ENGN6528 - Computer Vision: Term Project

John Aslanides

June 7, 2015

Abstract

We present a simple license plate recognition system which uses morphology-based segmentation, and artificial neural network-based recognition. The system achieves a license plate recognition rate of 75% on typical cropped Australian license plate images. We propose several methods to improve the classification rate.

1 Introduction

Automated license plate recognition is an important application of computer vision that is useful in numerous settings, including traffic data collection, law enforcement, motorway toll collection, and managing parking lots [1]. A system with the capability to robustly and accurately recognise license plates from various jurisdictions can be of great value when applyed to any of the above tasks; for this reason there has been significant research and development in this area in the past two decades [2]. The most typical license plate recognition system involves three steps [3]:

1. localising a license plate within a cluttered scene, possibly containing more than one car,
2. segmenting the license plate into its constituent glyphs, and
3. performing optical character recognition (OCR) on these glyphs to obtain the license plate string.

The problem we will be solving with our system consists of steps (2) and (3); that is, the images we will be dealing with consist of cropped license plates, and the task is to output a string corresponding to the correct license plate.

2 Background

There has been considerable ongoing research in the area of license plate recognition over the last twenty years, with a rich literature exploring different ideas and techniques. Below we provide some background on these techniques, concentrating on segmentation and recognition (we assume that license plate localisation and rectification has already been performed). We also provide a brief review of the theory of neural networks.

2.1 License Plate Segmentation

There are many difficulties in license plate character segmentation. Some of these include: image noise, plate frame occlusion, presence of bolts or rivets, and extraneous slogans and images on the license plate. There are numerous approaches to dealing with these in the literature; these include segmentation based on the Hough transform [4], projection histograms [5], clustering [6], template matching [7], and morphology [8]. The choice of segmentation technique seems to be largely a matter of taste; no single technique explored in the literature dominates the others in terms of performance; in one review study that compared numerous techniques, a variation in performance of less than 5% was recorded [8]. For this reason, there is no true 'state of the art' method, just a collection of different techniques with their own advantages and disadvantages.

2.2 License Plate Recognition

Once the license plate characters have been segmented, the problem essentially reduces to the standard OCR problem: given an image of a character represented in the pixel domain, to recognise the character. Again, there are numerous approaches to this problem in the license plate context; these include 'classical' methods such as template matching and the Hotelling transform [5], and learning based methods such as support vector machines (SVM) and neural networks [9, 10, 11, 12].

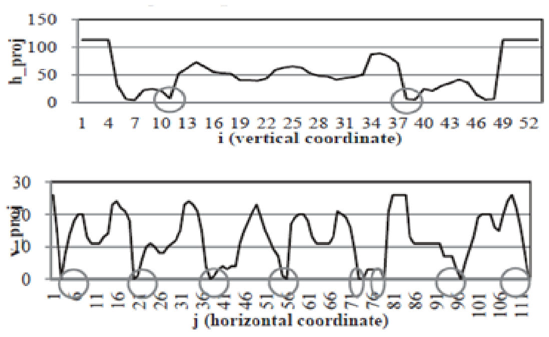

Figure 1: An example of the projection histogram technique. Troughs in the histogram correspond to gaps between symbols [8].

2.3 Artificial Neural Networks

Artificial neural networks (ANNs) are a powerful tool for classification and function approximation, which have been effective in many diverse domains and settings[1]. Algorithms that employ neural networks are the state of the art in many domains, including handwriting and character recognition [13]. The setup of a feed-forward neural network is as follows. The network is composed of layers of 'neurons', which each take as input a linear combination of the outputs of the previous layer, and apply a nonlinear activation function to this input:

$$a_j^{(i)} = \sigma\left(\sum_{k=1}^{N_{i-1}} w_{ij}^{(i-1)} a_k^{(i-1)}\right).$$

Here we define $a_j^{(i)}$ as the output of neuron j in layer i, the $w_{ij}^{(i-1)}$ are the weights joining layers $i-1$ and i, and σ is the nonlinear activation function, which is typically the logistic sigmoid:

$$\sigma\left(x\right) = \frac{1}{1+e^{-x}}.$$

Note that in the typical setup, each layer is fully connected to its adjacent layers. The first layer is the input data, and the last layer is the output (corresponding to a class, or regression, depending on the setting). Intermediate layers are called 'hidden layers', and these can be thought of as learning intermediate basis functions or representations for the data. Considered layer-by-layer, a neural network comprises an iterated series of generalised linear models. The entire L-layer network can be written down as a vector-valued function with elements given by

$$y_i\left(\boldsymbol{x}\right) = \sigma\left(\sum_{i_{L-1}=1}^{N_{L-1}} w_{ii_{L-1}}^{(L-1)} \sigma\left(\sum_{i_{L-2}=1}^{N_{L-2}} w_{i_{L-1}i_{L-2}}^{(L-2)} \sigma\left(\ldots\sigma\left(\sum_{i_1=1}^{N_1} w_{i_2 i_1}^{(1)} x_{i_1}\right)\right)\right)\right),$$

where $w_{ij}\left(k\right)$ is the weight between unit i of layer $k-1$ and unit j of layer k.

The classifier itself will simply take the index of the maximum value of $\boldsymbol{y}$; this is interpreted as the class that the input $\boldsymbol{x}$ belongs to, with highest probability, according to the model implied by the network. Hence out classifier C simply takes the output of the neural network and computes the arg max:

$$C\left(\boldsymbol{x}\right) = \arg\max_i y_i\left(\boldsymbol{x}\right).$$

Training can then be done in conjunction with gradient descent and an appropriate loss function E using the backpropagation algorithm, which amounts to an application of the chain rule for derivatives [14]:

$$\frac{\partial E}{\partial w_{ij}^{(l-1)}} = \frac{\partial a_j^{(l-1)}}{\partial w_{ij}^{(l-1)}} \sigma'\left(a_j^{(l)}\right) \sum_k w_{kj}^{(l)} \frac{\partial E}{\partial a_k^{(l)}}.$$

[1]For an interesting overview of recurrent neural networks and their uses, see 'The Unreasonable Effectiveness of Recurrent Neural Networks' at http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

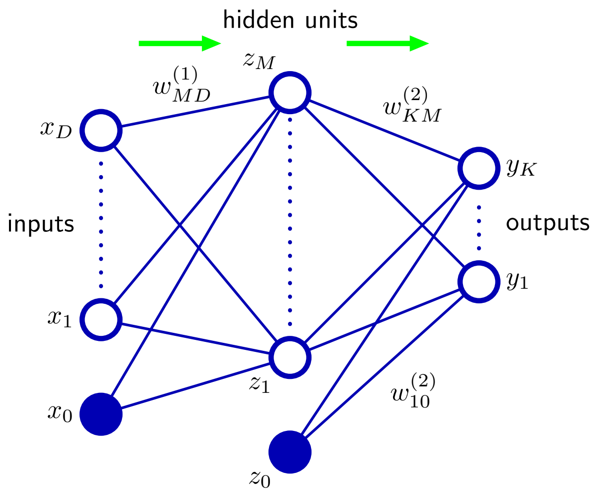

Figure 2: Architecture of a fully connected single hidden-layer feedforward neural network. The filled nodes x_0 and z_0 are bias terms.

It is important to note that the neural network has many discrete symmetries: there are a large number of permutations of the weights which leave the network invariant. This means that the objective function E is non-convex, and optimization via gradient descent methods is not guaranteed to find the global minimum. This is a limitation of neural networks when compared with SVMs, since the optimization involved in SVM is a quadratic program, and is therefore convex [15].

3 Design approach

3.1 License Plate Segmentation

We first take the RGB input image and convert it to greyscale by extracting only the green channel. We initially crop the edges[2], since we can be sure they don't contain any important information, and they typically present a large connected component (the plate border) which we want to be rid of for the following image analysis. We then binarize the resulting image, determining the threshold through Otsu's method[3]. We use a morphology-based approach. The general idea is to take advantage of the fact that we know *a priori* that the glyphs we are interested in will have certain properties:

- they collectively take up a significant portion of the license plate area,
- their upper and lower edges are colinear, and
- their aspect ratio is less than 1; that is, glyphs are taller than they are wide.

Using this knowledge, our segmentation algorithm uses a sequence of simple morphological operations to first generate 'candidate' cropped regions of the license plate, attempt to clean them up of extraneous details (such as bolts and rivets), and test them to see whether they fulfil the above requirements[4]. If a suitable number of candidates makes it through the testing process, we've succeeded, and we return the segmentation. If not, we try again, recursively calling the partition algorithm, by first inverting the colors of the plate, and then by progressively cropping the plate down, if we still don't find any candidate regions.

To generate the candidate regions, we initially take a similar approach to that used in the C-Lab1 morphology exercise, in which the goal is to segment lines of text from a textbook. We dilate the image horizontally by a wide structuring element, and crop the image to the vertical dimensions of the largest connected component that results. We then dilate vertically with a tall structuring element, and use connected component analysis[5] on the result to

[2]We find empirically that cropping 10% from the top and bottom, and 3% from the sides is effective.
[3]http://en.wikipedia.org/wiki/Otsu%27s_method.
[4]The MATLab function is found in `Code\image_partition.m`.
[5]We use MatLab's implementation `bwconncomp`, which uses the flood fill algorithm: http://en.wikipedia.org/wiki/Flood_fill.

segment the image into several candidate images. Typical plates will have between 10-20 candidates generated by this process. The vast majority are spurious candidates, and we eliminate them with numerous tests that follow.

Once we have a suitable number of candidate image regions, we progressively narrow them down by applying several tests to them. Let $\mathcal{C} = \{C_i\}_{i=1}^{N}$ be the set of candidates, and let $H(C_i)$ and $W(C_i)$ be the height and width of candidate region C_i respectively. Then for each candidate C_i we prune it from the list if it fails any of the following tests:

1. Is the aspect ratio R
$$R(C_i) = \frac{H(C_i)}{W(C_i)}$$
within the interval $[1, 10]$?

   We choose a lower bound of 1 by the empirical observation that all glyphs in Australian license plates appear to be taller than they are wide. The upper bound of 10 is purely to accomodate the glyphs '1' and 'I'; all other glyphs have aspect ratios roughly in the interval $[1, 2]$. This criterion is quite useful for trimming out parts of slogans or other artifacts of the morphological segmentation, since these often have aspect ratios outside this interval.

2. Is the mean intensity[6]
$$\bar{I}(C_i) = \frac{\sum_{j=1}^{H(C_i)} \sum_{k=1}^{W(C_i)} [C_i]_{jk}}{H(C_i)\, W(C_i)}$$
in the interval $[0.1, 0.9]$?

   Again, we find empirically that this is a useful heuristic for removing non-glyph segments. For example, the 'hyphen-like' separator often found between the first three and last three digits of the license plate is excluded by this test, since it is almost completely white.

3. Is the candidate's height within 10% of the mode of all passing candidates?
$$\frac{H(C_i) - \texttt{mode}(\mathcal{H})}{\texttt{mode}(\mathcal{H})} < 0.1,$$
where $\mathcal{H} = \{H(C_i) \mid C_i \in \mathcal{C},\ C_i \text{ passing}\}$ is the set of heights of passing candidates.

   This condition is quite stringent, and takes advantage of our prior knowledge that all the glyphs must be approximately the same height. It gets rid of plausible-looking candidate regions which contain, for example, state logos or other symbols or imagery.

The whole segmentation pipeline is laid out in Figure 3.1, with accompanying plate images to illustrate.

3.2 Neural Network training

We train a single hidden layer neural network on a dataset of over 36,000 images of computer fonts, taken from the Chars74K dataset[7]. We take as inputs 32×32 pixel square binary images, which correspond to input vectors of length 1024; The neural network architecture used is illustrated in Figure 3.2. We experimented with different feature representations, including integral representations such as the Radon transform[8], but found little to no improvement in performance. We use the 1-of-K encoding to represent our output vectors, which has 36 classes (26 alphabetical, and 10 numerical glyphs). We also use the cross entropy loss function

$$E(y, t) = -t \log y - (1 - t) \log(1 - y).$$

We use an l_2 regularizer on the weights, and let MATLab randomly divide the training set into training, test, and validation sets to avoid overfitting. We train for 76 iterations of scaled conjugate gradient descent[9], which takes 7 minutes on a 2.3 GHz i5 Macbook Pro with 16GB of memory. The mean cross-entropy error on the training set with the learned weights is 10^{-3}.

[6]Recall that these are binary images, so we are just counting the proportion of white pixels in the candidate region.
[7]This can be found at http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/.
[8]http://en.wikipedia.org/wiki/Radon_transform.
[9]http://en.wikipedia.org/wiki/Conjugate_gradient_method.

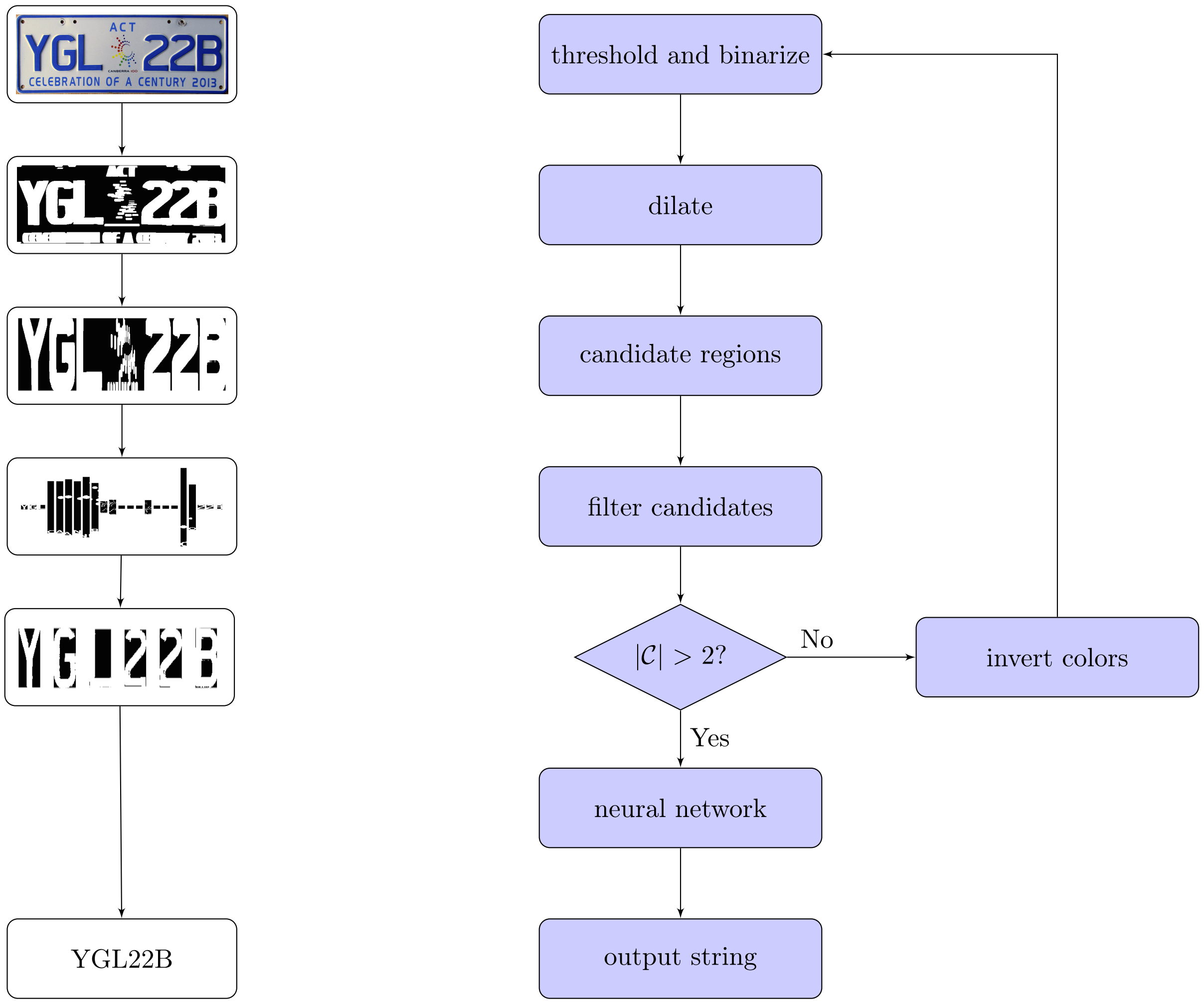

Figure 3: **Right:** License plate recognition pipeline. **Left:** Example of the morphological segmentation process.

4 Implementation and experimental results

The system is implemented with MATLab scripts and functions, which we include in the `Code` directory. The main segmentation work is done by `image_partition`, with calls to a helper function `tallestcomponent`. The neural network is trained beforehand, and is stored in a `.mat` file for ease of retrieval and re-use. The `classify` function performs the license plate segmentation and recognition. See Appendix A for details on how to run the system.

We tested the system on 70 cropped Australian license plate images taken from Wikipedia[10] and Plateshack[11]. Some sample plates with their segmentations are shown in Figure 4. On this test dataset, we achieved 94.9% glyph recognition accuracy, and 74.3% overall plate accuracy[12]. Note that the system makes no distinction between plates from different states; the same algorithm is used for segmentation and recognition for all license plates.

[10]http://en.wikipedia.org/wiki/Vehicle_registration_plates_of_Australia.

[11]http://www.plateshack.com/y2k/.

[12]Note that we take "O" to be equivalent to "0", and "1" to be equivalent to "I". This is common prctice, as in many US states, authorities do not distinguish between these glyphs.

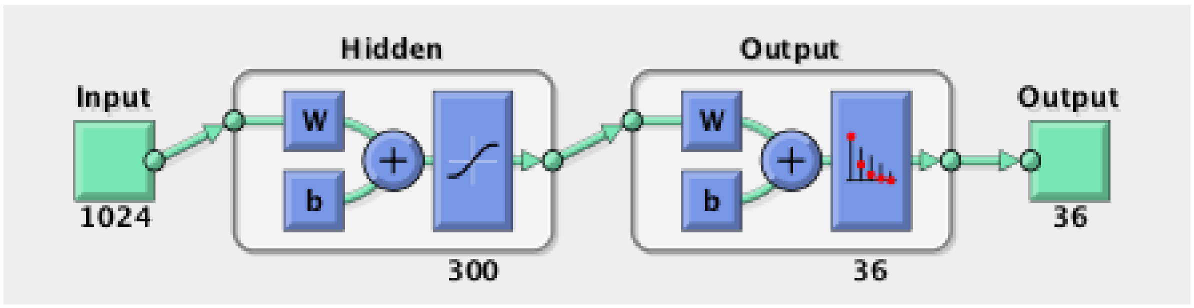

Figure 4: Architecture of our feedforward neural network.

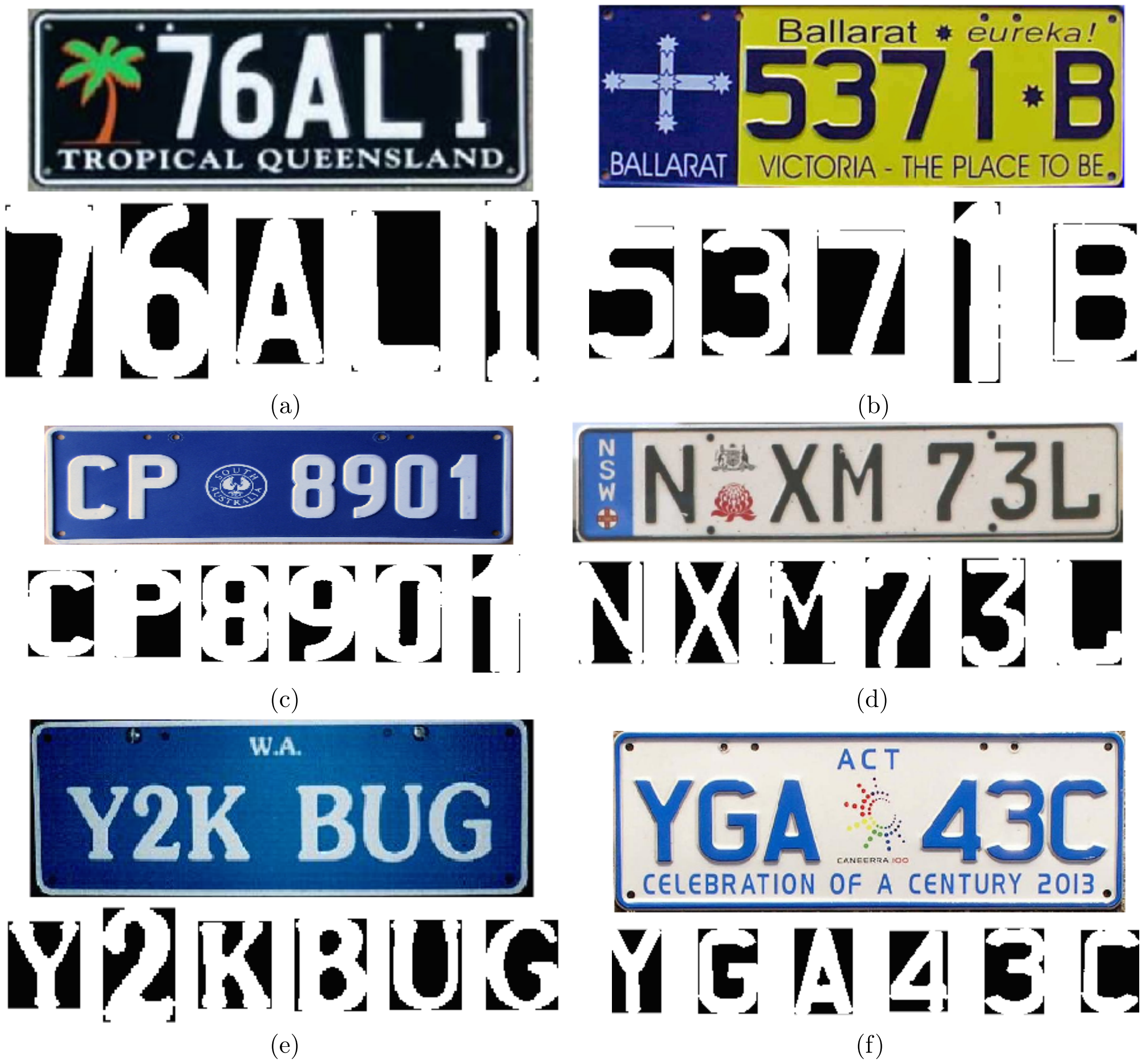

Figure 5: Six license plate examples, accompanied by their segmentations. Note that these plates present numerous challenges, including: large non-glyph graphics in **(a)** and **(b)**; uneven lighting and contrast in **(e)**; unusual glyph spacing in **(d)**. Note also the varying background and foreground colours, and the varying fonts used. These plates are all correctly segmented and recognised by the system.

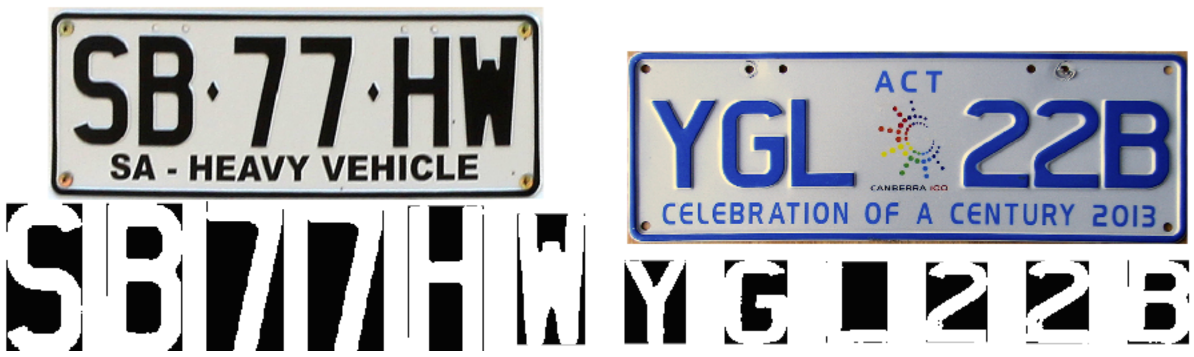

Figure 6: **Left:** The input plate is ‘SB77HW’, but the system classifies it as as ‘SB77HN’. This is symptomatic of the difficulties presented by the unusual, compressed fonts used in license plates. **Right:** The input plate is clearly ‘YGL22B’, but the system classifies it as ’Y6L22B’. This misclassification could possibly be improved by including extra features such as closed loop detectors, or computing how many disconnected regions there are.

5 Conclusions

Below we discuss the performance of the system, and propose numerous improvements. We also outline some of the learning outcomes from the project.

5.1 Limitations and possible extensions

The system has two main limitations: the plate segmentation algorithm assumes that we are already given a cropped license plate image. Adding a plate localization routine could address this issue, but it is outside the scope of the project. We’ll focus on our attention on the second issue, which relates to the performance of the neural network.

The single-glyph recognition rate of 94.9% achieved by the neural network is reasonably high, but compounding this over a mean plate length of six glyphs yields around the 75% plate accuracy that we achieved. If we require a 95% plate recognition accuracy, this requires an accuracy of $0.95^{1/6} \approx 99\%$ glyph accuracy. To achieve this level of accuracy in the neural network, there are a few possible approaches:

- Train the neural network on labelled license plates. This has the advantage of showing the neural network examples of the fonts used in the license plates, including the noise associated with the images. This should address the issue we see in Figure 5.1(a). The disadvantage is that this would require a large database of labelled Australian license plates images, which is not easily available, to the author’s knowledge.
- Use more advanced features such as line/loop feature detectors used in Cuneiform and Tesseract[16], or using SIFT [17]. We could augment the pixel representation with other global features, such as topological invariants[13], or other characteristics. Feature engineering such as this would probably help to disambiguate similar characters such as ’G’ and ’6’, which are commonly misclassified by my system (see Figure 5.1(b)).
- Use a state-of-the-art Convolutional Neural Network classifier, or a boosted classifier[14].

5.2 Learning outcomes

In the process of doing this project, I learned a number of things:

- I discovered and evaluated different solutions to the license plate recognition problem by performing a review of the international computer vision literature.
- I followed test-driven engineering principles, by implementing a test harness for the system, and subsequently making incremental changes to the system in an attempt to improve the test results.
- I trained a neural network on a large dataset and explored different feature representations.
- I experimented with numerous image processing techniques and became familiar with their implementations and uses.

[13]http://en.wikipedia.org/wiki/Topological_property.
[14]Using an algorithm such as AdaBoost.

A How to run the Matlab code

The code is stored in the `Code` directory; this directory will have to be added to your MATLab path. The main top level method is `classify`, which takes as inputs a neural net, and an RGB image. A neural net is saved in `Data/net.mat`. Load this into the main namespace with `load('Data/net.mat')`. Then, you can call `classify(im,net)`, where `im` is a $M \times N \times 3$ array of unsigned 8-bit integers, of the kind returned by `imread`. Alternatively, you can call the `demo()` function which will automatically run through the 70 plates included, showing the original, the segmentation, and printing the recognised output to the MATLab console.

References

[1] J.A.G. Nijhuis et al. Car license plate recognition with neural networks and fuzzy logic. *Proc. IEEE Int. Conf. Neural Networks*, 5:2232–2236, 1995.

[2] A. Gadicha S. Bailmare. A review paper on vehicle number plate recognition. *Int. Journal of Scientific and Research Publications*, 3(12), 2013.

[3] F. Ozturk & F. Ozen. A new license plate recognition system based on probabilistic neural networks. *Procedia Technology*, 1:124–128, 2012.

[4] Y. Zhang & C. Zhang. A new algorithm for character segmentation of license plates. *Proc. IEEE Intelligent Vehicles Symposium*, pages 106–109, 2003.

[5] H.A. Hegt et al. A high performance license plate recognition system. *Proc. IEEE Int. Conf. Systems, Man, and Cybenetics*, 5:4357–4362, 1998.

[6] L. Zheng & X. He. Character segmentation for license plate recognition by k-means algorithm. *Proc. Int. Conf. Ravenna*, pages 444–453, 2011.

[7] M-S Pan et al. Vehicle license plate character segmentation. *Int. Journal of Automation and Computing*, 5(4):425–432, 2008.

[8] V. Karthikeyan et al. Vehicle license plate character segmentation - a study. *Int. Journal of Computer and Electronics Research*, 2(1):47–54, 2013.

[9] A. Akoum et al. Two neural networks for license number plates recognition. *Journal of Theoretical and Applied Information Technology*, 12(1):25–32, 2009.

[10] Bhushan et al. License plate recognition system using neural networks and multithresholding technique. *Int. Journal of Computer Applications*, 84(5):45–50, 2013.

[11] L. Carrera et al. License plate detection using neural networks. *Lecture Notes in Computer Science*, 5518:1248–1255, 2009.

[12] D. ZhengHao & FengXin. License plate recognition based on support vector machines. *Journal of Multimedia*, 9(2):253–260, 2014.

[13] Yann LeCun et al. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[14] C.M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[15] C. Cortes & V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[16] R. Smith. An overview of the tesseract ocr engine. *Proc. Int. Conf. Document Analysis and Recognition*, 2007.

[17] A. Kae et al. Improving state-of-the-art ocr through high precision document-specific modelling. *Computer Vision and Pattern Recognition*, 2010.