8%	7/72
0.04		52.1%	1/72
0.10		23.8%	0/72

3.3 Performance with Real Imagery

Performance against the 4 "real" images I selected was poor. First, the synthetic images generated from the web application are a close approximation of "real" license plates, but have significant differences that impacted training (and subsequent testing against real imagery). As shown in Figure 7, the characters are approximately the same shape, but differ in character with and spacing; note also that the style of the digit "4" is different. Also, the background image has changed in the most recent Florida plates from the imagery shown on the right (an older style). Of course, the synthetic images also do not have the license plate stickers, frames or mounting bolts.

I modified the character extractor Matlab code to re-scale the "real" imagery to the size (12X24) used for training, but performance was still poor.

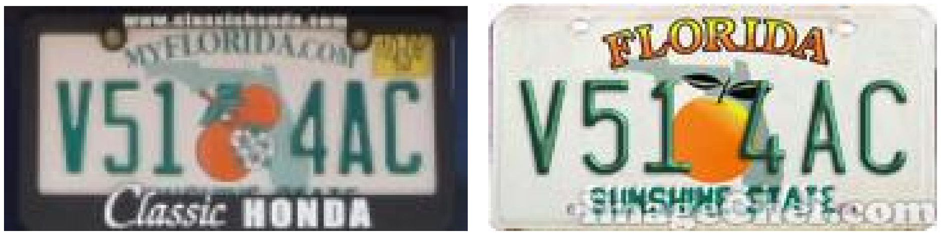


Figure 7. Comparison of Real License Plate Image (Left) With Synthetic Image (Right)

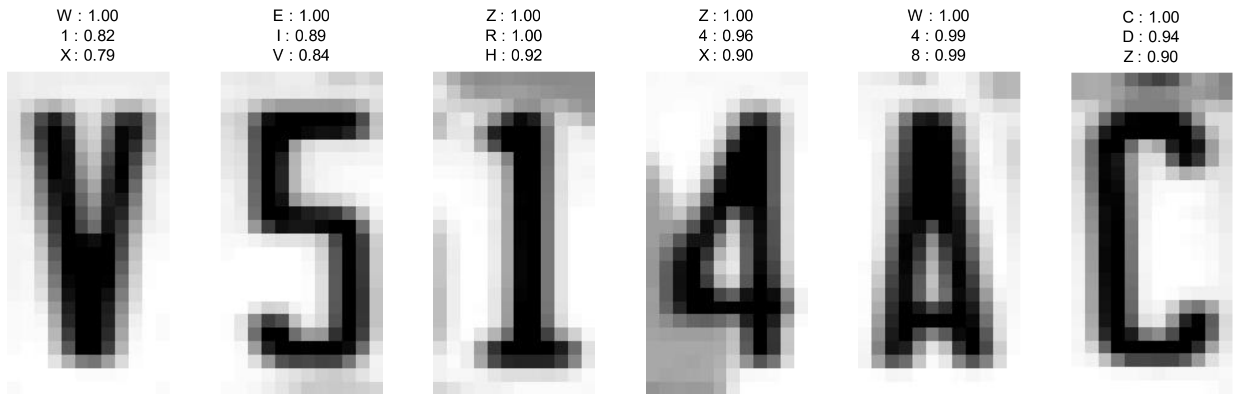


Figure 8. Network Results from Real License Plate Image "V514AC"

4. Conclusions

The completed backpropagation neural network was successful at recognizing all the synthetic license plate images in the training and testing dataset. Neural networks are a good fit to the “OCR problem” for license plate number recognition, as demonstrated in this project (as well as the systems described in my references). The network was sensitive to image noise, as expected; a "complete" recognition system would require some image pre-processing to remove artifacts to improve performance.

Furthermore, I would need to train on “real” imagery in addition to (or instead of) synthetic imagery. Synthetic imagery is no substitute for actual imagery for training a "real world" system. This would be a major expense in developing an image processing system like a license plate recognition system.

A complete system would also require additional image processing “front end” needed to extract license plate “rectangle” from complete image rather than my simplistic approach. It would also require a robust character extractor; the fixed location approach obviously lacks generality.

But given these simplifications, the concept of applying a backpropagation neural network to license plate recognition was successfully demonstrated in this project.

5. References

[1] Ron, Bar-Hen, “A Real-time vehicle License Plate Recognition (LPR) System”, 2002, Israel Institute of Technology (http://visl.technion.ac.il/projects/2003w24/)

[2] “Automatic License Plate Recognition System”, web site, www.platescan.com ; commercial system in use by numerous police agencies; based on proprietary neural network recognition engine

[3] Geovision License Plate Recognition System (http://www.ezcctv.com/license-plate-recognition.htm) ; commercially available system from Europe for parking systems, etc

[4] Automatic License Plate Recognition”, web site, http://www.platerecognition.info; good overview of algorithms and software

[5] YouTube video of License Plate Recognition System used by police in British Columbia, web page, http://www.youtube.com/watch?v=ENGY1CD9y_4

[6] Gaumont, Norm, “The Role of Automatic License Plate Recognition Technology in Policing: Results from the Lower Mainland of British Columbia”, Police Chief Magazine, March 2009, http://policechiefmagazine.org/magazine/index.cfm?fuseaction=display_arch&article_id=1671&issue_id=112008

[7] “Automatic number plate recognition”, Wikipedia article, http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

[8] Shapiro, Vladimir, Dimov, Bonchev, Velichkov and Gluhchev, "Adaptive License Plate Image Extraction", International Conference in Computer Systems and Technologies, 2003, http://ecet.ecs.ru.acad.bg/cst04/Docs/sIIIA/32.pdf

[9] Koval, Turchenko, Kochan, Sachenko and Markowsky, “Smart License Plate Recognition System Based on Image Processing Using Neural Network”, IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 8-10 September 2003, Lviv, Ukraine (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.7478&rep=rep1&type=pdf)

Appendix A: Explanation of Files Used in this Project

Table 4. Listing of Project Files

Matlab File	**Description**
project.m	Main project script; performs training and testing; calls custom functions ProcessImage and ExtractCharacters
ProcessImage.m	This function reads an RGB license plate image file and returns the image object of the original, plus a gray scale version, and a "histogram-stretched" grayscale version (to improve contrast). The grayscale image and the histogram-stretched version are re-sized to 45 pixels high and it maintains the aspect ratio of the original image (reduces image by 50% in both directions).
ExtractCharacters.m	This function extracts 6 characters from the license plate image (45 pixels high by 89 pixels wide). The characters extracted are 12 pixels wide by 24 pixels high. It performs a simple histogram stretch to improve contrast, and slightly saturates the image (by 10% on the high and low sides).
test.m	Simple script to run the network on a user-selected image file, and to display intermediate results for further analysis.
testall.m	Similar to "test.m", but runs the network on the entire training dataset and test dataset, just to show the overall performance.
testreal.m	Same as "test.m", but calls a modified version of "ExtractCharacters" to account for differences in the widths and spacing of the characters in "real" license plate images.
testnoise.m	Same as "test.m", but adds Gaussian white noise on the image to be processed before passing to the trained network.
testallnoise.m	Same as "testall.m", but adds noise to the image files, as in "testnoise.m"
AAAAAA.jpg, BBBBBB.jpg, etc	Synthetic license plate image files created from web-based application www.imagechef.com.