LICENSE PLATE RECOGNITION USING CONVOLUTIONAL-RECURRENT NEURAL NETWORK

TEE KAI FENG

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Science (Hons.) Actuarial Science

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2019

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : Tee Kai Feng

ID No. : 1504044

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"LICENSE PLATE RECOGNITION USING CONVOLUTIONAL-RECURRENT NEURAL NETWORK"** was prepared by **TEE KAI FENG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Hons.) Actuarial Science at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Dr Liew How Hui

Date : _____

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my deep gratitude to my research supervisors, Dr. Liew How Hui and Dr. Tay Yong Haur for their patient guidance, invaluable advice, and useful critiques throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement.

LICENSE PLATE RECOGNITION USING CONVOLUTIONAL-RECURRENT NEURAL NETWORK

ABSTRACT

License plate recognition is a technology that utilises computer vision techniques for identifying vehicles by their license plates. This technology is commonly used for traffic law enforcement, automatic toll collection and ticketless parking system. In this paper, a lightweight, segmentation-free approach for license plate recognition is proposed. This approach is inspired by the recent trend in deep learning research community to combine both Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) for sequence recognition task. The proposed Convolutional-Recurrent Neural Network (CRNN) contains a CNN without fully connected layers, which is used for extracting features from license plate image and these extracted features are passed to RNN for sequence recognition. This approach combines both CNN and RNN in a unified framework, which allows it to be end-to-end trainable. To solve the problem of two-row license plate in Malaysia, a text detection algorithm is integrated into our framework to detect the location of text in license plate image. As a result, the proposed CRNN is suitable for real-world deployment in Malaysia due to its lightweight architecture and state-of-the-art recognition accuracy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

ALPR	Automatic License Plate Recognition
CCTV	Closed-circuit Television
CNN	Convolutional Neural Network
CRNN	Convolutional-Recurrent Neural Network
CTC	Connectionist Temporal Classification
EAST	Efficient and Accurate Scene Text Detector
ER	Extremal Region
RNN	Recurrent Neural Network
HDRBM	Hybrid Discriminative Restricted Boltzmann Machines
LSTM	Long short-term memory
MSER	Maximally Stable Extremal Regions
OCR	Optical Character Recognition
ReLU	Rectified Linear Unit
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SWT	Stroke Width Transform

CHAPTER 1

INTRODUCTION

1.1 Background

Industry 4.0 has been a hot topic over the last several years. It is a name given for the current trend towards interconnectivity and advanced automation that aims to revolutionize different sectors.

Many breakthroughs in machine vision can be observed over the past few years due to the rise in artificial intelligence technologies. One of the implementations of machine vision is automatic license plate recognition (ALPR), which is a system that utilise machine vision techniques to read vehicle license plate images that are usually captured by closed-circuit television (CCTV) or dashcam. This technology aims to automate the process of monitoring vehicles on the road, which meets the requirements of promoting automation in Industry 4.0. A typical ALPR system takes two steps to read the license plate, which starts with detecting the location of license plate in a given image, follows by recognising the contents in the detected license plate region.

Beside monitoring vehicles on the road, ALPR system has many other applications. For example, ALPR system can be used to trace the entry and exit of vehicles into a premise automatically, and with that issuance of parking ticket can be avoided. This can save a lot of time and help in reducing traffic congestion. Other than that, ALPR system can also help users to locate cars in parking lots and detect stolen cars. Government organisations can also use ALPR as a tool for mass surveillance by tracking vehicle movements across the country.

1.2 Problem Statement

The main difficulty with creating a sophisticated ALPR system is that the algorithm needs to cope with a wide range of image qualities. Vehicle images captured by CCTV or dashcam usually face the following problems:

i. Low resolution (normally because the license plate is too far away from the camera)
ii. Blur (due to fog or smog)

iii. Motion blur
iv. Overexposed / Underexposed (due to lightning condition)
v. Varied viewing angle
vi. Shadow

Besides, it is also difficult for computer algorithms to differentiate ambiguous English characters such as “O (capital o)”, “0 (zero)”, “I (capital i)” and “1 (one)”, which are commonly found in license plates.

Many existing algorithms, especially those who apply traditional computer vision techniques tend to work well only under controlled conditions. In other words, there are hardly any algorithms that can perform well under open environment, which subject to all the difficulties and problems in ALPR mentioned above.

Traditional computer vision methods typically handle ALPR problem by further breaking it down into 2 parts, character segmentation and character recognition. The major drawback of this method is that character segmentation is widely regarded as a complicated task and generally do not work well under open environment. This is because most character segmentation algorithms require image pre-processing, which requires the programmer to make assumptions and setting thresholds, especially during the process of determining the segmentation edges for individual characters. Although there are abundant character recognition algorithms that can achieve state-of-the-art results, a character segmentation algorithm with poor performance can easily affect the overall performance since the final result depends on the interplay of both character segmentation and character recognition algorithms.

Hence, there is a need to develop a segmentation-free approach to address the ALPR problem.

1.3 Aims and Objective

Due to the limitation of traditional computer vision methods, modern segmentation-free approaches are usually favoured by researchers. Soo (2017), for example, uses a segmentation approaches called convolutional-recurrent neural network (CRNN) to build an ALPR system. The CRNN model proposed by Soo (2017) consists of a pre-trained VGG19 CNN layer and uses a single layer long short-term memory (LSTM) network for sequence labelling. When training the CRNN model, only parameters of

LSTM model are being trained while the parameters of VGG19 is fixed. There are a few disadvantages and limitations in this type of neural network design.

First, it is not end-to-end trainable. Since the weights of CNN layers are fixed and cannot be trained, we are relying on the performance of the pre-trained CNN model done by other researchers for features extracting. If the pre-trained CNN model happened to be performing poorly on our training sets, the accuracy of the final prediction will be affected greatly.

Second, using large pre-trained CNN architectures in a CRNN model is just overkill. Most pre-trained CNN architectures are trained on ImageNet datasets and are designed to recognise the features of daily life objects. These architectures usually consist of very deep layers to capture large number of features existing in daily life objects. The features of a license plate that need to be captured, however, are comparatively much lesser than daily life objects. Theoretically, pre-trained CNN architectures can definitely extract the features from license plate, though, may deemed superfluous. However, using pre-trained CNN comes at a cost. Since these architectures are generally very deep, the numbers of parameters are very high as well. Large number of parameters will increase the predicting time and might cause the ALPR software to be impractical in real-world application.

Therefore, the main objective of this study is to address the limitations of the CRNN proposed by Soo (2017) and come out with another variant of CRNN that can overcome these limitations. This study aimed to build an ALPR model that has the following capabilities:

i. End-to-end trainable
ii. Small architecture
iii. Short execution time
iv. High prediction accuracy
v. Able to recognise both single-row and two-row license plate

1.4 Scope

In this paper, we will not cover license plate localisation algorithms but only focus on the license plate recognition stage, which is to read from an image of localised license plate. We propose a segmentation free approach that incorporates the usage of an end-to-end trainable neural network architecture. This end-to-end trainable neural network architecture is called Convolutional Recurrent Neural Network (CRNN), which combines both Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN).

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

Methods used for ALPR can be roughly categorised into 2 groups, particularly segmentation approaches or segmentation-free approaches. Segmentation approaches typically involve 2 steps, which is character segmentation, followed by character recognition. Segmentation-free approaches, on the other hand, do not apply any character segmentation algorithms.

For countries having two-row license plate, such as Malaysia, the ALPR system usually needs text detection algorithms to detect the location of texts in a two-row license plate.

2.2 Segmentation Approach

Previous work on ALPR typically requires two steps. The characters in license plate need to be segmented first and then recognised using optical character recognition (OCR) techniques. The accuracy of character segmentation plays an important role in this type of ALPR system because an improper segmentation will definitely lead to an incorrect labelling of individual characters. However, character segmentation process is considered by many to be a challenging task and had been heavily researched in the past.

2.2.1 Character Segmentation

Researchers had come out with many different approaches for character segmentation. For example, Turkyilmaz and Kacan (2017) use a method called "vertical projection method". This method requires image pre-processing which involve a threshold operation that converts each pixels of an image to either white or black pixel, followed by mean filtering before the segmentation process. Figure 2.1 illustrates how the threshold operation converts the image into black and white pixels. Besides, Extremal Region (ER) used by Gou, Wang, Yao and Li (2016) is another commonly used method for character segmentation. Maximally Stable Extremal Region (MSER), an extension of ER method had also been previously used by

researchers for image segmentation, such as Hsu, Chen, and Chung (2013) and proven to be more effective than ER.

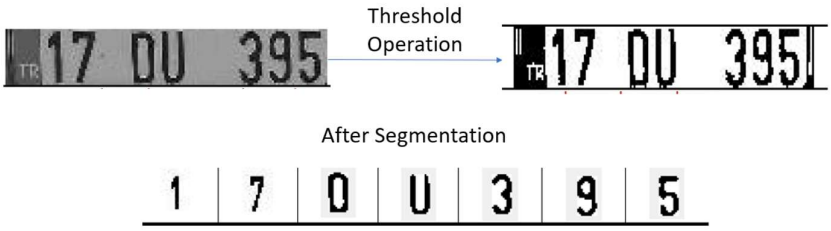


Figure 2.1: Threshold Operation

Bu and Xie (2013) proposed another character segmentation method called "sliding window search". This approach uses a small window to slide over license plate image to capture many tentative characters repeatedly and the contents of these windows are recognised using OCR. The final output is computed by choosing the characters with highest count consecutively. This method is able to produce an accuracy level of 98.92% for character level recognition. The process of sliding window search is illustrated in Figure 2.2.

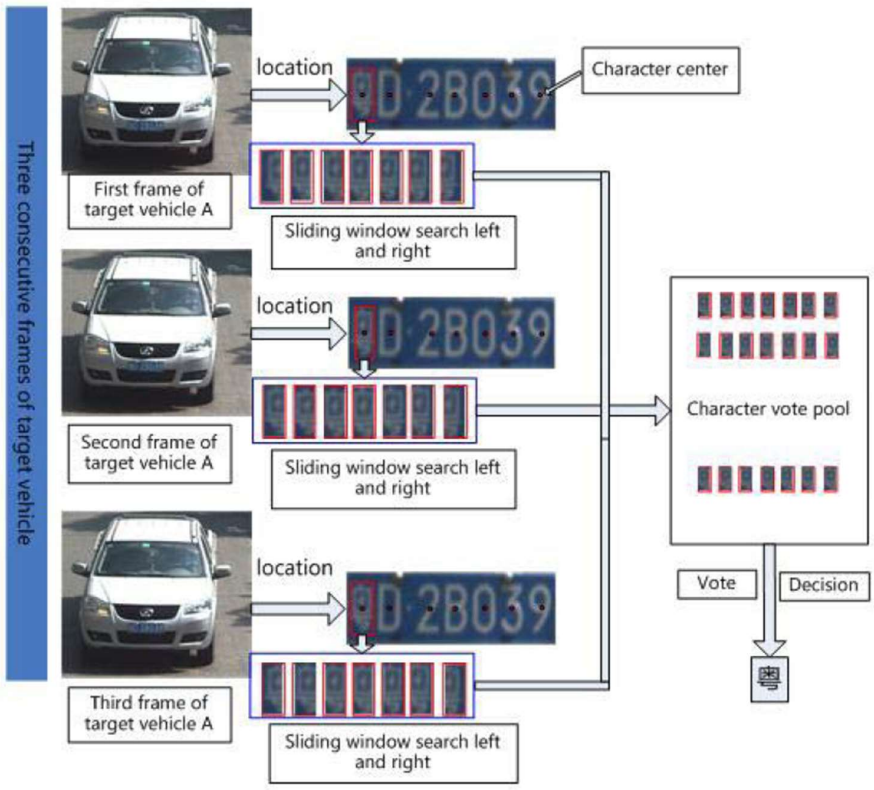


Figure 2.2: Sliding Window Search

The Inception-RPN framework proposed by Zhong, Zhang and Feng (2016) is similar to sliding window search method but showed stronger robustness. The Inception-RPN framework is a neural network architecture that combined both Inception Net and Region Proposal Network (RPN). This type pf framework slides an InceptionNet with multi-scale windows over the top of feature maps extracted by convolutional layers and associates a set of text characteristic prior bounding boxes with each sliding position to generate word region proposals. Experimental results showed that this framework achieved state-of-the-art performance in text detection.

2.2.2 Character Recognition

Character recognition is another important component of ALPR but is considered by many to be a less challenging task. Many different approaches used for character recognition had achieved state-of-the-art results. For example, Turkyilmaz and Kacan (2017) integrated CNN into their framework for character recognition and achieved 96.92% accuracy for character level recognition. Meanwhile, Hsu, Chen, and Chung (2013) use a probabilistic classifier for character recognition called Hybrid Discriminative Restricted Boltzmann Machines (HDRBM). With the combination of MSER for character segmentation and HDRBM for character recognition, they achieved 98.2% for character level recognition accuracy. Template matching method is also applicable for character recognition. For example, Gillyand and Raimond (2013) applied thresholding techniques for character segmentation and utilise template matching for character recognition. Template matching method compares the portions of input image against the template images. The input image is then classified by choosing the label of the template images that has the highest correlation with the input image. Although this method is easy to implement, the results are not very satisfactory. Any font change, rotation or noise in the input image can easily lead to errors.

2.3 Segmentation-free Approach

ALPR that utilises character segmentation and character recognition face the same limitation. All input images for this kind of ALPR system requires pre-processing to facilitate the segmentation process because segmentation process by itself is a complicated process. Therefore, to avoid dealing with the hassles of character

segmentation, segmentation-free approaches are more preferred. Shi, Bai and Yao (2015) proposed a novel neural network architecture named Convolutional Recurrent Neural Network (CRNN), which combines both CNN and RNN. The CNN layers are used to extract features from the input images. The extracted feature maps will then be passed into RNN layers for sequence labelling. Experimental results showed that this kind of architecture is efficient for sequence labelling process, including ALPR. For instance, Choi, Fazekas and Sandler (2016) applied CRNN for music classification while Bartz et al. (2017) utilised CRNN for language identification. Besides, Soo (2017) also used CRNN architecture for his ALPR model. Figure 2.3 shows the CRNN architecture used by Soo (2017).

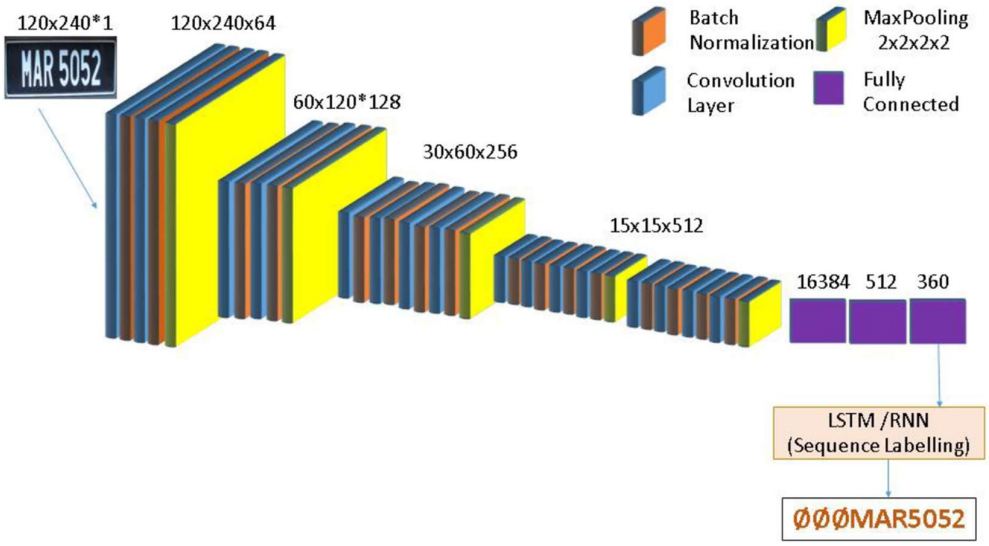


Figure 2.3: Soo's (2017) CRNN Architecture (VGG19 + LSTM)

2.4 Text Detection Algorithms

Some ALPR system has a text detector model embedded in it in order to be able to read multiple-row license plates. Traditional text detection approaches can be roughly classified into three groups, mainly:

i. Region-based approaches, which gather pixels based on the similarity of characteristics of text, such as size, stroke width and edges
ii. Texture-based approaches, which find and merge candidate text regions based of textural properties

iii. Hybrid approaches, which utilise the advantage of region-based approaches and texture-based approaches

There are two important steps involved in a text detection algorithm, which is finding letter candidates and grouping these letter candidates into regions of text.

Epshtein, Ofek and Wexler (2010), for example, made use of Stroke Width Transform (SWT) to find and filter letter candidate and utilised connected component algorithm for grouping these letter candidates into text regions. SWT is a local image operator that computes per pixel the width of the most likely stroke containing the pixel. The flowchart of Epshtein, Ofek and Wexler's (2010) text detection algorithm is shown in the figure below.

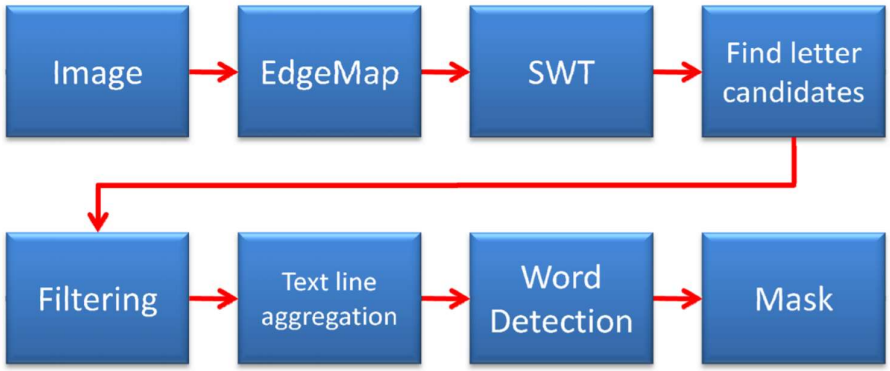


Figure 2.4: Text Detection Flowchart

González et al. (2012), on the other hand, used MSER to find letter candidates but still utilised SWT to filter out these candidates. The similar approach was also adopted by Li and Lu (2012), which applied both MSER and SWT when extracting letter candidates from input image.

Though traditional approaches in text detection had demonstrated promising results across various benchmarks, they usually fall short when dealing with challenging scenarios. The main reason for this occurrence is because the overall performance of these traditional approaches typically relied on the interplay of multiple stages and components in the pipelines (such as extracting, filtering and grouping letter candidates). In order to avoid having stages in the text detection pipeline, Zhou et al. (2017) proposed a single neural network that directly predicts

text lines of arbitrary orientations and quadrilateral shapes without intermediate steps. The model proposed by them is named EAST (An Efficient and Accurate Scene Text Detector).

There are also some other text detection algorithms, which had been summarised by Shanbhag, Thakkar and Patel (2015). The details of these algorithms can be found in the table below. However, do note that all these methods are not robust because they only work well under certain controlled environment.

Table 2.1: A survey on various approaches of Text Detection Algorithms

No	Author	Methods used	Accuracy	Benefits
1	Kohei et al.	Edge-based method, connected component labeling method, Morphology erosion filter, Comic text extraction method	94.66%	Works on complex background.
2	Shivakumara et al.	Maximum Color Difference (MCD), Boundary Growing Method (BGM), K-means clustering algorithm	89.67%	Insensitive to contrast
3	Luz et al.	Morphological filters, Decision tree classifier	85.93%	Insensitive to position
4	Shyama et al.	Colour based segmentation, Light edge enhancement, Heavy edge enhancement	94%	Insensitive to size, orientation
5	Fabrizio et al.	Multiple SVM classifiers, Toggle Mapping Morphological Segmentation	88.83%	Insensitive to lighting, orientations
6	Pan et al.	Local binarization approach, Conditional Random Field (CRF) model, Minimum classification error (MCE) learning, Graph cuts inference algorithm, Minimum spanning tree (MST), Energy minimization model	83.44%	Robust and accurately localize texts
7	Angadi et al.	DCT based high pass filter, Discriminant functions	96.60%	Handles different type of size, alignment, nonlinear text region

CHAPTER 3

METHODOLOGY

3.1 Overview

In order to be able to recognise both single-row and two-row license plate, we proposed a pipeline that consists of 2 stages. The first stage is text detection using a model named EAST, which was proposed by Zhou et al. (2017) and the second stage is text recognition using CRNN model. The following diagram gives an overview on how single-row and two-row license plate are being processed in the proposed model.

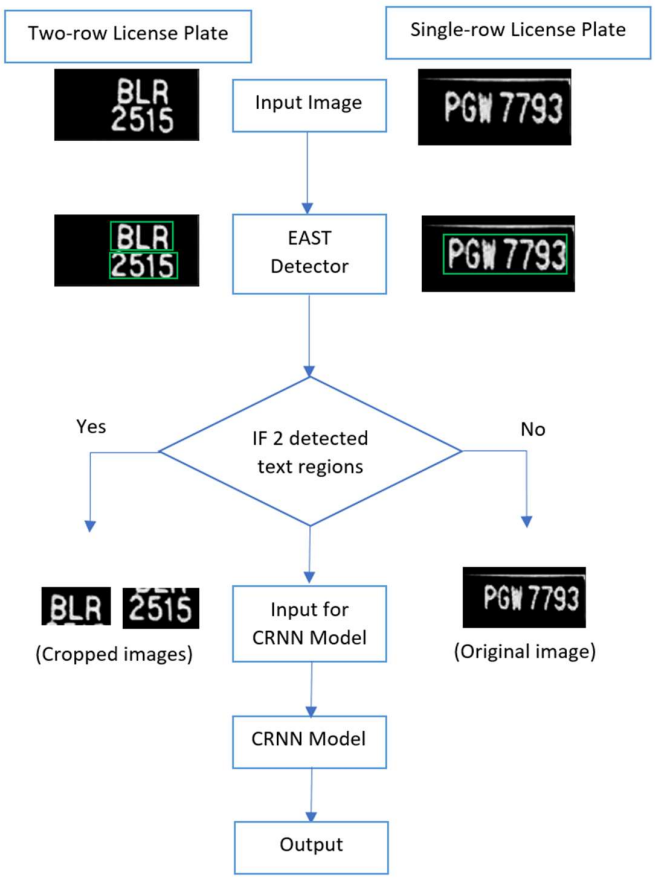


Figure 3.1: Proposed ALPR Pipeline

3.2 Data Collection

Our training datasets contain 25720 localised license plate images cropped from images captured by ALPR camera deployed in a real-world setting. The dimension of these images is fixed at 240px (width) x 120px (height) so that the neural network

model is easier to parallelize at training time. Besides, our training data sets only consists of single line license plate. Samples of our training dataset are shown below:

Figure 3.2: Train Dataset

The quality of the training datasets varies. Some images are blurred, noisy while others are sharp and clear.

For benchmarking purpose, we use 3 different test datasets, which we will name them as LPR44, LPR45 and Open Environment Datasets. The image quality deteriorates across these datasets, with LPR44 having the least noise level and Open Environment Datasets having the most noise in images, such as blurring and shadows.

Samples from LPR44, LPR45 and Open Environment Datasets are shown in the figure below.

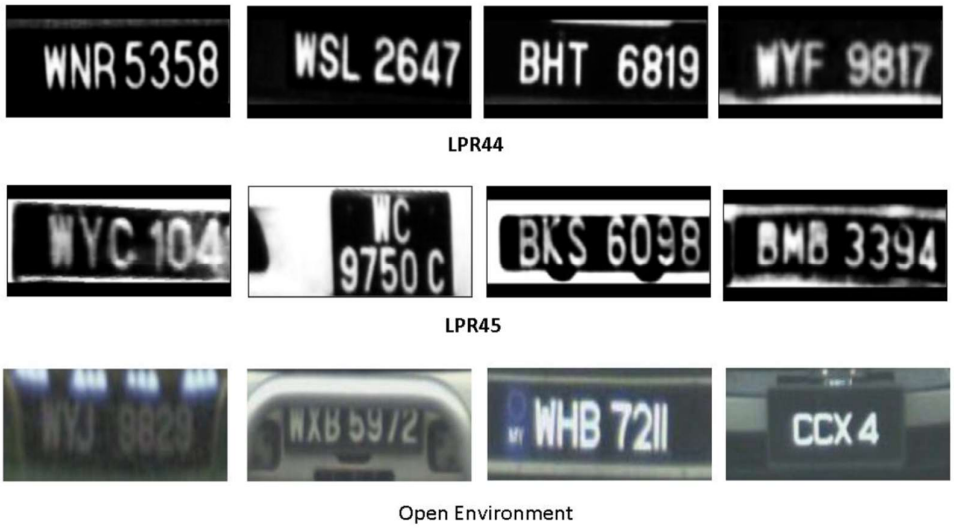


Figure 3.3: Test Datasets

3.3 Efficient and Accuracy Scene Text (EAST) Detector

EAST is a text detector with robust performance. It is a version of the U-Net, which is well known for its performance in extracting features that varies in size. Figure 3.4 shows the structure of EAST network.

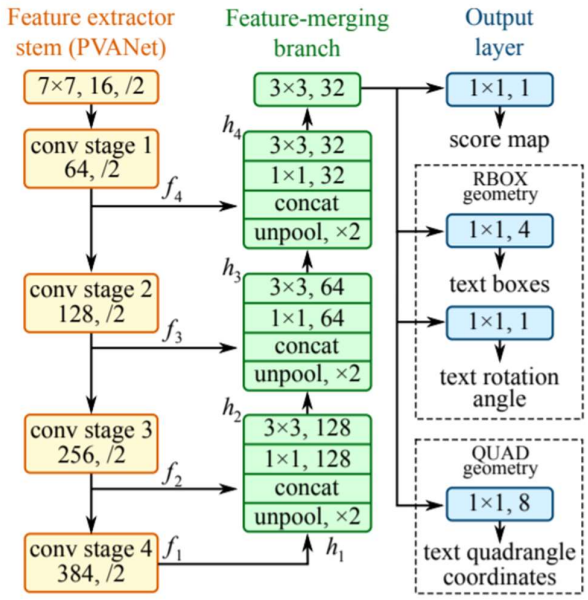


Figure 3.4: Structure of EAST network

The original implementation of this network output two types of bounding boxes, which is standard rectangular bounding box with rotation angle or a quadrangle with all 4 coordinates of the vertices. However, in our implementation of EAST detector, we only output rectangular bounding boxes without rotation angle for simplicity. Besides, we did not retrain the EAST network with our own datasets since our train dataset was not labelled with the coordinates of bounding boxes containing license plate text. Instead, we use the weights of pre-trained EAST model, which had been trained on ICDAR 2013 and ICDAR2015 with much more sophisticated computational resources.

3.4 Convolutional-Recurrent Neural Network (CRNN) Architecture

Since the features that need to be extracted from license plate images is very limited, we propose a simple CNN architecture for feature extraction in our CRNN model. This CNN architecture was inspired by Shi, Bai and Yao's (2015) work and is shown below:

Table 3.1: Architecture of Proposed CRNN

Layer Name	**Output size (channels x height x width)**	**Parameters**
Input	1 x 32 x 100	-
Conv1	64 x 32 x 100	#filters: 64, k=3x3, s=1, p=1
Maxpool1	64 x 16 x 50	k=2, s=2, p=0
Conv2	128 x 16 x 50	#filters: 128, k=3x3, s=1, p=1
Maxpool2	128 x 8 x 25	k=2, s=2, p=0
Conv3	256 x 8 x 25	#filters: 256, k=3x3, s=1, p=1
BatchNorm	-	-
Conv4	256 x 8 x 25	#filters: 256, k=3x3, s=1, p=1
Maxpool3	256 x 4 x 26	k=2, s=2x1, p=0x1
Conv5	512 x 4 x 26	#filters:512, k=3x3, s=1, p=1
BatchNorm	-	-
Conv6	512 x 4 x 26	#filters: 512, k=3x3, s=1, p=1
Maxpool4	512 x 2 x 27	k=2, s=2x1, p=0x1
Conv7	512 x 1 x 26	#filters: 512, k=2x2, s=1, p=0
BatchNorm	-	-
Bidirectional-LSTM		#hidden unit: 256
Bidirectional-LSTM		#hidden unit: 256
Transcription	-	-

Note that kernel size, stride and paddings of a convolutional layers are denoted by k, s, and p respectively in Table 3.

There are 7 layers of convolutional layers in this architecture. All the convolutional layers, except Conv7 has stride 1 convolution with kernel size of 3 and 1 padding on both sides of the images. This configuration ensures that the width and height of the feature maps remain the same after each convolutional layer. Rectified Linear Units (ReLU) is added after each convolutional layer to perform element-wise activation. ReLU function can be written in the follow way:

$$f(x) = \max(0, x) \tag{3.1}$$

where x is the activation of the feature maps. Dimension reduction of feature maps is solely done by MaxPooling layers. There is a total of 4 MaxPooling layers in our proposed architecture. Both MaxPool1 and MaxPool2 will reduce the feature maps' width and height by half while MaxPool3 and MaxPool4 will only reduce the height of feature maps by half while maintaining the dimension of the width.

BatchNorm layers are also added throughout the architecture to accelerate the training process. BatchNorm layers normalise the activations to ease the training process. The following are the parameters in a BatchNorm layer:

$$\mu_{\mathrm{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{3.2}$$

$$\sigma_{\mathrm{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathrm{B}})^2 \tag{3.3}$$

$$\widehat{x_i} = \frac{x_i - \mu_{\mathrm{B}}}{\sqrt{(\sigma_{\mathrm{B}}^2 + \epsilon)}} \tag{3.4}$$

$$BN_{\gamma,\beta} = y_i = \gamma\hat{x}_i + \beta \tag{3.5}$$

where μ_{B} is the mini-batch mean, σ_{B}^2 is the mini-batch variance, $\widehat{x_i}$ is the normalised activation, and y_i is the output value after scale and shift. Note that γ, β are the learnable parameters in a BatchNorm layer.

For the recurrent layers, LSTM is used instead of normal RNN to prevent exploding gradient problem. Instead of using one layer of LSTM, we stacked two bi-directional LSTM together to capture more details and to increase the accuracy of per-frame predictions. LSTM is just a variant of RNN with more complex units for activation. The memory cell of LSTM at single time-step, t is illustrated in the figure below:

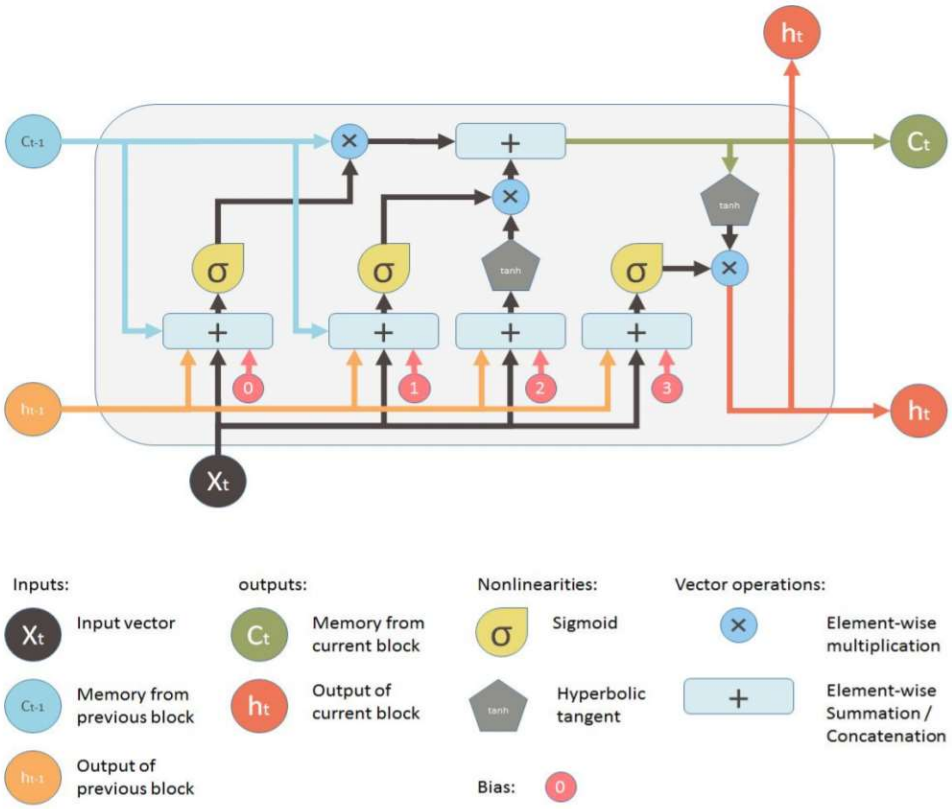


Figure 3.5: Memory cell of LSTM at each time-step

There are five important components in a typical LSTM memory cell, particularly the forget gate, f_t , input gate, i_t , candidate values, $\widetilde{C_t}$, cell state, C_t and output state, o_t .

In order to avoid the vanishing gradient problem in traditional RNN, we included a forget gate that looks at previous hidden state, h_{t-1} and input vector, x_t , which then output a value between 0 and 1 (because of sigmoid function, σ). This output value suggests how much we need to "forget" about the previous state and it can be calculated using the formula below:

$$f_t = \sigma\left(W_f \,.\, [h_{t-1}, x_t] + b_f\right) \tag{3.6}$$

Next, we need to determine how much new information need to be stored in the cell state and this will be decided by the value of input gate, i_t , which can be computed from:

$$i_t = \sigma(W_i \,.\, [h_{t-1}, x_t] + b_i) \tag{3.7}$$

Candidate value, $\widetilde{C_t}$ are calculated by passing it into $tanh$ activation function, which maps the resulting values into the range of -1 to 1.

$$\widetilde{C_t} = \tanh(W_C \,.\, [h_{t-1}, x_t] + b_C) \tag{3.8}$$

In order to find the current cell state, C_t , we sum previous cell state and current candidate value up, weighted by the forget gate and input gate respectively. This is to adjust the amount of information we want to "forget" and "remember" from past and present states.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{3.9}$$

The last step is to calculate the output gate, o_t which conditionally decides what to output from the memory cell.

$$o_t = \sigma(W_o \, [h_{t-1}, x_t] + b_o) \tag{3.10}$$

Finally, we output the value of current hidden state, h_t by multiplying output gate and cell state activated by $tanh$ function.

$$h_t = o_t * \tanh(C_t) \tag{3.11}$$

Transcription layer is added at the end to convert the per-frame predictions made by LSTM into a label sequence.

Note that there are only around 5 million trainable parameters for the convolutional layers in our proposed architecture, which is much lesser than the VGG19 network used in Soo's (2017) CRNN model, which has around 2 billion parameters. The main reason for the huge difference in number of parameters is because the convolutional layers in our proposed CRNN abandons the usage of fully connected layers that typically have large number of parameters. The reduction in number of trainable weights is an added advantage because it can speed up the training time and reduce the prediction time, which is an important aspect in real-world application. Figure 3.5 shows the number of trainable parameters in each convolutional layer of CRNN.

Proposed CRNN Architecture Size

Layers	Number of parameters
conv3-64	3 x 3 x 1 x 64 = 576
conv3-128	3 x 3 x 64 x 128 = 73728
conv3-256	3 x 3 x 128 x 256 = 294912
conv3-256	3 x 3 x 256 x 256 = 589824
conv3-512	3 x 3 x 256 x 512 = 1179648
conv3-512	3 x 3 x 512 x 512 = 2359296
conv2-512	2 x 2 x 512 x 512 = 1048576
SUM	5,546,560

Total number of parameters: 5,546,560

Soo's (2017) CRNN Architecture Size

Layers	Number of parameters
conv3-64	3 x 3 x 1 x 64 = 576
conv3-64	3 x 3 x 64 x 64 = 36864
conv3-128	3 x 3 x 64 x 128 = 73728
conv3-128	3 x 3 x 128 x 128 = 147456
conv3-256	3 x 3 x 128 x 256 = 294912
conv3-256	3 x 3 x 256 x 256 = 589824
conv3-256	3 x 3 x 256 x 256 = 589824
conv3-256	3 x 3 x 256 x 256 = 589824
conv3-512	3 x 3 x 256 x 512 = 1179648
conv3-512	3 x 3 x 512 x 512 = 2359296
conv3-512	3 x 3 x 512 x 512 = 2359296
conv3-512	3 x 3 x 512 x 512 = 2359296
FC1	(15x15x512) x 16384 = 1887436800
FC2	16384 x 512 = 8388608
FC3	512 x 360 = 184320
SUM	1,906,589,972

Total number of parameters: 1,906,589,972

Figure 3.6: Comparison of Architecture Size between Proposed CRNN and Soo's (2017) CRNN

3.5 Loss Function for Training CRNN

Loss function is one of the main components in neural networks. The purpose of loss function is to evaluate the inconsistency between predicted and actual values. By taking the derivatives of the loss function with respect to trainable weights, we can then determine the changes needed on the trainable weights in order to reduce the loss function. This process is called "backpropagation".

For this particular neural network, we utilised a special loss function called Connectionist Temporal Classification (CTC) loss that is first proposed by Graves, Fernandez, Gomez, and Schmidhuber (2006). Under this loss function, the output of a network is treated as conditional probability distribution over all possible label sequence. An objective function can be derived from this conditional probability to directly maximise the probabilities of the correct labelling. Backpropagation through time is possible because the objective function is differentiable.

According to Graves, Fernandez, Gomez and Schmidhuber (2006), a CTC network need to use softmax function as output layer and the number of unit in this softmax layer should be 1 unit more than L, where L is a set containing all possible labels in framewise prediction, that is {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, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. The activations of the first L units represent the probabilities of observing the corresponding labels while the extra unit represents the probabilities of observing a "blank" at particular times. The probabilities of a given label sequence can then be calculated by multiplying the probabilities of getting individual characters in the sequence.

To put all these in a more formal way, let us convert the concepts stated above into a more mathematical way. Assume that we have an input sequence x of length T, a recurrent neural network with m inputs, n outputs and weight vector w can be defined as a continuous map $N_w: (\mathbb{R}^m)^T \rightarrow (\mathbb{R}^n)^T$. Let $y = N_w(x)$ be the sequence of network outputs, and denote by y_k^t the activation of output unit k at time t. Then y_k^t can be treated as the chances of observing label k at time t, which define a distribution over the set L'^T of length T sequence over the alphabet $L' = L \cup \{blank\}$:

$$p(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t \ , \forall \pi \in L'^T \tag{3.12}$$

where π in equation (3.12) is referring to paths, which is the elements of L'^T. Note that we have implicitly assumed that the outputs of the network at different time are conditionally independent.

Next, we define a many-to-one mapping, $B : L'^T \rightarrow L^{\leq T}$, where $L^{\leq T}$ is the set of possible labellings. This set of possible labelling are computed by removing all blanks and repeated labels from the path. For example, B maps "--aa-p-p-ll-e--" ('-' represents a "blank") onto "apple". Lastly, we use B to define the conditional probability of a given labelling $l \in L^{\leq T}$ as the sum of the probabilities of all the paths corresponding to it:

$$p(l|x) = \sum_{\pi \in B^{-1}(l)} p(\pi|x) \tag{3.13}$$

Directly computing equation (3.13) would be computationally infeasible due to the exponentially large number of summation items. Therefore, we adopted a faster

parallel implementation of CTC, named "Warp CTC", developed by Amodei et al. (2015).

If we assume that the path with highest probability corresponds to the most probable labelling, then $h(x)$ can be approximated with the following equation:

$$h(x) \approx B(\pi^*), \qquad where\ \pi^* = \arg\max_{\pi \in N^t} p(\pi|x) \tag{3.14}$$

When training a network, we want to minimize the negative log-likelihood of conditional probabilities. Let us denote the training dataset by $\chi = \{I_i, l_i\}$, where I_i is the training image and l_i is the actual label of the license plate image. Then, the negative log-likelihood of conditional probability can be written as:

$$O = -\sum_{I_i, l_i \in \chi} \log p(l_i | x_i) \tag{3.15}$$

where x_i is the sequence produced by the CRNN model from I_i. This objective function calculates a cost value directly from an image and its actual label. This allows our model to be end-to-end trainable on pairs of images and labels, which cannot be done using other loss functions.

3.6 Evaluation Criteria

3.6.1 Accuracy

The accuracy of prediction is calculated using the simple equation below:

$$Accuracy = \frac{\#\ of\ correctly\ labelled\ license\ plate}{Total\ number\ of\ license\ plate}\ x\ 100\%$$

A license plate can only be considered correctly labelled if all individual characters in the license plate are predicted correctly.

Although the equation above is able to reflect the actual performance of ALPR system, we are unable to evaluate how well is the system performing in character level recognition. So, we will calculate the character level recognition accuracy using the following equation:

$$Character\ level\ accuracy = \frac{\#\ of\ correctly\ labelled\ character}{Sum\ of\ \#\ of\ characters\ in\ all\ license\ plates}\ x\ 100\%$$

3.6.2 Levenshtein distance

We also introduce another indicator for error analysis called Levenshtein distance, sometimes known as edit distance to measure the difference between the actual and predicted label of license plate. Levenshtein distance measures the minimum number of single-characters edits (insertion, deletion or substitution) required to convert one string to another. For example, if our actual license plate is "WWB1327" while the predicted license plate is "WVB1327", then the Levenshtein distance is 1 because we only need to substitute "V" in predicted license plate to "W" to make it become "WWB1327".

The formula for Levenshtein distance is given as follow:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

$|b|$ respectively. $lev_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b.

3.7 Computation of Prediction Confidence

It is important to have an indicator to reflect the confidence level of a prediction. This is to alert us for any potential prediction mistakes so that we can take necessary actions to reduce the impact from these errors.

Recall that at each time-step, the last LSTM layer in our model will output a hidden state, h_t , which is essentially just a vector of 36 real numbers (the total number of all possible labels in framewise prediction). We pass this vector to a softmax function, which then outputs the discrete probability distribution of all labels at each time-step. The formula of softmax function is shown below:

$$\sigma(h_t)_i = \frac{e^{(h_t)_i}}{\sum_k^{36} e^{(h_t)_k}} \tag{3.166}$$

for $i = 1, 2, \dots, 36$

One possible way to find the top most probable sequences is by treating the probability distribution at each time-step as sets, and then sort the product of each element in the n-fold Cartesian product of these sets in ascending order. However, this method can be computationally expensive as the number of possible combinations increases exponentially with the number of time-steps. In fact, since

our model has 26 time-steps and each time-step has 36 possible labels, there are a total of 36^{26} elements in the Cartesian product and sorting these elements might takes a long time.

Therefore, we proposed a heuristic method to find the top predictions with highest confidence level. For the sake of simplicity, let us assume that there are only 3 time-steps and 3 possible labels, namely “A”, “B” and “C” and the softmax function gives us the following probability distributions at each time-step:

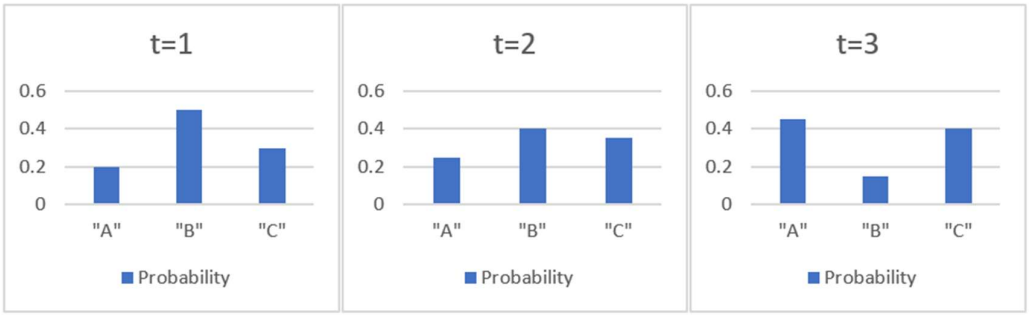


Figure 3.7: Probability distribution of labels at each time-step

Clearly, the most probable label sequence resulted from these sample probability distributions is “BBA”, which was obtained by selecting the labels with highest probability from each time-step.

In order to find the second most probable label sequence, we need to search for the label with highest probability among labels that had yet to appear in the most probable sequence at every time-step. We will then substitute this new label with the label in most probable sequence within the same time-step. For the example given above, we will substitute “A” at time-step 3 with “C” because “C” has a probability of 0.4, higher than other candidate labels that had yet to appear in the sequence.

The next most probable sequence can be found by repeating the steps above, by searching for the highest probability among remaining labels and update the previous most probable sequence with the new label, until there is no any other combination of sequences left.

3.8 Implementation Details

We implemented the whole CRNN network using PyTorch 0.4.1 framework with CUDA version 8.0. All experiments are carried out on a cloud platform named Google Colab that is equipped with single 2.3GHz Intel Xeon Processor and NVIDIA Tesla K80 GPU that has 2496 CUDA cores and 12GB GDDR5 VRAM. The training process that involves 25720 input images took us around 30 minutes on average.

For the EAST detector in our model, due to lack of resources, we did not retrain the whole network using our own data but instead utilised the weights of a pre-trained model. The weights used for our EAST detector can be found inside OpenCV Library, called “frozen_east_text_detection.pb”.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Result

4.1.1 Overview

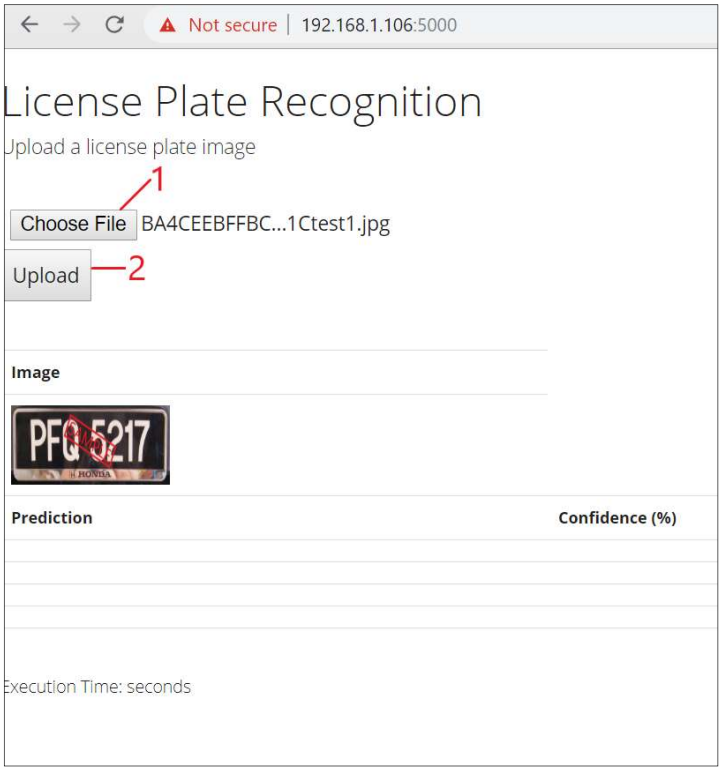


Figure 4.1: Front Page of ALPR Web Application

A simple web application prototype was built in order to demonstrate the recognition ability of the final ALPR model. This web application was built using Flask (Python Microframework) and it can be deployed to cloud providers such as Amazon Web Services, Google Cloud and Heroku. Figure 4.1 shows the home page of the web application.

Users need to click “Choose File” button to select a vehicle license plate image and then click “Upload” button to recognise the text inside the input image.

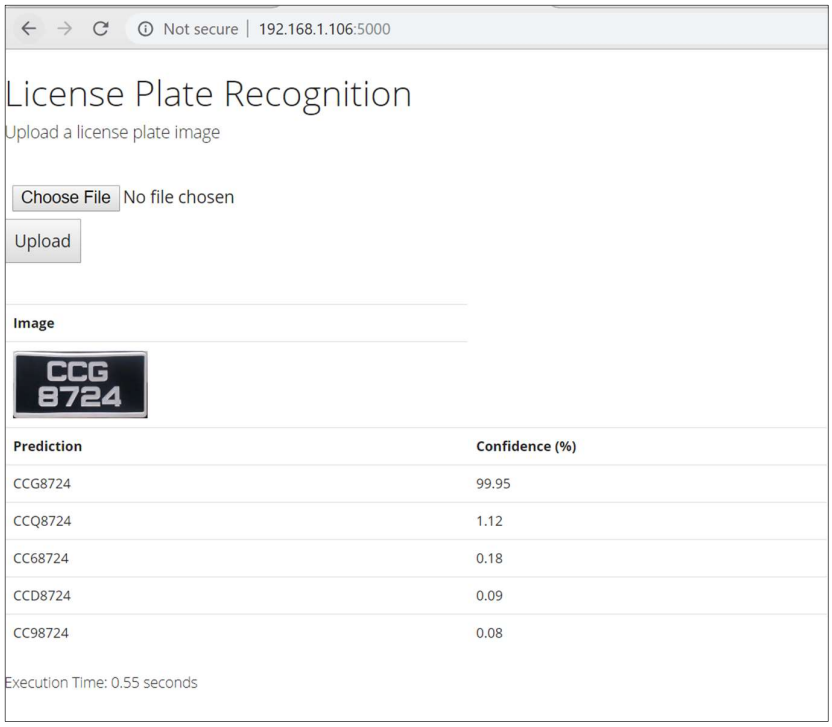


Figure 4.2: Prediction Result of ALPR Web Application

Once the "Upload" button is clicked, the input image will be shown below. Top 5 predictions made by the ALPR model and their corresponding prediction confidence will be displayed under the input image.

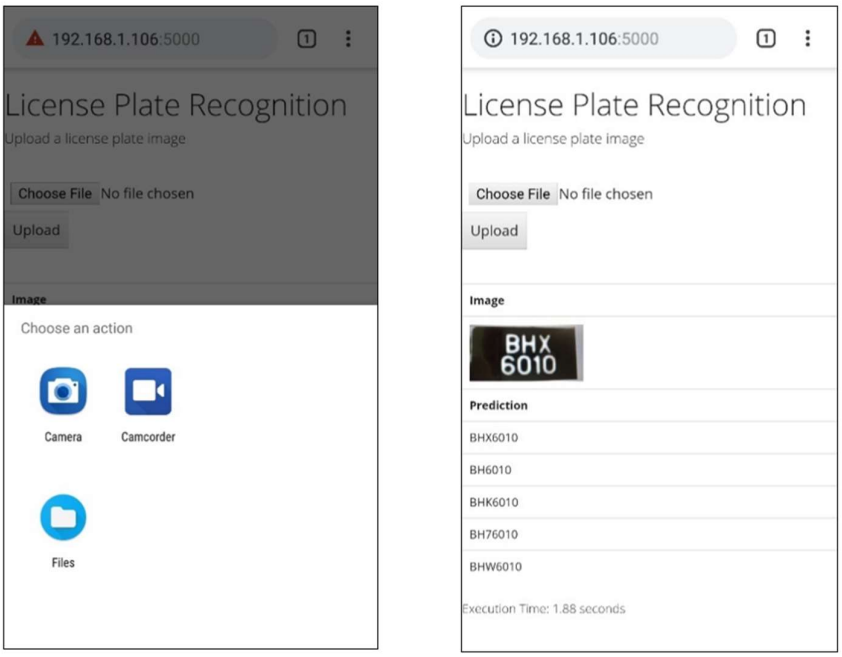


Figure 4.3: Mobile Page for ALPR Web Application

If the user is accessing the web application using smartphone, they can choose to upload the images from camera or select pre-existing files.

4.1.2 Comparison between Candidate CRNN Models

During the CRNN model development process, we trained the model with 60 epochs. After each epoch, the training loss and validation loss are calculated. By comparing the difference between training loss and validation loss, we can roughly speculate whether the model is overfitting or underfitting. If the training loss is much higher than validation loss, this suggests that our model is overfitting whereas a situation when validation loss is much higher than training loss suggests that the model is underfitting. Therefore, it is best to select model with the least difference in training loss and validation loss. The top 3 models with least difference in training loss and validation loss happened during 30^{th}, 36^{th} and 39^{th} epoch and their corresponding values are recorded in the table below.

Table 4.1: Comparison of Training Loss and Validation Loss Between Candidate Models

Model	Training Loss	Validation Loss	|Training Loss – Validation Loss|
CRNN30	0.356654	0.322708	0.033946
CRNN36	0.328760	0.324879	0.003881
CRNN39	0.327699	0.323998	0.003701

The difference between training loss and validation loss should only serves as an indicator of whether the model is overfitting or underfitting. Since, all three models above have relative low difference in training loss and validation loss, we are not able to further deduce which model is better using this indicator. In order to select the best model among these 3 models, we need to look at their performance on our test datasets.

To evaluate the performance of CRNN model, we use 3 separate test datasets for benchmarking purpose, mainly LPR44, LPR45 and Open Environment Dataset. The difficulty level increases across these datasets due to the variation in image quality and noise level with LPR44 having the best image quality and least noise and

Open Environment Dataset having the worst image quality and highest noise level. The overall prediction accuracy for each model are shown in the tables below:

Table 4.2: Comparison of Prediction Accuracy Between Candidate Models Across Test Datasets

Model	Accuracy		
	LPR44 (409 samples) [Difficulty Level - Low]	LPR45 (553 samples) [Difficulty Level – Medium]	Open Environment Dataset (2533 samples) [Difficulty Level – High]
CRNN30	99.2665%	93.4900%	78.7998%
CRNN36	99.2665%	93.6709%	78.4445%
CRNN39	99.2665%	93.4900%	78.6814%

All 3 models performed equally well on the easiest dataset, LPR44 while CRNN36 slightly outperformed other models in LPR45. CRNN30 model, however, achieved the highest accuracy in Open Environment Datasets. Since we want to have a model that generalise well under different environments, CRNN30 was selected as our final model due to its performance in the most difficult dataset.

Note that although the image quality of Open Environment Dataset is very different from our training dataset, we still managed to achieve accuracy level of around 78%. This shows that our model has robust performance due to its ability in handling noises that never appeared in training dataset.

4.1.3 Error Analysis

In order to find out the weaknesses and limitation of our final model (CRNN30), we first list out all 3 mislabelled samples (out of 409 samples) in LPR44 and compute the Leveinshtein distance for each of these samples. The results are shown in the table below:

Table 4.3: Mislabelled samples in LPR44

	Predicted	Actual	Leveinshtein Distance	Input Image
1	BKM4122	BKW4122	1	
2	W9GL486	BGL486	2	
3	WLY872	WLY8725	1	

From the table above, we noticed that the Leveinshtein distance for each sample are within 1 to 2, which suggests that the predictions made are not too far from actual label. After reviewing the mislabelled samples, we observed that one of the license plates, "WLY8725" is difficult to be recognised, even at a human level because the last digit "5" was mostly covered by shadow.

Due to the low error rate in LPR44, we could not observe any noticeable weaknesses in our model. Therefore, we looked into the all the mislabelled samples in LPR45 (36 out of 553 samples) in order to look for any patterns in the errors made. Upon further inspection, there are 2 noticeable situations where the model fails consistently. First, the model is unable to predict a character if it is too close to the edge of the image or if only part of the character is shown at the edge. Second, the model is unable to recognise a character if it is partially or fully covered by shadows. Samples of these mislabelled images can be found in the tables below:

Table 4.4: Mislabelled samples in LPR45 (Character is close to the edge)

Predicted	Actual	Leveinshtein Distance	Input Image
AEQ56	AEQ561	1	

WXG43	WXG431	1	
AJN1	AJN17	1	
BNL95	BNL954	1	
BMD41	BMD415	1	

Table 4.5: Mislabelled samples in LPR45 (Character is blocked by shadow)

Predicted	**Actual**	**Leveinshtein Distance**	**Input Image**
WTT7151	WTT7191	1	
JKQ94	JKQ92	1	
SGX8284	BGX8284	1	
JMK5707	BMK5707	1	
WLH202	WLH203	1	

One possible solution to overcome these limitations is by performing image augmentation at training time. We can introduce additional noises to our train data by purposely cropping some of the images so that the characters are close to the edge. Besides, we can also simulate the effect of shadows on train data by purposely blackening out the pixels of random parts in images.

4.1.4 Character Level Accuracy

Although the most important performance metric in an ALPR system is the accuracy of final predictions, it is also crucial to investigate the performance of character level recognition in our model because a poor character recogniser can ultimately lead to low final prediction accuracy. To do so, we use confusion matrix to get an overview of the misclassification rate at character level recognition. The diagrams below show the confusion matrix for LPR44, LPR45 and Open Environment Dataset:

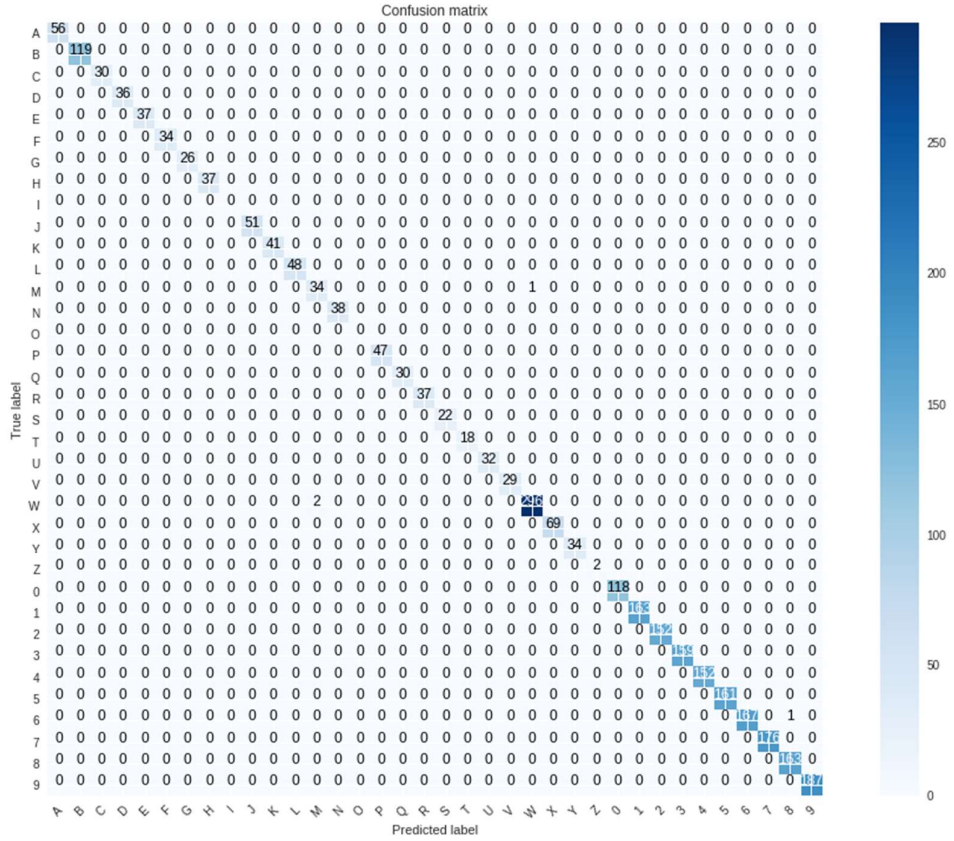


Figure 4.4: LPR44 Confusion Matrix

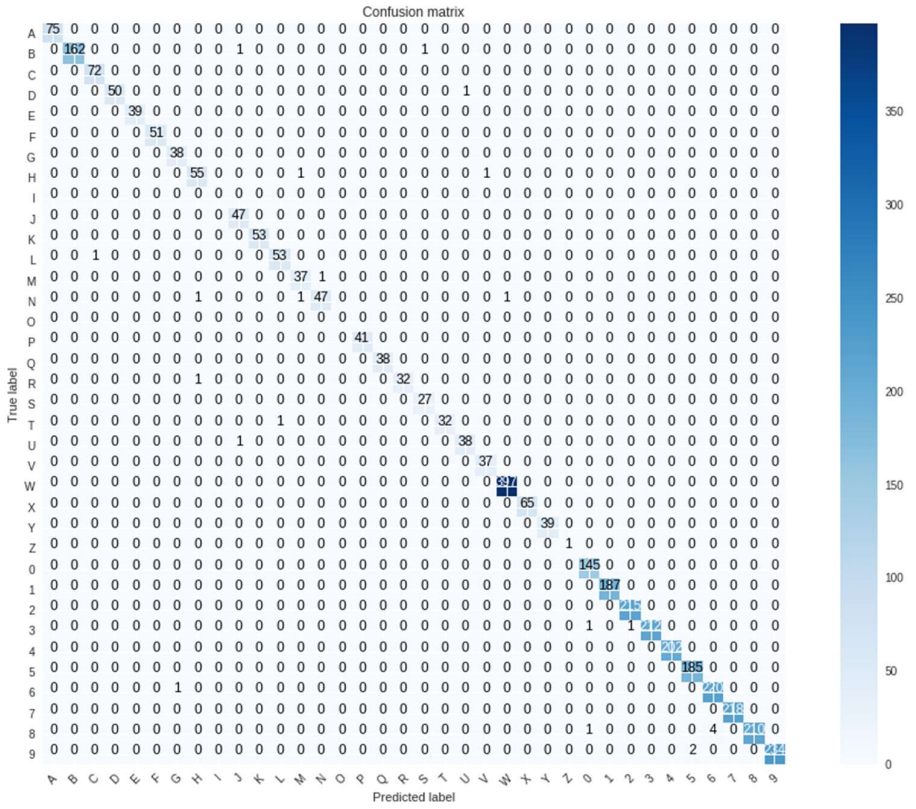


Figure 4.5: LPR45 Confusion Matrix

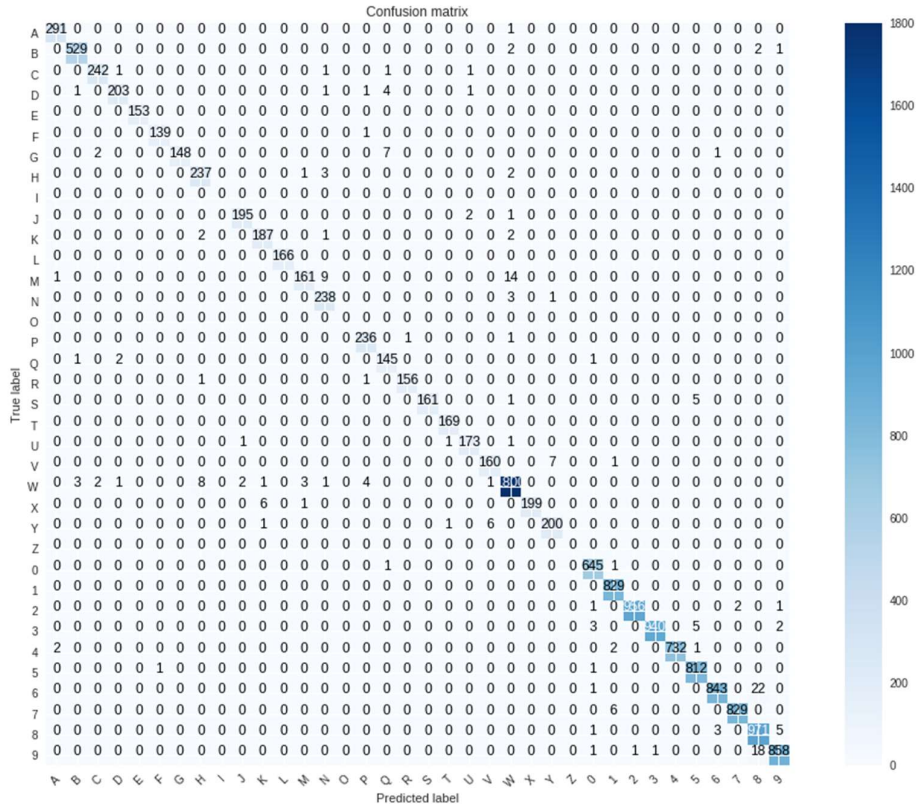


Figure 4.6: Open Environment Dataset Confusion Matrix

From Figure 4.4 and Figure 4.5, we observed that most mislabelled characters only happened once, which suggest that we can treat them as outlier cases and conclude that there is no any noticeable weakness in our model's character recognition ability in predicting LPR44 and LPR45 datasets. However, the same cannot be said for predicting Open Environment Datasets as the model seemed more likely to mislabel certain pairs of characters, mainly

i. "8" and "9" (mislabelled 18 times)
ii. "8" and "6" (mislabelled 22 times)
iii. "M" and "W" (mislabelled 14 times)

One possible reason for this occurrence is due to the low image quality in Open Environment Datasets. Notice that the appearance of number "6", "8" and "9" are visually similar as all of them consist of circle shape in them. When the image quality is low, some of the circle shape might not be obvious and would easily trick the model into falsely recognising these digits.

Regardless, the overall character level recognition ability of the model is satisfactory as the character level accuracy in all 3 test datasets are close to 100%. The table below summarized the character level accuracies across all three test datasets.

Table 4.6: Character Level Accuracy for LPR44, LPR45 and Open Environment Dataset

Test Datasets	**Character Level Accuracy**
LPR44 (2805 characters)	99.8574%
LPR45 (3577 characters)	99.3570%
Open Environment Dataset (14878 characters)	98.6759%

4.2 Comparison with Previous Researches

As mentioned previously in "Section 1.3: Aims and Objectives", the main research objective of this research is to improve the CRNN model proposed by Soo (2017). The table below summarizes the difference in performance of our proposed model and Soo's (2017) model.

Table 4.7: Comparison between proposed CRNN model and Soo's (2017) model

Criteria		Final CRNN Model	Soo's (2017) CRNN Model
Accuracy	LPR44	99.27%	98.04%
	LPR45	93.49%	94.84%
	Open Environment	78.80%	54.85%
Character Level Accuracy	LPR44	99.86%	99.70%
	LPR45	99.36%	98.91%
	Open Environment	98.68%	84.15%
Average Prediction Time		< 1 second	7~8 seconds
Architecture Size		~5.5 million parameters	~1.9 billion parameters
End-to-end Trainable		Yes	No
Ability to Recognize Two-row License Plate		Yes	No
Ability to Predict Sequence with Varied Length		Yes	No

Notice that our proposed CRNN model outperformed Soo's (2017) model in most aspects, such as prediction accuracy, prediction time, architecture size and the ability to predict sequence with varied length and two-row license plate, except having slight lower prediction accuracy on LPR45 dataset.

4.3 Problems Faced and Solutions Taken

4.3.1 Problem 1: Overfitting

When training the model, we observed that the training loss quickly dropped below the test loss. This is a sign that our model is overfitting too quickly.

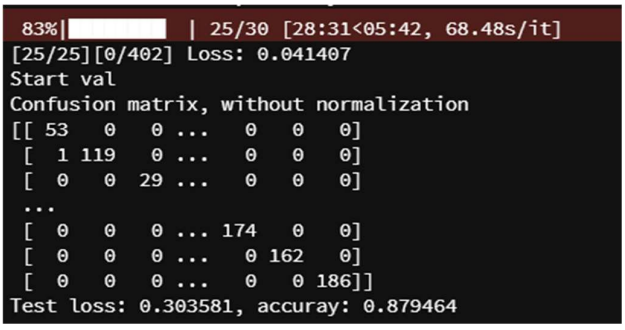


Figure 4.7: Model Overfitting During Training Process

From the figure above, we can notice that the training loss at 25th epoch is 0.041407, much lesser than the test loss of 0.30581, which is a clear sign of overfitting.

Therefore, we added Dropout layers after Conv1, Conv2 and Conv3 layers to regularize the neural networks. Dropout layers will randomly set the activations in our feature maps to 0. This can effectively remove unwanted noise and prevent overfitting.

Table 4.8: New Architecture of Proposed CRNN

Layer Name	**Output size (channels x height x width)**	**Parameters**
Input	1 x 32 x 100	-
Conv1	64 x 32 x 100	#filters: 64, k=3x3, s=1, p=1
Dropout		**p=0.5**
Maxpool1	64 x 16 x 50	k=2, s=2, p=0
Conv2	128 x 16 x 50	#filters: 128, k=3x3, s=1, p=1
Dropout		**p=0.2**
Maxpool2	128 x 8 x 25	k=2, s=2, p=0
Conv3	256 x 8 x 25	#filters: 256, k=3x3, s=1, p=1
Dropout		**p=0.2**
BatchNorm	-	-
Conv4	256 x 8 x 25	#filters: 256, k=3x3, s=1, p=1
Maxpool3	256 x 4 x 26	k=2, s=2x1, p=0x1
Conv5	512 x 4 x 26	#filters:512, k=3x3, s=1, p=1
BatchNorm	-	-
Conv6	512 x 4 x 26	#filters: 512, k=3x3, s=1, p=1
Maxpool4	512 x 2 x 27	k=2, s=2x1, p=0x1
Conv7	512 x 1 x 26	#filters: 512, k=2x2, s=1, p=0
BatchNorm	-	-
Bidirectional-LSTM		#hidden unit: 256
Bidirectional-LSTM		#hidden unit: 256
Transcription	-	-

The parameter p in Dropout layers is the probability of randomly setting activations to 0.

Besides, we also noticed that the decreasing rate of training loss is too slow. Consequently, we added Adadelta optimiser to our model to optimise the Stochastic Gradient Descent (SGD) process. There are many different optimisers available to speed up the SGD process, such as Adam, RMSProp and AdaGrad. The performance of each optimiser on MNIST datasets is shown in the figure below:

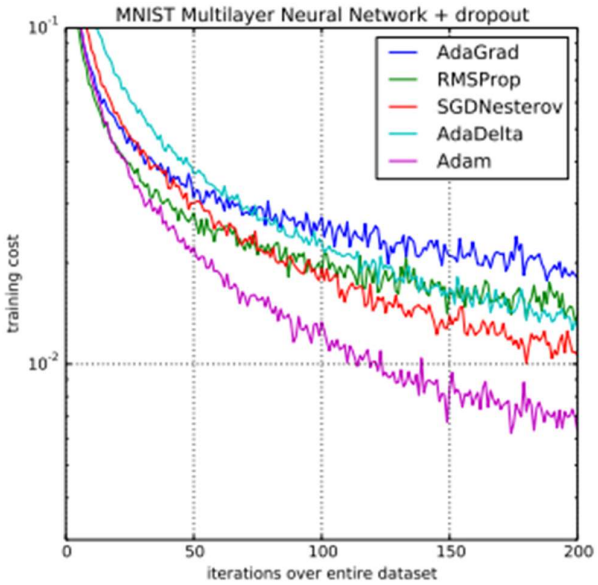


Figure 4.8: Performance of Optimiser on MNIST Dataset

After adding Adadelta optimiser into our model, we observed that the training loss decreased at a much faster rate. The training loss started at 73.391541 and quickly decreased to 18.055309 and 7.281705 during the second and third epoch.

```
  0%|          | 0/30 [00:00<?, ?it/s]
[0/25][0/402] Loss: 73.391541
Start val
Test loss: 73.166763, accuray: 0.000000
[0/25][360/402] Loss: 19.649828
Start val
Test loss: 18.496323, accuray: 0.000000
  3%|▌         | 1/30 [01:19<38:20, 79.34s/it]
[1/25][0/402] Loss: 18.055309
Start val
Test loss: 18.619183, accuray: 0.000000
[1/25][360/402] Loss: 14.276855
Start val
Test loss: 7.494201, accuray: 0.035714
  7%|▋         | 2/30 [02:23<33:26, 71.68s/it]
[2/25][0/402] Loss: 7.281705
Start val
Test loss: 6.454233, accuray: 0.058036
[2/25][360/402] Loss: 4.796819
Start val
Test loss: 2.883775, accuray: 0.397321
```

Figure 4.9: Effect of Optimiser During Training Process

Nevertheless, the decreasing rate of training loss slowed down rapidly after around 10^{th} epoch, when the training loss had been reduced to less than 1. This suggests that there is a need to adjust the learning rate after 10^{th} epoch because lower

learning rate is usually needed if the training loss is already very low. Thus, we adopted the Cosine Annealing learning rate scheduler to adjust the learning rate as the number of epochs decrease.

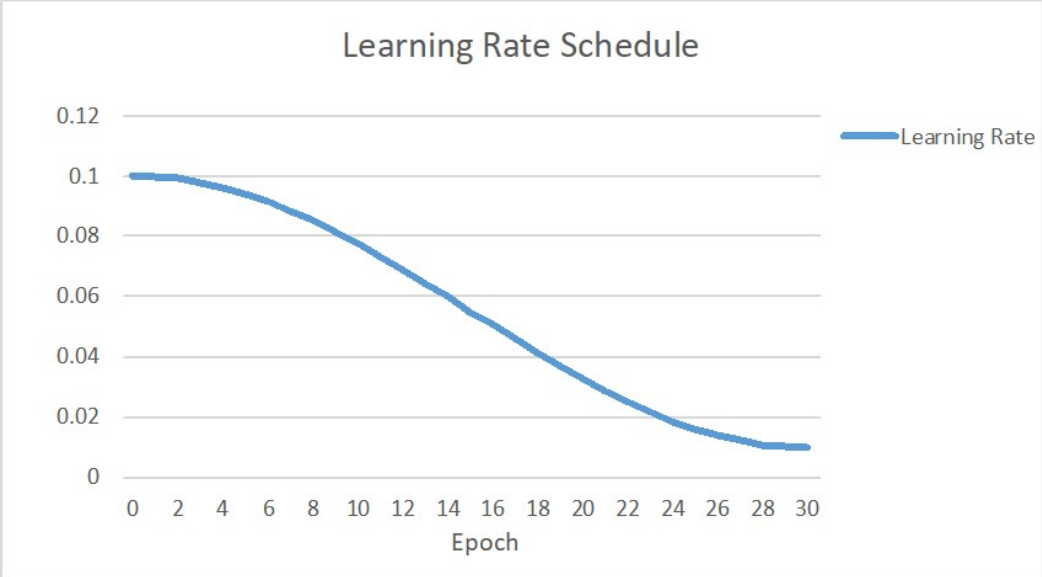


Figure 4.10: Learning Rate Schedule

The figure above showed the learning rate schedule in our model. As the number of epochs increases, the learning rate decreases following the shape of a cosine curve. The initial learning rate is set at 0.1 and gradually decreased to 0.01.

4.3.2 Problem 2: Low Prediction Accuracy for Two-row License Plate

When we first test for our ALPR model performance, the prediction accuracy for two-row license plate was not performing well. Upon further investigation, we realised that the underlying problem was because the text in detected text regions are too close to the edge. Since our training data for CRNN were consisted of single-row license plate images which typically have spaces between the edge of image and the characters, our CRNN model struggled to recognise these cropped images that looked too different from the training data.

In order to resolve this issue, we introduced some noises to the detected text regions by increasing the area of detected text regions suggested by EAST model. This is to ensure that the cropped images are visually similar to our train data, which has spaces between characters and edges of images.

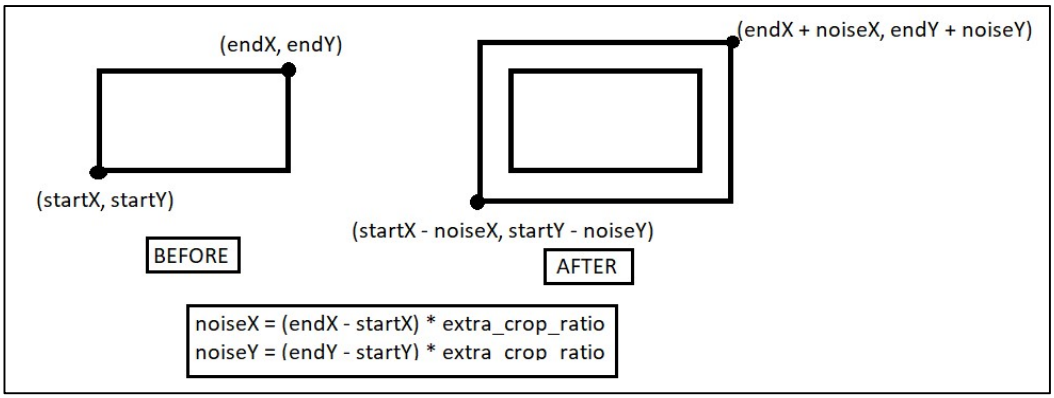


Figure 4.11: Increasing Area of Detected Texts

Figure 4.12 shows the cropped regions for detected texts before and after introducing noises and the improvement in prediction.

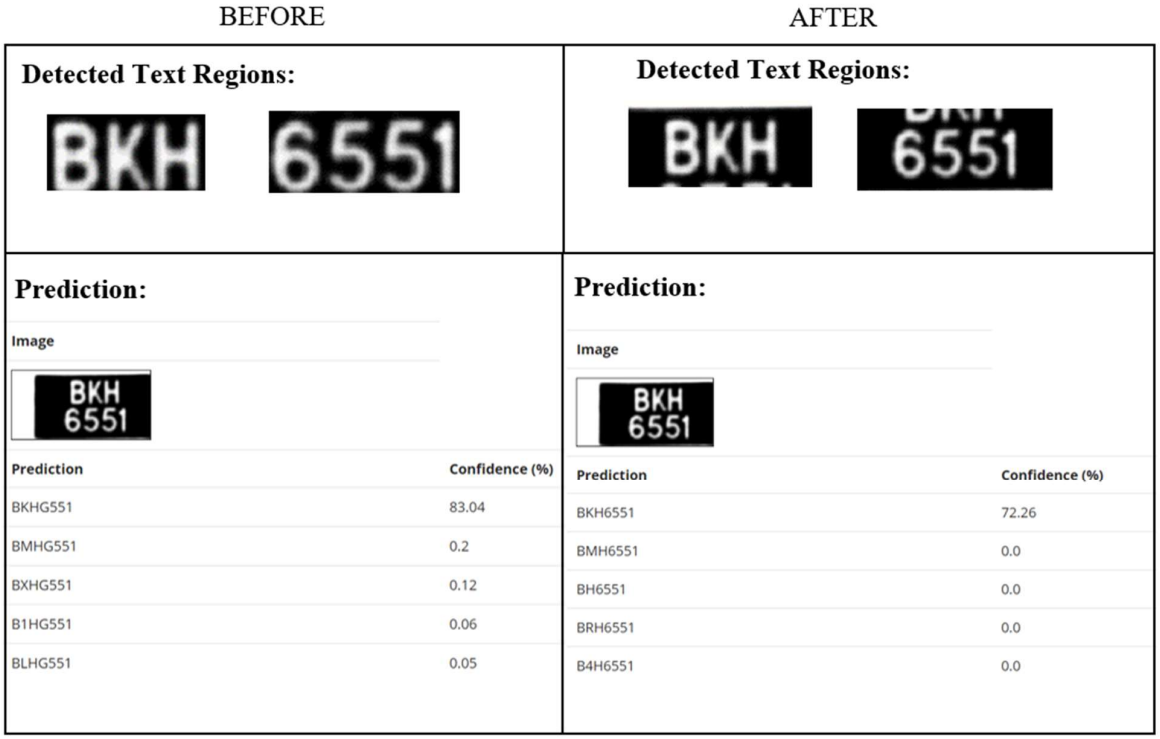


Figure 4.12: Predictions Before and after Increasing Area of Detected Text Regions

4.3.3 Problem 3: Detecting non-license plate texts

BEFORE

Detected Text Regions:

Prediction:

Image

Prediction	Confidence (%)
H0N0APFQ5127	26.69
0N0APFQ5127	11.7
0NDAPFQ5127	5.35
W0NDAPFQ5127	0.23
WWQNDAPFQ5127	0

AFTER

Detected Text Regions:

Prediction:

Image

Prediction	Confidence (%)
PFQ5217	100.0
PFQ517	0.0
PFQ5817	0.0
PFQ5117	0.0
PFQ5317	0.0

Figure 4.13: Predictions Before and after Filtering out Small Text Regions

Another problem that we observed during the model testing process is that texts that are not part of license plate, such as the car brand name and car distributor name that are written under the license plate were also detected and recognised by our model.

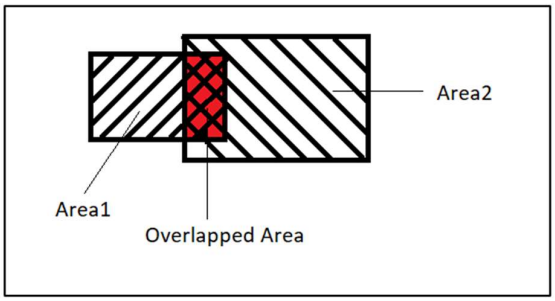
In order to prevent this occurrence, we need to set a threshold to filter out text regions that are comparatively smaller than other text regions. This can be done by computing the ratio of area of each text regions to area of largest text region and remove the regions with ratio less than a pre-set threshold set.

$$Text\ Region\ Size\ Ratio = \frac{Area\ of\ text\ region}{Area\ of\ largest\ text\ region}$$

4.3.4 Problem 4: Overlapping cropped regions

While testing our model performance, we also realised that there are cases where the detected text regions are overlapped. This will cause the final prediction to have duplicated texts.

To avoid this situation, we decided to compute the overlapped ratio of detected text regions and drop the regions where overlapped ratio is at an unacceptable range. Figure 4.12 shows how the overlapped ratio is computed.



$$Overlapped\ Ratio = \frac{Overlapped\ Area}{Area}$$

Figure 4.14: Overlapped Ratio Calculation

The improvement in prediction before and after considering the overlapped ratio is shown in Figure 4.15.

BEFORE

Detected Text Regions:

Prediction:

Image

Prediction	Confidence (%)
WB6WB6266S	58.26
WBEWB6266S	23.33
WB8WB6266S	4.77
WBWB6266S	3.4
WB5WB6266S	0.72

AFTER

Detected Text Regions:

Prediction:

Image

Prediction	Confidence (%)
WB6266S	99.86
WB8266S	0.12
WB8266	0.0
WB5266	0.0
WB5466	0.0

Figure 4.15: Predictions Before and after Filtering out Highly Overlapped Text Regions

4.4 Limitation of Model

The main limitation of our ALPR model is that it can only recognise uppercase letters. With that said, it cannot recognise some of the vanity license plate that contains lowercase letters, such as “Putrajaya” or “BAMbee”, a plate that was introduced during 2000 Thomas and Uber Cup which was held in Kuala Lumpur.

Besides, our model is designed for detecting Malaysia license plate only, which either has single-row or two-row. Therefore, its performance on foreign license plate is not guaranteed due to differences in license plate format and font type.

Moreover, there are two noticeable situations where our model fails consistently. First situation is when the characters in license plate is very close to the edge of image and the second situation is when there are shadows covering part of or whole input image.

Another drawback of our suggested pipeline is that it contains two stages, text detection (for two-row license plate) and text recognition. The main disadvantage of having multiple stages is that the overall performance is highly determined by the interplay of these stages or components. In other words, if either component fails, the final prediction will most likely be inaccurate. Besides, post-processing is usually required when 1 or more stages are involved. In our case, we need to filter out detected text regions that has much smaller area compared to other detected regions to remove any potential noises. Also, we have to ensure that the overlapped regions among the detected text regions are within an acceptable range. We also had to explicitly increase the size of detected text regions to ensure that the characters are not too close to the edge.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this study, we had developed an ALPR system that can be used to recognise both single-row and two-row license plates that are available in Malaysia. We combined two neural networks in a single pipeline, mainly EAST for text detection and CRNN for text recognition. We proposed a small CRNN architecture design which abandons the usage of fully connected layers and this design is proven to be more effective compared to larger CRNN architecture.

Our model had successfully outperformed Soo's (2017) CRNN model in the following aspects:

i. Higher Overall Accuracy
ii. Higher Character Level Accuracy
iii. Shorter Average Prediction Time
iv. Smaller Architecture Size
v. End-to-end Trainable
vi. Is Able to Recognize Two-row License Plate
vii. Is Able to Predict Sequence with Varied Length

Besides, due to the nature of a CRNN model, we avoided using any character segmentation algorithms, which in turn allows us to skip all pre-processing that usually requires by segmentation algorithms. Also, due to the usage of deep neural network, our model shows high robustness in performance due to its capability in handling noises.

5.2 Future Work

5.2.1 Integrate Text Detection and Recognition in a Unified Framework

Our current model consists of two stages, text detection and text recognition that utilised two different networks, particularly EAST and CRNN. The main disadvantage of using two separate neural networks is that we have two convolutional layers that are used for the same purpose — feature extraction, which

can be deemed to be redundant. Instead, a better approach is to extract features from input images using only one convolutional layer and the extracted feature maps will be utilised for both text detection and text recognition tasks. This can effectively make our network more compact by reducing the architecture size and at the same time avoid post-processing that might be required after text detection. Ideally, this novel neural network architecture should be end-to-end trainable, which means that we are able to train the convolutional layers, text detector (regressor) and text recogniser (classifier) within the same neural network concurrently. Training a multi-output neural network is possible by using weighted sum of regression loss and classification loss.

5.2.2 Dataset Expansion and Data Augmentation

In order to make our model more robust, we can collect and add license plate images from foreign country into our train datasets so that it can predict license plates from not only Malaysia, but also from other countries.

Besides, we can also expand our current train datasets by performing data augmentation. The augmentation techniques that need to be applied should simulates the problems that commonly face by images captured in the real world. For example, we can randomly blacken out the pixels of part of an image to simulate the effect of shadows on license plate.

REFERENCES

Amodei, D., et al. 2015. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. [online]. https://arxiv.org/abs/1512.02595

Bartz, C., Herold, T., Yang, H. and Meinel, C. 2017. Language Identification Using Deep Convolutional Recurrent Neural Networks. [online]. https://arxiv.org/abs/1708.04811

Bu, Y. J. and Xie, M., 2013. A New Method for License Plate Characters Recognition Based on Sliding Window Search. IEEE 11th International Conference on Dependable, Autonomic and Secure Computing, Chengdu, China, 21-22 December, 2013, pp. 304-307.

Choi, K., Fazekas, G. and Sandler, M. 2016. Convolutional Recurrent Neural Network for Music Classification. [online]. https://arxiv.org/abs/1609.04243

Epshtein, B., Ofek, E., and Wexler, Y. 2010. Detecting text in natural scenes with stroke width transform. 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 2963–2970

Gilly, D. and Raimond, K., 2013. License Plate Recognition – A Template Matching Method. International Journal of Engineering Research and Applications, pp. 1240-2345.

González, Á., Bergasa, L. M., Yebes, J. J. and Bronte, S. 2012. Text location in complex images. Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), pp. 617–620

Gou, C., Wang, K., Yao, Y. and Li, Z. 2016., Vehicle License Plate Recognition Based on Extremal Regions and Restricted Boltzmann Machines. IEEE Transactions on Intelligent Transportation Systems, 17(4), pp. 1096-1107.

Graves, A., Fernandez, S., Gomez, F. and Schmidhuber, J. 2006. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Network. Proceedings of 23rd International Conference on Machine Learning, Pittsburgh, 2006.

Hsu, G.S., Chen, J. C. and Chung, Y. Z., 2013. Application-Oriented License Plate Recognition. IEEE Transactions on Vehicular Technology, 62(2), pp. 552-561.

Li, Y. and Lu, H. 2012. Scene text detection via stroke width. Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), pp. 681–684

Shanbhag, A., Thakkar, R. and Patel, H. (2015). A Comparative Study of Text Detection Algorithms for Natural Scenes. International Journal of Research in Computer and Communication Technology, vol. 4, no. 9, pp. 735-740

Shi, B., Bai, X. and Yao, C., 2015. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. [online]. https://arxiv.org/abs/1507.05717

Soo, C. P. 2017. Segmentation-free License Plate Recognition using Deep Learning. [online].
http://eprints.utar.edu.my/2747/1/FYP2_SooChingPau_1305682_SE.doc.pdf

Turkyilmaz, I. and Kacan, K., 2017. License Plate Recognition System Using Artificial Neural Networks. ETRI Journal, 39(2), pp. 163-172.

Zhong, Z., Jin, L., Zhang, S. and Feng, Z. 2016. DeepText: A Unified Framework for Text Proposal Generation and Text Detection in Natural Images. [online]. https://arxiv.org/pdf/1605.07314.pdf

Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W. and Liang, J. 2017. EAST: An Efficient and Accurate Scene Text Detector. [online]. https://arxiv.org/pdf/1704.03155.pdf