Digital License Plate Character Segmentation Combined with Identification Using a Neural Network

Jacob Breckenridge
Digital Signal Processing - ECE 529
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721
jbrecken@email.arizona.edu

Abstract

The process of License Plate Character Identification (LPCI) can be broken down into two primary stages: character location and character identification. The scope of this paper is to utilize the digital license plate extraction algorithm described by Faradji, Rezaie, and Ziaratban in their paper, "A morphological-based license plate location" in the first step towards character location (the step of plate location) [2]. I used Matlab code for implementing the algorithm available on Alireza Asvadi's website [1]. I extended their methodology to include character location methods in Matlab, and I implemented character identification by using a simple neural network built in Python using the Keras API's model. I developed a 2-D digital filtering algorithm of my own design using Matlab to build a successful database for training the network, provided a small set of characters. I implemented existing functions in Matlab for convolution, geometric transformations, and other operations.

1 Background

Introduction

Having a formal method of LCPI can have many challenges. First, one must successfully be able to locate the position of a license plate on a vehicle. What may seem a simple task can be a struggle for a computer which cannot decipher the meaning of one pixel from the next. For this reason, it is useful to guide the program with an algorithm of image filtering that reductively identifies a location of maximum likelihood for a license plate location. Many complications may arise in a single picture, you may have multiple plates, blurs, lights, and partial obstructions to the visibility of the plate.

Further, once the plate is successfully located, one must identify the location of the characters and then identify the specific characters. This can equally be challenging as often there are issues with identifying the characters; there may be distortions to the plate from colors or other objects located on the plate. The final process of character identification can also have difficulty due to low qualtiy images or character type differences from region to region. Even within one region, character types may change over the years so a license plate from 1970 may have a number 4 that looks rather different from a plate dated 2018.

Therefore, for the scope of this paper, it is necessary to define parameters for testing of the algorithm. Once an algorithm is established to be successful, it then can be extended to more parameters.

1.1 Image Parameters

For testing consistency, images were taken at a distance of about 3-5 feet from each vehicle. The license plate was positioned within the center vertical and center horizontal thirds. Primarily, this was to remove background noise from the image for the plate location process. The plates were also restricted to Arizona license plate with white backgrounds and dark characters. This reduces the need for extraneous needs of filtering of images for testing of the model validity *(Figure 1)*. These constraints were required to deal with a line of issues that arose during experimentation with the model, including, but not limited to:

- Background noise preventing successful location of the plate.
- Occluded characters and noise on the plate.
- character separation for identification due to small size of characters if image too small.
- character separation for identification due to small size of characters if plate too distant.
- variety in character type preventing well performing neural network.
- color of plate in contrast to the characters affected separation of characters.

Figure 1: Input Image Features

1.2 Character Parameters

Once the characters have been segmented, they are run through a neural network. The neural network must be trained on classified samples of the letters in order to learn to make identifications. The network needs a large amount of samples to train on, therefore it is necessary to use a tool for data generation. The python neural network API, "Keras," comes with image filtering and modulation tools. However, these filters often utilize rotations and flips that might be too destructive for characters. Therefore, in order increase the amount of data, I created two filtering tools in Matlab. The first was an overall noise generating tool that took the images and applied various functions to the data. In the end, the first set was useful for noisy samples, but the neural network had low testing results when trained only on this noisy set. And it needed more samples with less variation. Therefore, the second filtering tool I created used less dramatic filters and only output binary valued images.

2 License Plate Location Process

2.1 Step 1: Grayscale

At the start of the process, the image is converted to grayscale *(Figure 2)* which helps with edge detection and future processing of features.

2.2 Step 2: Edge Detection

Vertical edge detection is used to identify likely plate regions. The sides of the plate and characters contained within will have many long vertical lines. A vertical Sobel operator is convolved with the matrix for the plate to locate vertical edges (Figure 3). The vertical Sobel operator is defined as

$$SobelOperator = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Figure 2: Grayscale

Figure 3: Sobel Mask

Figure 4: Normalized

After the Sobel Mask is applied to the image, the output is normalized to determine the strongest regions *(Figure 4)*.

2.3 Step 3: Threshold using Otsu Method and Horizontal Histogram

The edge image is binarized using the Otsu method *(Figure 5)*, which "involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold."[3] Otsu showed that the optimal threshold is the one that maximizes the variance between the two classes.[4] This provides a maximum candidate for the the license plate identifying that locally it has maximum variation. A horizontal histogram *(Figure 6)* is used to explicitly identify the area with maximum variation above a chosen threshold as the likely horizontal region in the image for the plate*(Figure 7)*.

Figure 5: Threshold

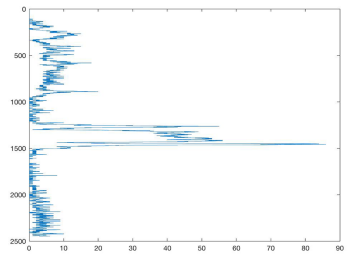


Figure 6: Horizontal Hist

Figure 7: Horizontal Isolation

2.4 Step 4: Dilation of Threshold

Next in the algorithm, vertical and horizontal dilations are applied to the threshold. The vertical dilation operator is a structuring element of height 80 pixels by width 4 pixels *(Results: Figure 8)* and the horizontal dilation operator is a structuring element of height 4 by width 80 pixels *(Results: Figure 9)*. The dilation takes the complement of the black pixels which lie within neighborhoods of the shape of the operator around the white pixels *(Figure 10)*.

$$D_y = [80 \quad 4] \qquad D_x = [4 \quad 80]$$

After the vertical and horizontal dilations are applied, the intersection is found. The holes in the intersection are filled.

Figure 8: Vertical Dilation

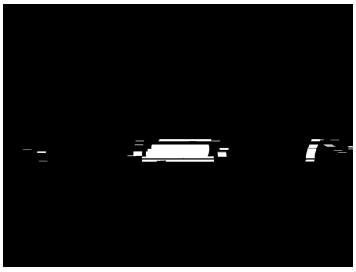


Figure 9: Horizontal Dilation

Figure 10: Intersection

2.5 Step 5: Detection of Likely Plate Region

Finally, a horizontal erosion structuring element is applied to the image. This takes the complement of the set of pixels in the white region which intersect with the neighborhoods of the shape of the operator elements in the black pixel set. After the erosion is applied, the largest region is determined to be the most likely candidate region for the the plate *(Figure 11)*. The region is extracted and binarized with the characters pulled as the foreground for the character identification process *(Figures 12-13)*.

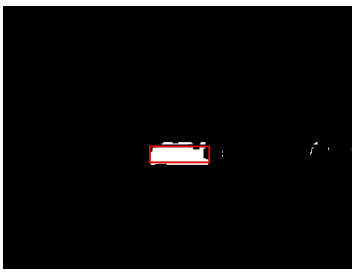


Figure 11: Largest Region

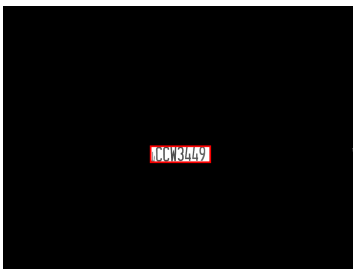


Figure 12: Plate Location

Figure 13: Cropped Image

3 Character Location Process

The character location process can be done in many ways, including histogram analysis, region analysis, etc. For the project, I used bounding boxes which analyses connected regions. The alternative approach of histogram analysis has similar results as described by Rathore, Manisha, and Kumari in [5]. In their research, the success of the plate and character location process has similar requirements on the location of the image as described here.

First, the characters are eroded to remove as many connections as possible between regions. This prevents as much effect from obstruction from dirt on the plate, etc., as possible *(Figure 14)*.

The bounding box feature is used in Matlab to determine the location of all of the objects in the cropped plate (Figure 15). There are often extraneous features on the plate, so it is necessary to have an algorithm for determining what is likely a character versus a non-character feature of the plate. The detected plate should have mostly characters as boxes in the region (Figure 16). To discard the non-character features, the following criteria were applied to the boxes:

- The height of the box must be greater than 70% of average height for all the boxes in the region.
- Height:Width ratio of the character is greater than the average H:W ratio of all characters minus one to offset larger H:W ratios.
- Twice the area of the box must be greater than the difference between the median area and 1000.

- The vertical position of the top to the box must be less than 25% of the average vertical position of all the boxes.

The goal of these parameters is to allow for some variation due to angle of the photo and difference of the characters, however if it too greatly fails any of these parameters the box will be discarded as a potential character. Some leeway is necessary for other issues such as the character for a one may be very thin.

Figure 14: Eroded Characters

Figure 15: Bounding Boxes

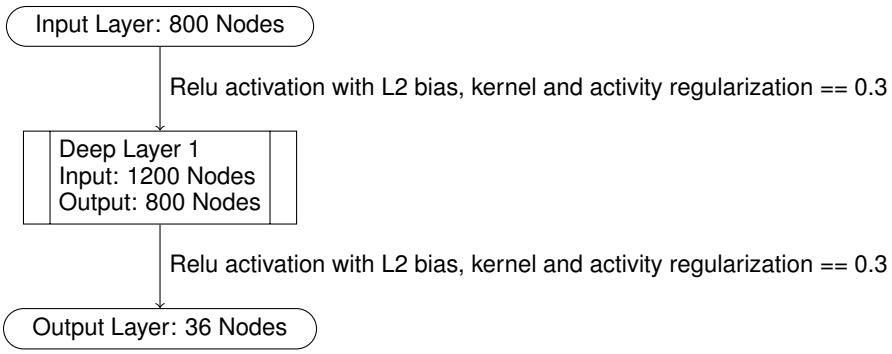
Figure 16: Detected Characters

4 Neural Network for Character Identification

A Multilayer Perceptron was used for learning the character types. The Network was built in Python using the Keras Python Deep Learning Library. A multilayer perceptron is a robust method of image learning and identification that consists utilizes probability weights of each character to learn what each character should look like. Each image in the dataset is restricted to 40 pixels height and 20 pixels width. The characters from the license plate extraction are converted to binary and then turned in to a 1x800 row-matrix that the network learns the probabilistic arrangement for each character. Not converting the characters to binary is possible, however doing so returned the highest accuracy with the model (*Table 1* in *Section 5.3* reviews the results of experiments with combinations of the data).

4.1 Network Model

The network takes input of an 1x800 row vectors. Each input has a corresponding classification representing each character that the network is trained on. The network then passes the data through one layer of weights which learn the connection between the input and the output. It creates a weight for all the values of the inputs of the array, the weights represent activation of nodes in the layer. Weighted activation of the nodes determines the network's guess for the classification of the character. It learns whether it was correct in its estimation, corrects its weights if necessary and then retrains. The output is a single layer of 36 nodes representing each character. The node with the highest estimated probability on the output is the classification.



In order to perform well on various situations, the neural network needs a large dataset from which it can learn. However, it can be quite challenging to retrieve copious examples of characters from license plates. Therefore, it is useful to have a program that can apply numerous filters and noise in combination to generate a large dataset of characters from which the network can learn.

5 Character Generation for Neural Network Data

In order to build a neural network for character identification a large dataset is necessary for training the network. One could pull images of license plates and extract characters, but the cost in time

of doing so would be enormous. The method described repeatedly applies filters in iteration to a small set of generated fonts. There were about 5 fonts used for training data and one for testing data. Some of the data used in training was filtered using noise and pixelation tools available on Adobe Photoshop and Adobe Illustrator. *(See [7]-[11] for font information).*

However, the scope of the project was primarily to generate data using a set of filters which use a small set of characters to generate a large, useful one. Two filters were used, which applied a combination of two filters in incrementally increasing amounts to the generated fonts.

5.1 The First Filter

The first filter applied strong variations to the data using the following filters and formulas applied to the images.

The following features were applied to the images in combinations of two filters at a time:

- Cropping
- Convolution of the image according to the i-th iteration to create a beveled-like version of the image

$$ConvolutionOperator = \begin{bmatrix} 1/64 & 100-i & 1/64 \\ 100-i & 1/16 & -100+i \\ 1/16 & -100+i & 1/16 \end{bmatrix}$$

- Another Convolution of the image according to the i-th iteration to create a heavier beveled-like version of the image

$$ConvolutionOperator = \begin{bmatrix} 1/32 & (100-i)/128 & -1/128 \\ (100-i)/128 & -1/128 & (100-i)/128 \\ -1/128 & (100-i)/128 & 1/32 \end{bmatrix}$$

- Convolution of the image using a Gaussian Blur Operator

$$GaussianBlurOperator = \begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/6 \end{bmatrix}$$

- A meshgrid of 2-D sine noise was generated and multiplied by the character image. The sin noise was generated with respect to the i-th iteration according to the function:

$$Grid = i * sin((y+i) * (ix-i))$$

- A meshgrid of 2-D cotangent noise was generated and multiplied by the character image. The Cotangent noise was generated with respect to the i-th iteration according to the function:

$$Grid = 23 * cot((y+i) * (ix-i))$$

- A shear was applied with respect to the i-th iteration according to the geometric transformation:

$$GeometricTransformation = \begin{bmatrix} 1 & 0 & 0 \\ -0.1*i & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Post-Filtration of this filter, the images were further run through variations of noise and deblurring as well as binarization and inverting of the colors. Resulting in 8 outputs for every iteration of every filtration of the images *(Figure 17)*. Making it easy to generate an immense amount of data from which the neural network could learn. However, training only on this dataset, the neural network had issues with characters that had small variations. It often mislabeled ones for sevens or T's. It also mislabeled eights as B's fairly often, etc. Improvement in the network's ability to label characters was found when it was trained on a dataset that had smaller variations in the characters.

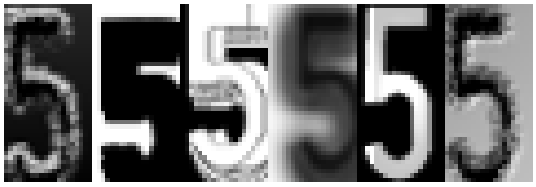


Figure 17: Samples from first set

5.2 The Second Filter

The second filter applied used smaller variations in one font to stuff the network with a firmer base form for each character. Each filter was also applied two effects at a time and then the output was binarized *(Output in Figure 18)*.

- Cropping
- A Band-Limited adjustment of the levels of the character cutting off the levels with respect to the i-th iteration at

$$LowerLevel = 0.002 * i \qquad UpperLevel = 1 - 0.002 * i$$

- Gaussian Noise
- Shear Left with respect to the i-th iteration according to the geometric transformation:

$$GeometricTransformation = \begin{bmatrix} 1 & 0 & 0 \\ -0.00025 * i & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Shear clockwise with respect to the i-th iteration according to the geometric transformation:

$$GeometricTransformation = \begin{bmatrix} 1 & 0.0000025 * i & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Erosion of the character
- Dilation of the character
- Gaussian Blur

Figure 18: Samples from second set

5.3 Network Model Results Training

It can be seen from *(Table 1)* that the effects of filtering images can really help or hurt a dataset. When over-filtered, the network is not very useful for recognizing simple images. This is important for text and character recognition as there can be a lot of variation between characters, but overall they have truly subtle distinctions. Therefore it is also helpful to evaluate on data that has minute changes. The network can learn the overall primary shape of the characters.

Table 1: Accuracy of Neural Network

Training Data	# of Samples	Epochs Trained	Test Set	Accuracy
Filter 1 Only	117908	10	Filter 1 Only	40.2%
Filter 1 & Filter 2	220724	10	Filter 1 Only	51.6%
Filter 1 Only	117908	10	Filter 2 Only	18.2%
Filter 2 Only	102816	10	Filter 2 Only	71.9%
Filter 1 & Filter 2	220724	10	Filter 2 Only	87.9%
Filter 1 & Filter 2	220724	200	Filter 2 Only	90.3%

6 Results and Evaluation of Model

One of the most difficult problems for license plate detection is collection of data. This is a problem for creating a method that is useful on a grand scale past the parameters of the testing. Even with large amounts of data, the number variations of plates and the number of potential obstructions to character detection is innumerable. Data collection is simply also difficult because it ultimately requires one to take images of car license plates. Even friends and family members seem to feel uneasy about letting you take images of their plates, let alone strangers. Hence, to evaluate the model I had to use an external website collection of images of license plates [6]. These images have few parameters of location of the plates in the image. Three primary categories of issues occurred frequently enough to have a large effect on the results: a noisy background preventing location of the plate*(Figure 19)*; a noisy foreground (i.e. reflections on the vehicle) preventing location of the plate *(Figure 20)*; low quality images resulting in difficulty identifying characters. The results of the testing on this data were as follows *(Table 2)*:

Table 2: Testing on Data from [6]

Number of Images	Plate Detection	Char. Detection	Char. Accuracy	Erroneous Char.
100	32%	87.5%	63.7%	49%

Indicating that of 100 images tested from the website, 32 of the plates were detected accurately. Of the 32 plates detected, 87.5% of the time the characters were segmented accurately. Of those characters segmented, 63.7% of them were accurately labeled. And 49% of the time plates were accurately detected, there were non-characters which were labeled as characters. I also calculated how commonly each of the three issues listed above occurred to cause a failure of either plate location, character segmentation, or character identification above 50% accuracy *(Table 3)*:

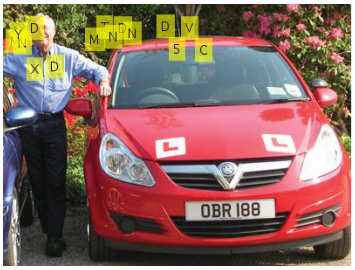


Figure 19: Background Noise

Figure 20: Foreground Noise

Figure 21: 83.3% Accurate

Table 3: Failure causes

Background Noise	Foreground Noise	Too Low Quality of an Image
50%	35.7%	14.3%

Looking at the causes of failures can help identify the need for parameters of control in the image if an identity were to utilize this method for plate identification. Although the data fit within the parameters is small, we can compare the model when the parameters to the image are applied. From Table 4, one can see the improvement in photo selection as an early parameter for the plate detection

process. Also, the importance of a high quality image at this stage is great for being able to have accurate character identification once the characters are located. Per image, there are also fewer erroneous characters listed.

Table 4: Controlled Testing

Number of Images	Plate Detection	Char. Detection	Char. Accuracy	Erroneous Char.
8	87.5%	85.7%	92.9%	37.5%

Figure 22: Example Result

Figure 23: Example Result

7 Conclusion

Many algorithms have been suggested for location of plates and segmentation of characters. However, many of these algorithms require restrictions to the images in order to have a successful starting algorithm that can be branched out from for grander scopes. Machine learning can be successful for character identification when complemented by current filtering techniques. Filtering can be utilized for image preprocessing. And, in this case, it was necessary to use image filtering techniques to generate a copious amount of data from a small set. One set of the data was over-filtered and although its data gave decent results, it was not entirely useful alone without a set of slight variations in the data. The second set of data with fewer increments in the variation really seemed to help the data learn what to look for.

8 References

Resources:

[1] Alireza Asvadi. (2013) “License Plate Detection Using Morphological Operators.” http://a-asvadi.ir/license-plate-detection-using-morphological-operators/

[2] Faradji, Farhad, Amir Hossein Rezaie, and Majid Ziaratban. "A morphological-based license plate location." Image Processing, 2007. ICIP 2007. IEEE International Conference on. Vol. 1. IEEE, 2007.

[3] Greensted, Andrew. "Otsu Thresholding." *The Lab Book Pages* http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html. 2010

[4] Otsu, Nobuyuki. "A threshold selection method from gray-level histograms." IEEE transactions on systems, man, and cybernetics 9.1 (1979): 62-66.

[5] Rathore, Manisha, and Saroj Kumari. "Tracking number plate from vehicle using matlab." International Journal in Foundations of Computer Science & Technology (IJFCST) 4.3 (2014).

Image Database:

[6] Anonymous. *NumberPlates.com* http://www.numberplates.com. Accessed May 1, 2018.

Font Sources:

[7] License Plate USA font. Copyright SpideRaYsfoNtS 2012-2018.

[8] Custom License Plate Font Copyright 2000-2018 Stargazer Designs.

[9] Heavy Equipment Font. No Copyright. Public Domain.

[10] GL-Nummernschld-Eng Font made by Gutenberg Labo, 2009. No Copyright.

[11] Registration Plate UK. SpideRaYsfoNtS 2012-2018.

All fonts and images used in paper were used for educational purposes only.