Building End-to-End Chinese Car-plate Detection and Recognition System

Mengjie Shen (ms11141)*, Yajie Xiao (yx1750)*, Yuanhao Shen (ys3426)*
New York University Shanghai

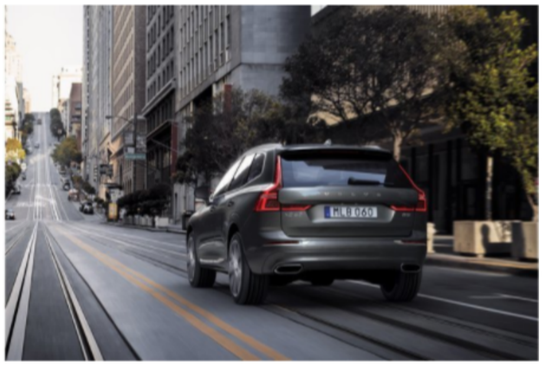
***Abstract*—This project aims at building an end-to-end Chinese car-plate detection and recognition system using deep neural networks. The project is divided into two parts: car-plate detection and character recognition. In the first stage of the system, we would customize the YOLOv5 object detection model to locate the bounding box coordinates and take the sub-image containing the car-plate. Then, we would conduct pre-processing on the sub-image: pre-scaling the images, transforming to grayscale images, and drawing contours of each character in the car-plate. The main contribution of this project is that we perform a modified version of LeNet 5 CNN to recognize the Chinese character and apply pytesseract to recognize the numerals and alphabets.**

***Index Terms*—Machine Learning, Deep Learning, Printed Text Recognition, Computer Vision, OCR, LeNet**

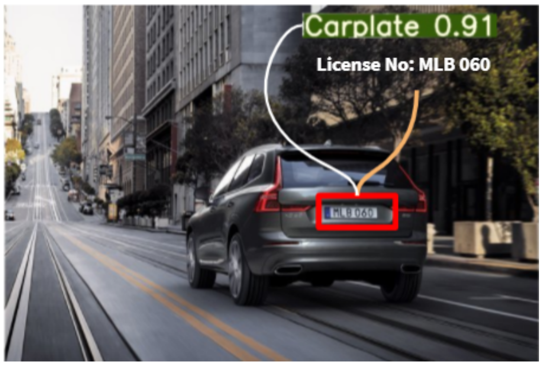
I. Introduction

Cars have become one of the most popular means of transportation in China. However, with the surging number of cars on the road, problems regarding traffic control also emerge, such as the increasing traffic violation rates. For example, a single CCTV camera takes over 40,000 pictures per day. In order to increase the efficiency of vehicle regulation, it is essential to implement a Machine Learning model that can detect and recognize car-plates from images taken by the CCTV cameras. Car-plate detection and recognition is widely applied in many real-world surveillance system, such as traffic monitoring and parking system. This project aims at developing an end-to-end Chinese car-plate detection and recognition system, and as the name suggests, it includes two parts: car-plate detection and text recognition.

For the task of detecting car-plates from images, we implement object detection model YOLOv5 proposed by Redmon et al. [1]. YOLOv5 is the fifth version of You Only Look Once, a neural network that predicts bounding boxes and class probabilities directly from full images in one evaluation. The model outputs the coordinates of a bounding box around the target object and it would be suitable for extracting a sub-image of the car-plate as car-plates are mostly in rectangular shape unless largely skewed. We can further implement text recognition tasks on the sub-image of the car-plate.



(a) Sample Input



(b) Sample Output

Figure 1. Example of how the system works

A standard Chinese car-plate is consist of a Chinese character indicating the province (e.g. the character “湘” stands for Hunan Province), and a string combination of both numerals and alphabets. In order to achieve both high accuracy and efficiency, we divide the text recognition task into two sub-tasks, Chinese character recognition and numeral/alphabet recognition. The car-plate character is in a specialized font developed by the Ministry of Public Security, and therefore unmatched in common computer systems, so the Chinese character recognition is equivalent to handwritten character recognition rather than machine-printed character recognition.

Preliminary research shows that LeNet-5 convolutional neural network proposed by Yann LeCun et al. achieves high accuracy in handwritten character recognition. And we would apply a modified version of LeNet-5 based on Yann LeCun et al.’s paper from 1998 to capture the finer details of the Chinese character [2].

For numeral and alphabet recognition, we work with the Python package pytesseract which is based on tesseract, a command-line optical character recognition (OCR) program that can read the text contained in pictures.

II. Data Pre-processing and Augmentation

The data set used in the project is from a GitHub open source repository, which mainly consists of images that were taken by digital cameras, with various dimensions, in different places and times [3]. Due to the legal privacy and other issues involved in the data used for training Chinese car-plates, the available data set lack in quantity, and many are taken by very old cameras and under bad lighting. So we implemented data augmentation techniques to improve the quality of the image set.

In addition, before feeding our data set into text recognition models, we would conduct a series of pre-processing procedures to increase the recognition accuracy. The pre-processing procedure is divided into three parts: (1) Training the car plate detector to find the rough location of the car plate (2) detect the edges of the area and locate the vertices of the rectangle (3) apply perspective transformation to obtain the flattened image of the car plate.

1) **Training the car plate detector to find the rough location**
We start with the labeling of the image, which involves annotating the approximate area of the car plate, and storing the data into a text file containing the pixel coordinates. These processed data is then compiled into a yaml file, acting as an index for the training set.

Due to the limited amount of data we collected, we use the module called Roboflow to perform data augmentation, including image rotating and image shearing. As we have 196 images originally, we get 588 images in total after data augmentation. Then, we customized the object detection model YOLOv5 provided by Ultralytics and feed the augmented data into the model [4]. We have found that when we send the epoch at 500, we get the best precision and recall rate of the model. The output is consist of the input image and the car plate area predicted by the YOLOv5 model with a confidence level expressed as a float number. As figure 2 shows, the YOLOv5 model achieves the state-of-the-art performance at 500 epochs with a confidence threshold which is above 0.8.

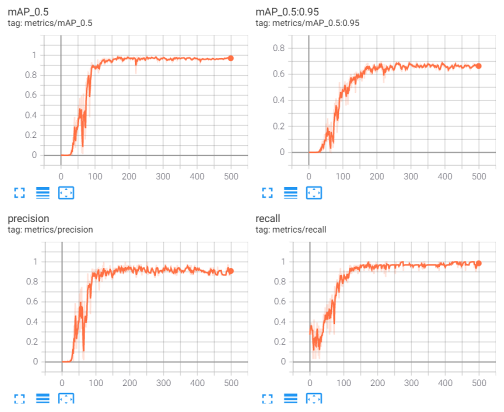
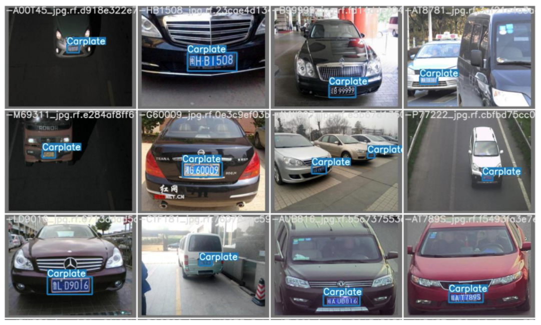
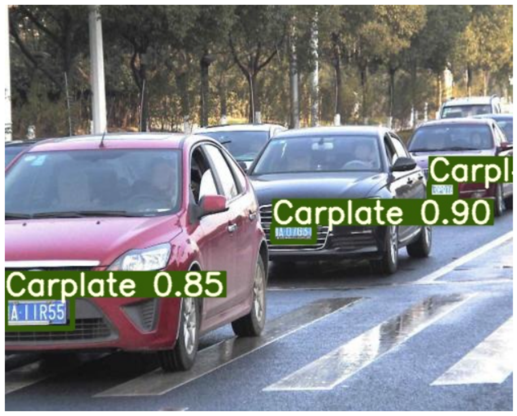


Figure 2. Precision and Recall Metrics of YOLOv5 Model



(a) Sample Training images



(b) Sample Testing image

Figure 3. YOLOv5 Model for Car-plate Detection

2) **Detect the edges of the area and locate the vertices of the rectangle**
With the successful detection of the approximate area of the car plate, we then crop the original image and only maintain the area where the car plate is located. Then the cropped image is fed into a Python file. Here we use the openCV module, a python image processing package. We first convert it into the gray scale image, which helps to reduce the noise of the lines. In order to convert from three channels (i.e. RGB) into gray scale image, we apply the average method, which is to calculate the

avearge of the three channels and then condense them as 1 channel between [0, 255]. Then, we uses the Gaussian blur formula:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-k+1)^2}{2\sigma^2}\right); \tag{1}$$

where $1 \leq i, j \leq (2k+1)$

Bilateral filter is also applied to reduce the noise interference. Below is its definition:

$$I^f(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(||I(x_i) - I(x)||) g_s(||x_i - x||) \tag{2}$$

where W_p is defined as follows:

$$W_p = \sum_{x_i \in \Omega} f_r(||I(x_i) - I(x)||) g_s(||x_i - x||) \tag{3}$$

Then the altered images are sent into the Canny function to draw the contour lines in the gray scale image. Then, we find out the first 10 closed contour lines that have the largest areas in order to ensure that the area describing the exact shape of car plate is included in the list. We then use the rectangle detection algorithm to filter out the areas that are not rectangle and the largest rectangular contour is set as the exact edges of the car plate.

3) **Apply perspective transformation**
As we have the information regarding the exact regions of the car plate, we then need to restore the tilted image or the sheared image back to the flatted view as if we are looking at the car plate from the top down angle in order to tweak the characters back to the form that is closest to its original form. Below is the code snippet used to conduct the perspective transformation:

```
def four_point_transform(image, pts):
    # obtain a consistent order of the
    # points and unpack them individually
    rect = order_points(pts)
    (tl, tr, br, bl) = rect
    # compute the width of the new image
    widthA = np.sqrt(((br[0]-bl[0])**2)
            + ((br[1] - bl[1]) ** 2))
    widthB = np.sqrt(((tr[0]-tl[0])**2)
            + ((tr[1] - tl[1]) ** 2))
    maxWidth = max(widthA,widthB)
    heightA = np.sqrt(((tr[0]-br[0])**2)
            + ((tr[1] - br[1]) ** 2))
    heightB = np.sqrt(((tl[0]-bl[0])**2)
            + ((tl[1] - bl[1]) ** 2))
    maxHeight = max(int(heightA),
                int(heightB))
    dst = np.array([
        [0, 0],
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight-1]],dtype="float32")
    # compute the perspective
    # transform matrix and then apply it
    M = cv2.PerspectiveTransform(rect,dst)
    warped = cv2.warpPerspective(image,
            M, (maxWidth, maxHeight))
    # return the warped image
    return warped
```

After transformation, the image will make the model more robust while increasing the prediction accuracy.

III. Methodology for Character Recognition

1) **Tesseract for Number and Alphabet Recognition**
In order to correctly detect the characters on the car-plate after we input the pre-processed gray scale images, we started with the pytesseract Optical Character Recognition (OCR) package since it is the optimal solution in recognizing printed texts. However, we find that although pytessseract perform relatively well in detecting alphabets and numbers, it has some bottlenecks at recognizing the Chinese characters on the car-plate, therefore causing a relatively low accuracy in the actual performance. Considering the bottleneck of the existing models is caused by pytesseract, we decided to implement deep neural networks to recognize the Chinese characters separately and then combine it with the pytesseract model in order to improve the accuracy and efficiency of the overall model.

2) **Modified LeNet for Chinese Character Recognition**
   a) Modified LeNet Structure
   We implement the LeNet-5 model to recognize the Chinese character at the first position of the car plate based on Prof. Yann Lecun's model architecture for recognizing handwritten digits in 1998 [5]. We modified the structure of the original neural network to better suit our case.

   The number of units in each fully connected layer is decided because of size of the input data is 64x64, and also on the nature of Chinese characters. The following are the justifications for some of the modifications from the original paper:
      i) The original LeNet-5 paper specified that the first full-connected layer with 120 units was in fact a convolution layer, but since the kernel size matched the input size for that layer, it functioned as a full-connected layer. Here, we explicitly state it as a FCL, and made it to have 500 units to capture the finer details of Chinese characters.
      ii) The kernel size is now 3x3 - since Chinese is a language with fine and distinct strokes, which might be blurred and thus, hidden if we use a larger kernel.
   b) Choice of Optimizer and the Loss Function
   We use *optim.Adam* as the optimizer, with a learning-rate of 0.0005. The Adam optimizer is

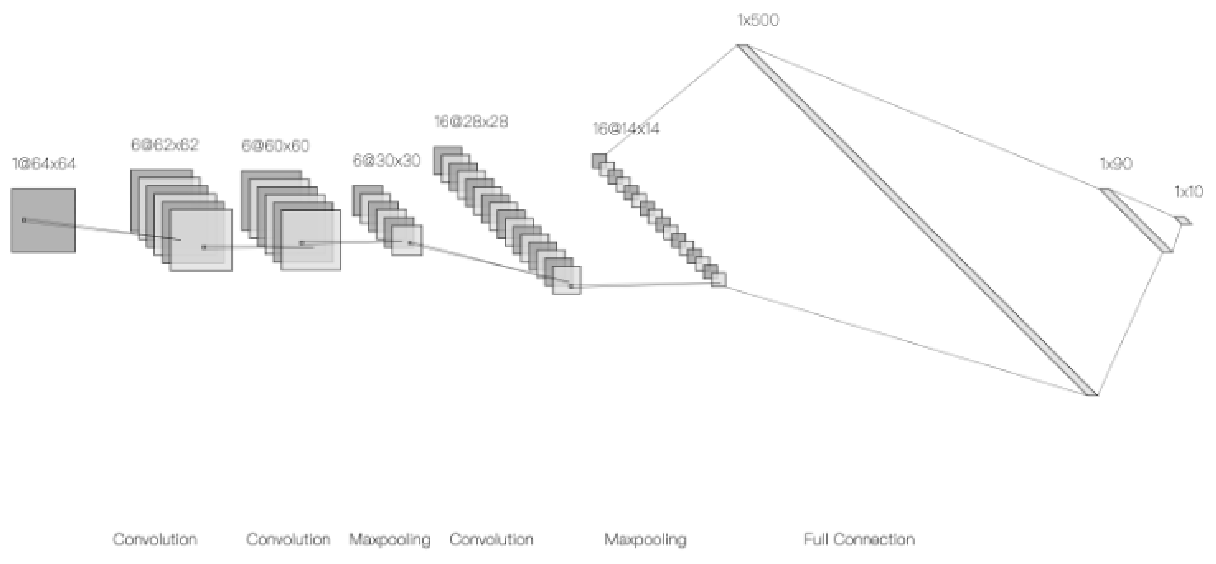


Figure 4. Modified LeNet-5 Architecture

generally deemed to converge faster than stochastic gradient descent, which is a major point in its favor. The loss function used is *nn.CrossEntropyLoss* since this is a multi-class classification problem. The formula is given as follows,

$$loss(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right)$$
$$= -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

For the model evaluation, we use accuracy score, which is

$$accuracy = \frac{\#correctly\ classified}{\#total\ data}$$

c) Model Training
Considering our data set is relatively small, we performed the train-validation-test split, and then implemented data augmentation using package *augmentor* and generated the data set with 10000 images.

We used GPU to execute our code, considering it takes a lot of time to train such a large data set on CPU. This is actually our first time to learn the associated syntax and add the relevant changes to our original code.

We first tried to train the model with 50 epoches. We observed that the train accuracy reached nearly 100% after around 20 epoches, and the validation accuracy reached the peak at around 40 epoches. Figure 5 shows the plot of train and validation accuracy against number of epoches.

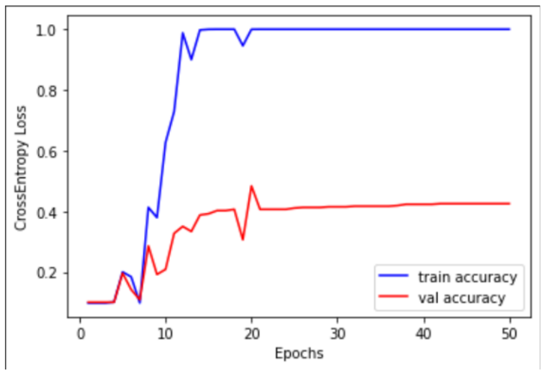


Figure 5. Plot of accuracy against number of epoches

d) Performance of the test data set
We tested our model on the testing data set with 1000 images. It achieved the accuracy of 85.1%. The confusion matrix generated by results is presented in figure 6. We can see that the color on the diagonal line of the matrix is lighter than the others, meaning that our model recognized most of Chinese characters correctly. However, it still has some limitations in recognizing some of the characters. For example, the accuracy for class 4,

which corresponds to Lu (鲁) is relatively low, the model wrongly recognized it as class 0,1,7, which are correspondingly Gan, Gui, Yu (赣, 贵, 豫).

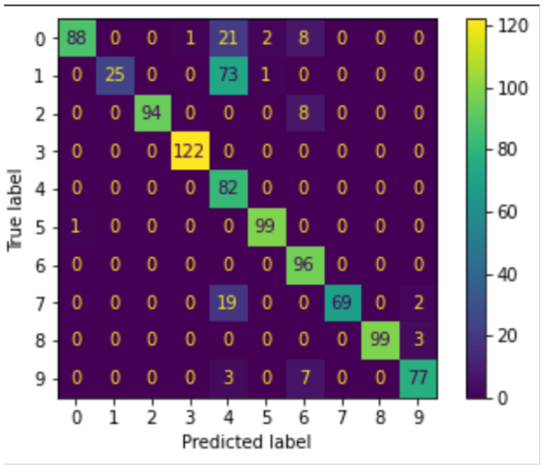


Figure 6. Plot of Confusion matrix

IV. Conclusion

In this paper, we present data augmentation techniques when faced with low quality car-plate dataset and a novel deep neural network architecture for Chinese car-plate detection and recognition. We use Roboflow, an online data processing website to construct data for model training. Since the model tends to run slowly and typically costs around 60 minutes to complete for big epochs such as 300 or 500, we split the augmented data into mini batches in order to accelerate the training process and find that a batch that is typically around 20 speeds up the training process by approximately 20%. Images of cars are then being detected for car-plates and car-plates are divided into two parts according to the unique features of Chinese car-plates for more efficient recognition, which includes the utilization of openCV along with respective modules for edge detection, vertex anchoring and perspective transformation. Additionally, we construct novel model structure combined with YOLOv5, tesseract and LeNet-5 and data pre-processing techniques in order to adapt to our pre-processed data while making our model valuable for object detection, object segmentation, and object recognition.

The overall performance of the model is satisfying, with an accuracy of 80% in the prediction of the Chinese characters on the car plate, as we have seen on the graph along with the confusion matrix above. For the recognition of alphabets and numbers, pytesseract achieves impressive accuracy when handling these situations. Therefore, we find our overall structure of the project logical and the outcome meaningful for real world applications.

V. Discussion

However, there are also a few future improvements that are necessary for our overall model to achieve state-of-the-art performance under real-world application.

First, we should include a larger data set in order to reduce the bias of our model. Since we only find 196 images from GitHub, the data may not be representative in terms of constructing a complete system, and such problem is salient when the input image is taken in a relatively low light environment, or the exposure of the car plate is not accurate. Besides, our system fails to recognize those special car plates in China, such as the ones with Chinese characters are at the top while the numbers are at the bottom, and the car plates for specialized vehicles such as ambulances and police cars. Also in terms of LeNet, since the data set for Chinese car plate character is hard to find, we cropped from the car plates and labeled them by ourselves. And then we generated images using data augmentation due to the relatively small size of the data set. This makes the images we generated share relatively similar features with each other and has a relatively high correlation. This is probably the reason why we reached nearly 100% for the training accuracy.

Also, the accuracy of edge detection desires to be improved. As we only used the Canny function in the openCV package, problems such as broken edges sometimes occur, which deteriorates the forming of a closed rectangle, which affects the capturing of the precise location of the car plate under the ranking of the areas. Such problems can be solved by constructing another model dedicated to edge recovery and connection, as Muhammet Baştan has mentioned in his paper "Active Canny: edge detection and recovery with open active contour models" [6]. He suggests that machine learning models may help to learn connecting the broken edges through the implementation of a mini memory system. If this is incorporated into our project, the overall accuracy will experience a huge boost when we make classifications.

Finally, the performance of our LeNet neural network should also have a slight tweak in order to achieve the optimal performance. From the confusion matrix of the model above, we have experimented it with 10 classes of Chinese characters representing different provinces. However, it is salient that label 1 is often confused with label 4, corresponding to the Chinese character "gui" and "lu". Therefore, we should tune the hyper parameters such as the number of epochs more carefully, or slightly tweak the learning rate within the gradient descent so that the error rate of false prediction could experience a decrease. Moreover, we are currently using Adam with a fixed learning rate. We could also try different optimizers and see which performs better, for example, Stochastic Gradient Descent plus momentum may outperform Adam as we learned in class.

Acknowledgment

The authors would like to thank professor Li Guo for her help and support during the process of finishing this project.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] airxiechao, “airxiechao/simple-car-plate-recognition,” https://github.com/airxiechao/simple-car-plate-recognition.

[4] Ultralytics, “ultralytics/yolov5,” https://github.com/ultralytics/yolov5.

[5] U. Chaturvedi, “Modified lenet-5: Chinese mnist,” https://www.kaggle.com/youteek/modified-lenet-5-chinese-mnist/execution.

[6] Muhammet Baştan, “Active canny: edge detection and recovery with open active contour models,” https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/iet-ipr.2017.0336.