

Automatic Number Plate Recognition and User Identification during accidents

A project report submitted in partial fulfillment
of the requirements for the

Third Year of Computer Engineering

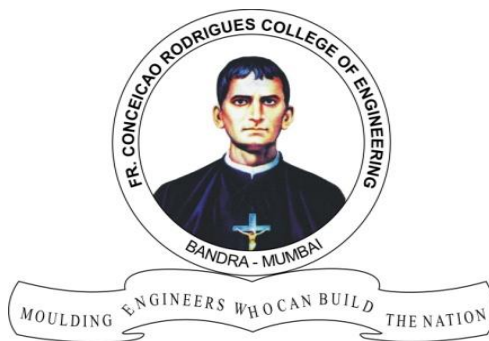
by

Astle Pinto (8627)

Rolwyn Raju (8634)

Joshua Godinho (8607)

Under the guidance of
Prof. Kalpana Deorukhkar



DEPARTMENT OF COMPUTER ENGINEERING

Fr. Conceicao Rodrigues College of Engineering, Bandra (W), Mumbai - 400050

University of

Mumbai 2020-21

*This work is dedicated to my family.
I am very thankful for their motivation and support.*

Internal Approval Sheet

CERTIFICATE

This is to certify that the project entitled "**Project Title With Every First Letter in Upper Case**" is a bonafide work of **Astle Pinto (Rollno:-8627)**, **Rolwyn Raju (Rollno:-8634)**, **Joshua Godinho (Roll no:-8609)** submitted to the University of Mumbai in partial fulfillment of the requirement for term work submission of Project based learning Third year Computer Engineering.

(Name and sign)

Supervisor/Guide

(Name and sign)

Head Of Department

(Name and sign)

Principal

Abstract

Traffic control and vehicle owner identification has become major problem in every country. Sometimes it becomes difficult to identify vehicle owner who violates traffic rules and drives too fast. Therefore, it is not possible to catch and punish those kinds of people because the traffic personal might not be able to retrieve vehicle number from the moving vehicle because of the speed of the vehicle. Therefore, there is a need to develop Automatic Number Plate Recognition (ANPR) system as a one of the solutions to this problem. There are numerous ANPR systems available today. These systems are based on different methodologies but still it is really challenging task as some of the factors like high speed of vehicle, non-uniform vehicle number plate, language of vehicle number and different lighting conditions can affect a lot in the overall recognition rate. Most of the systems work under these limitations.

Contents

1. INTRODUCTION

2. LITERATURE REVIEW

3. PROPOSED SYSTEM

3.1. VEHICLE DETECTION

1. YOLO Model
2. YOLO custom Dataset
3. Data Preprocessing
4. Data Configuration

3.2. LICENSE PLATE DETCTION

1. Components
- 2 License Detection
3. Segmentation
4. Prediction
5. Implementation

4. Web Application deployment
 - Implementation of models using Django
 - Creating the user-friendly interface using ReactJS
5. RESULT
6. CONCLUSION
7. REFERENCES

Chapter 1

1. Introduction

Deep learning has been a huge topic in machine learning research closer to Artificial Intelligence. A few decades ago, computer scientists developed a number of algorithms that trained computers to distinguish multiple instances of the same object.

With the growing number of drivers, and an increase of vehicles on the road come problems associated with traffic. Some of these problems, such as accurate bridge and highway tolling, parking lot management, and speed prevention, can now be solved using machine learning. This project will explore the use of deep learning for the purpose of vehicle tracking and license plate recognition.

This project gave us the basic understanding of the modern neural network and how it works with applications in computer vision. By using the building blocks of neural networks, we were able to improve the accuracy of a model with its pre-trained model. We used our understanding of the pre-defined building model of the neutral network to compare the model accuracy using TensorFlow and Keras framework.

2. Literature Review

There are many published papers that discuss topics about vehicle detection. We will focus on one article named ‘Image-based vehicle analysis using deep neural network: a systematic study’ . In this article, they use a DNN (YOLO) model to achieve vehicle detection. The article also covers training for vehicle classification on normal images and dark images. We are able to infer the success of our chosen project through the study of this article.

3.Proposed System

Our project is divided into two major components: vehicle detection and license plate character recognition.

Phase I is detecting whether an input contains vehicles or not. If the algorithm predicts that the input contains vehicles, then we need to define a method which will precisely locate and crop the vehicle from the original data. Starting with the output from Phase I as the input, Phase II should search for the vehicle’s license plate and have the ability to recognize the number on the car plate.

Through investigating, we found that vehicle detection is already quite developed and there are numerous existing methods online;; however, we still decided to explore deep learning through this project. We attempt to improve the accuracy with some pre-trained models.

What’s more, in consideration of time and resource limitation, (we neither have enough time nor access to any GPU machine), we decide to challenge image based detection rather than video based.

3.1 Vehicle Detection

3.1.1 Initial Plan

We first decided to build our own CNN model while using ImageNet as our starting point. We also considered applying some data augmentation on our dataset as a feasible method to improve prediction accuracy. Based on this idea, we did some research on currently popular solutions for vehicle detection.

3.1.2 YOLO Model

YOLO caught our attention during our online investigation, as it was the most popular method of real-time object detection. Unlike prior detection systems, which apply the model to a target at multiple locations with different scales, YOLO only applies a single neural network to the whole target image. According to the YOLO official website, the image is divided into regions with different weighted predicts bounding boxes. Each region has its own probability. Compared to previous work for object detection like R-CNN, which requires recursion and needs a lot of time and memory, YOLO only looks once at the whole image. Therefore, it is able to digest more global context in the image which makes it work extremely fast.

Since ‘car’ is a known class for YOLO, we decided to use this method. We found some valuable code from GitHub, which gave us insight into the use of YOLO weights to detect objects and how to implement visualization of the detected results. The available code uses TensorFlow implementation with pretrained YOLO small weight.

Based on the code we found, we first made a single non-greyscale image as input data, and tried to test whether the code performed detection well or not. Every image that passes into the program is resized to 448x448, in order to match the model. In the meanwhile, we found that OpenCV gives wrong colour to coloured images when loading. This is because OpenCV uses BGR as its default colour order for images while matplotlib uses RGB. When displaying an image loaded by OpenCV in matplotlib, the colours will be incorrect. This problem requires the use of OpenCV to explicitly convert data back to RGB:

```
RGB_img=cv2.cvtColor(BGR_img,cv2.COLOR_BGR2RGB)
```

We also implemented the visualization part to mark the detected vehicle on the input image and cropped it for future use. Our implementation is able to crop multiple detected cars as well (see Figure 1).

According to the source code, the author defines his own layers rather than using Keras layers and sets up a convolutional neural network model. We wanted to apply YOLO tiny weight from the YOLO official website to the model schema in Figure 2 by using a similar method, but the result was unsuccessful. The reason for this failure is that the weight from YOLO website is designed for a specific model which is built in a .cfg file. In the .cfg file, the model is not sequential. After some investigation, we cannot find any Keras parallel model. For the purpose of using pre-trained weight, we needed to convert the .cfg format model into the model suitable for Keras through darkflow. This sample code helped us to do this job. Due to the time limitation of our project, we determined not to use this method.

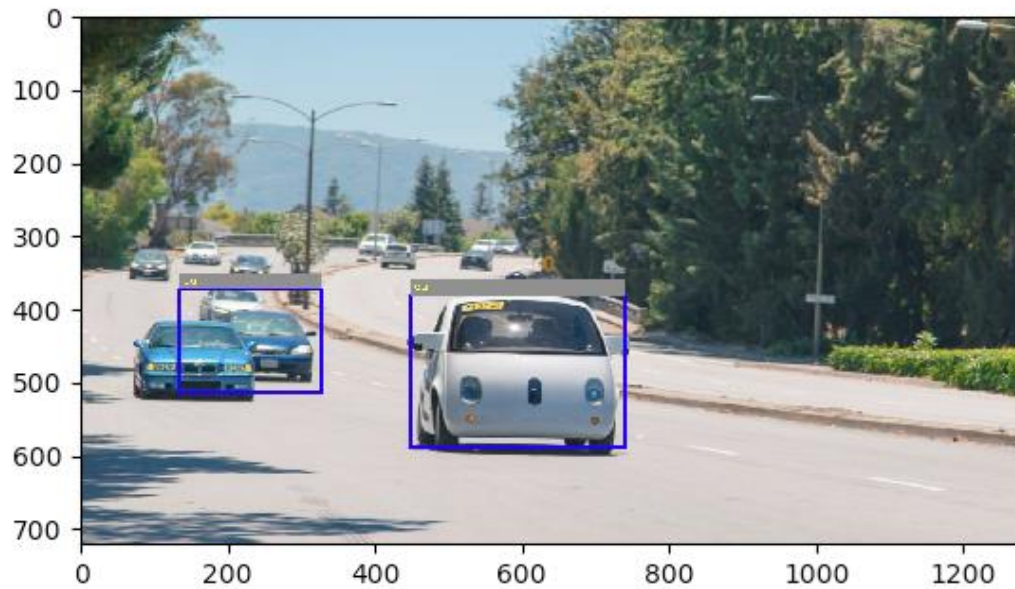
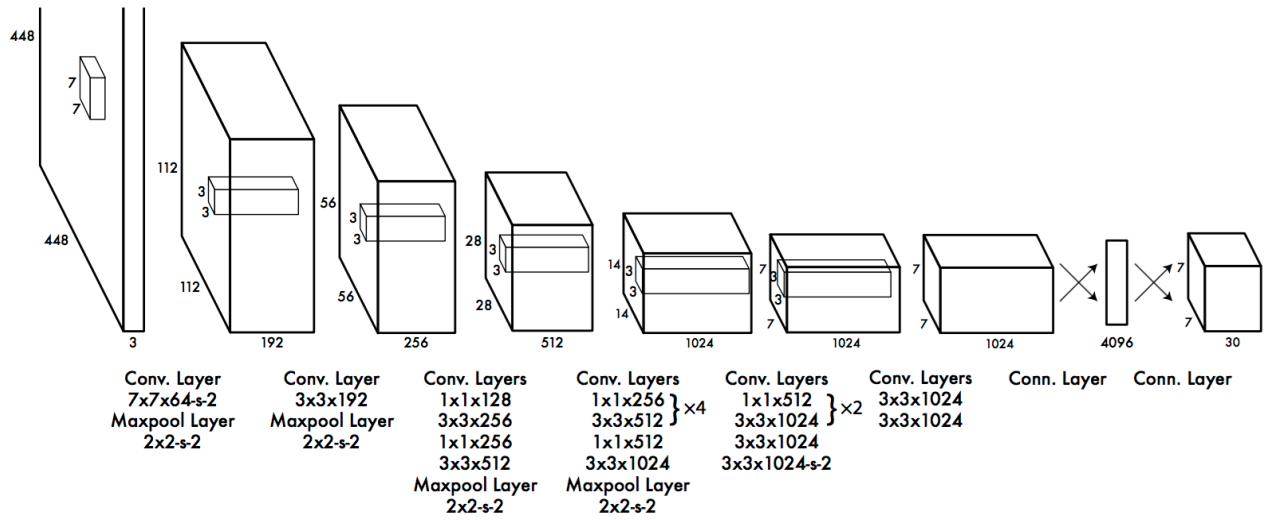


Figure 1. Tiny YOLO model [2]

Instead, we passed in 8000 test images through a loop. The images were all from a ‘cars dataset’ published by Stanford University [4]. This is a large-scale, fine-grained dataset of cars. The average image size of this dataset is 980 x 728, and the largest image resolution is 3280 x 2240. The reason for choosing this dataset over some other small dataset is that we needed to crop the detected vehicle and pass the output to process the license plate for character recognition. If the resolution of initial input is too low, the whole program will lose accuracy.

So, there is a trade-off between speed and performance. Large images required more training/testing time and a more powerful device, but it gave a higher chance for us to detect a car and the car plate. But a problem arose while using the high-resolution images as a test set;; the memory was not large enough to accommodate them. So, after looping the whole program 32 times (in order to process 32 images), the system crashed and read the 33rd image as a grayscale image.



3.1.3 YOLO Custom Dataset

Since the modification of neural network structure is limited by time and knowledge, we decided to adopt the common YOLO network structure, but we trained it from scratch with our custom dataset. We were interested in the extent of improvements via training with the relatively small number of data.

Some specifications need to be clarified before proceeding into this project. First, YOLO program is implemented on the Linux platform, which means that modifications and debugging will be easier in Linux instead of Windows. Second, several applications are required to execute the files, including Python 2.7, OpenCV and the Darknet Toolkit. Notice that neither a virtual environment nor a specific backend is required since the YOLO configuration file is already implemented. Third, due to conflicts of the Nvidia graphic driver on a dual booting system, the CUDA toolkit cannot be utilized here, which in other words, means we cannot use a GPU to perform training. Lastly, in order to save time on data processing, we will focus on image dataset rather than real-time detection.

3.1.4 Data Pre-processing

The first challenge of the project was pre-processing our custom datasets. The dataset was obtained from the Stanford website. This dataset contained both a training set and a validation set, and each set contained more than 8000 car images. YOLO detection features the way of bounding boxes as shown in Figure 1. So, for car detection, we needed to manipulate our training sets and generate the information of each bounding boxes.

We used the BBox-labeling tool to manually draw the bounding box. This tool was able to record the appearances of cars in each image, and the absolute coordinates of the bounding boxes. Unavoidably, the accuracy of training depended on our drawing

in this case. In addition, as mentioned in its documentation, this tool only recognized images with the suffix of “.JPEG”. Therefore, non-JPG type files needed to be converted first, and file names were required to be reorganized by some naming software like Bulk Rename in Ubuntu.

However, YOLO program recognized the coordinates in a different way from the labelling tool. YOLO expected XY coordinates of both the object centre and width, while the labelling tool generated XY coordinates of four vertices . We then used the script called “convert.py” from Darknet to complete this conversion .

3.1.5 YOLO Configuration

Next, we needed to implement a custom YOLO configuration file in order to train our dataset efficiently. In the Darknet program, several configuration files were already given, including the whole neural network structure setup as shown in Figure 2. Therefore, there were only a few items to be modified. In the car detection phase, there was only one class which is car itself. The object data was referred to by the directories of all training images and validation images. The batch number indicated the number of images processed at each training step, which was further rectified by subdivisions according to the computer performance. The number of filters was determined by the following formula:

$$\text{Filters} = (\text{Class} + 5) \times 5$$

Which in our case, was equal to 30 . Furthermore, we will take a pre-trained weight as a

starting point. But from the testing results later on, we find this weight is basically of scratch.

3.2 License Plate Detection and Character Recognition

3.2.1 Components

In order to create an effective license plate reading program, we needed to divide the process into separate components. First, the license plate had to be located within an image of a car. Next, the individual letters and numbers contained within the plate needed to be identified and cropped. Finally, the value of the cropped images of the individual characters had to be predicted.

3.2.2 License Plate Detection

The initial design of the license plate detection component was based on the assumptions that the input image would be a close-up picture of a vehicle with a North American style license plate. This component was developed using the Scikit-Image library, in addition to open source code by Femi Oladeji .

The input image was first converted to a binary image using Otsu's method by creating a threshold. The connected components in the binary image are then located and labelled. These labelled regions have properties which can be iterated through. Using the `regionprops()` function of the Skimage library, the labelled regions were compared with certain criteria to see if they could be a license plate.

We tried many different combinations of criteria to determine if a region was a license plate, such as the region's size relative to image size, and the height to width ratio. Unfortunately, it was difficult to develop a reliable set of criteria since there were many possible variances in the input images. If successful, this step yielded a region that contained the black and white license plate.

3.2.3 Segmentation

Locating the characters within the license plate was done in a similar fashion as the previous step. First, the region containing the plate was labelled. Then, the regions within the plate were tested against some criteria. In order to find the license plate characters, the criteria we used were minimum and maximum dimensions based on the plate size. Once located, the characters were resized to 32x32, and appended to a list to be used in the next step.

3.2.4 Prediction

Character recognition was achieved using deep learning. Specifically, we created a convolutional neural network using Keras and TensorFlow. Our first model, based on The Semicolon's code only gave us 71% accuracy . We subsequently changed to a model by Anuj Shah , which gave us much better results. The CNN model consisted of three two- dimensional convolution layers, and multiple drop out and max-pooling layers. The model is summarized in figure 7. We chose to use the adam optimizer as our update procedure. In order to train the network, we used a data set called Chars74k, which contained thousands of images consisting of letters and numbers . We narrowed down the dataset to only black and white computer fonts of capital letters and numbers from 0 to 9. At this point we ran into an issue regarding the competency of our hardware. Since we did not have access to a high-performance GPU, we had to train on a laptop CPU. During training, the sensors were reporting very high CPU temperatures. Due to the risk of overheating the CPU, we decided to constrain our training in a few different ways. Primarily, we wanted to keep training time at a minimum. Therefore, we decided to cap the number of epochs at 10. Another way to reduce training time, as well as memory usage, was to reduce the size of the training images. We decided that 32x32 images would be good enough to achieve the result we were expecting.

The newly trained model took about 2 minutes per epoch for a total of about 20 minutes. The reported accuracy was ~94% and a test loss was ~0.225. This model was saved into a .h5 file to be used in the prediction program.

The prediction program was initially designed to read the segmented letters from the previous segmentation step. This required saving images to numpy arrays, and expanding the array dimensions to satisfy Keras' input requirements. For TensorFlow, the input had to be the following:

3.2.5 implementation

After training the “characters & numbers” and segmenting a license plate, figure 9 shows the connection between both tasks at the testing step.

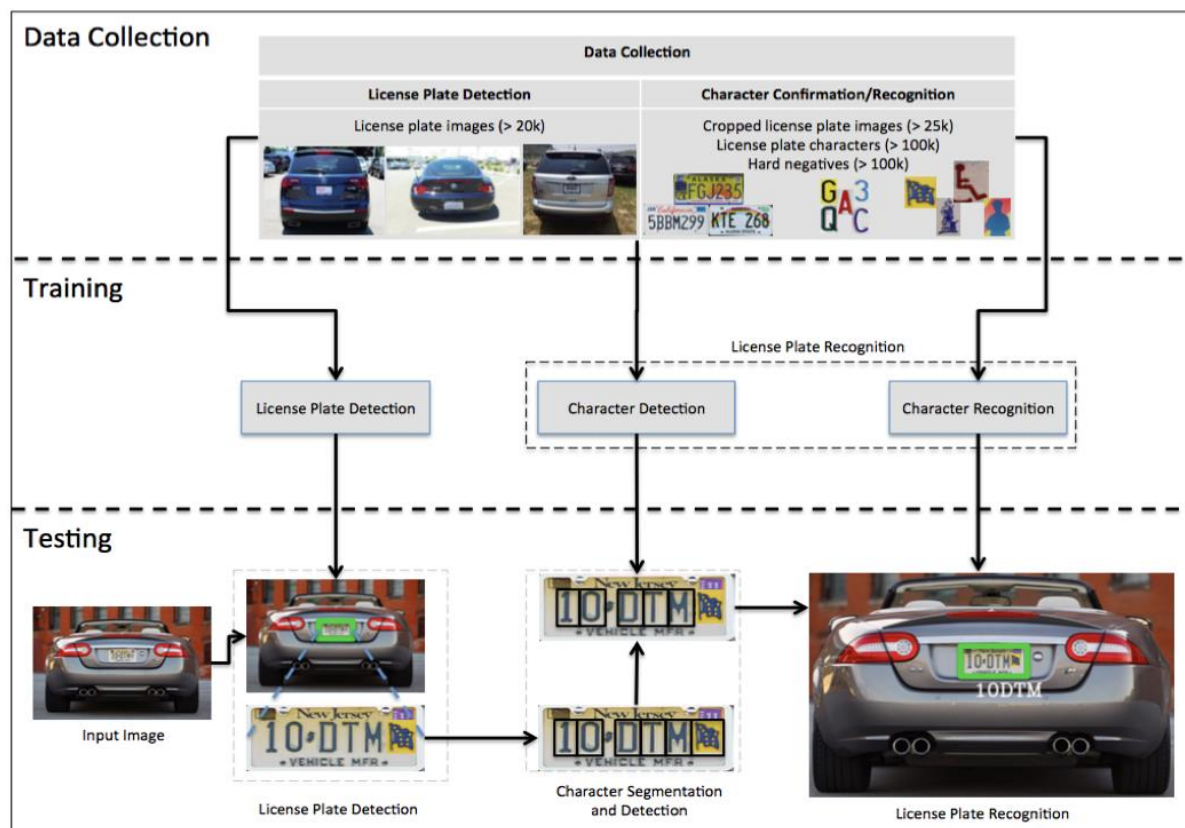


Figure 9 .Merge both character segmentation and detection algorithms

Screenshots of ML .pynb file

```
In [2]: import cv2
import numpy as np
import os
import pyteseract
import matplotlib.pyplot as plt
%matplotlib inline

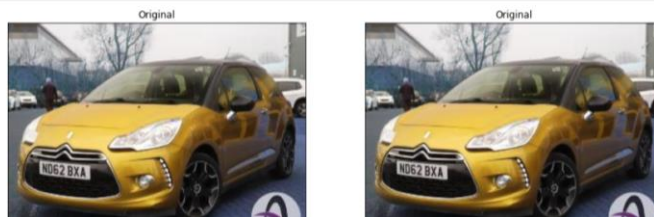
In [3]: def plot_images(img1,img2,title1="",title2=""):
fig=plt.figure(figsize=[15,15])
ax1=fig.add_subplot(121)
ax1.imshow(img1, cmap="gray")
ax1.set(xticks=[],yticks=[],title=title1)

ax2=fig.add_subplot(122)
ax2.imshow(img2, cmap="gray")
ax2.set(xticks=[],yticks=[],title=title2)

In [263]: path = "./images/car5.jpg"

In [264]: image = cv2.imread(path)

In [265]: plot_images(image,image,title1="Original",title2="Original")
```



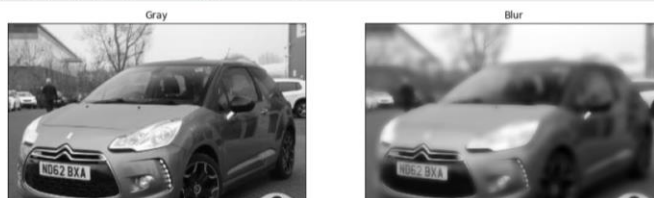
```
In [266]: gray= cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

In [267]: plot_images(image,gray,title1="Original",title2="Gray")
```



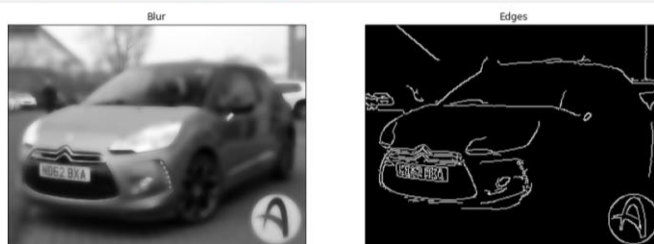
```
In [268]: blur= cv2.bilateralFilter(gray, 11, 90,90)

In [269]: plot_images(gray,blur,title1="Gray",title2="Blur")
```



```
In [270]: edges=cv2.Canny(blur, 30,200)

In [271]: plot_images(blur,edges,title1="Blur",title2="Edges")
```

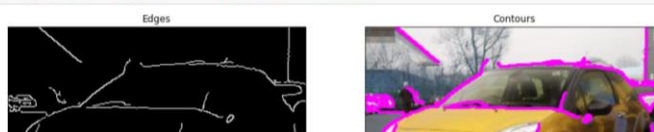


```
In [272]: cnts, new=cv2.findContours(edges.copy(),cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

In [273]: image_copy=image.copy()

In [274]: _=cv2.drawContours(image_copy,cnts,-1,(255,0,255),2)


In [275]: plot_images(edges,image_copy,title1="Edges",title2="Contours")
```




```

In [277]: cnts=sorted(cnts,key=cv2.contourArea,reverse=True)[:10]

In [278]: image_reduced_cnts=image.copy()
          _=cv2.drawContours(image_reduced_cnts,cnts,-1,(255,0,255),2)
          plot_images(image_copy,image_reduced_cnts,title1="Contours",title2="Reduced Contours")

Contours                                     Reduced Contours


In [279]: print(len(cnts))
10

In [280]: plate = None
          for c in cnts:
              perimeter=cv2.arcLength(c,True)
              edges_count=cv2.approxPolyDP(c,0.02 * perimeter, True)
              if len(edges_count)==4:
                  x,y,w,h=cv2.boundingRect(c)
                  plate=image[y:y+h,x:x+w]
                  break
          cv2.imwrite("plate.png",plate)


Out[280]: True

In [279]: print(len(cnts))
10

In [280]: plate = None
          for c in cnts:
              perimeter=cv2.arcLength(c,True)
              edges_count=cv2.approxPolyDP(c,0.02 * perimeter, True)
              if len(edges_count)==4:
                  x,y,w,h=cv2.boundingRect(c)
                  plate=image[y:y+h,x:x+w]
                  break
          cv2.imwrite("plate.png",plate)

Out[280]: True

In [281]: plot_images(plate,plate,title1="plate",title2="plate")

plate                                     plate


In [ ]:

In [284]: import pytesseract
          pytesseract.pytesseract.tesseract_cmd=r'C:\Program Files\Tesseract-OCR\tesseract.exe'
          text = pytesseract.image_to_string(plate, lang='eng')
          print(text)
ND62 BXA

In [ ]:

```

4. Web Application Deployment

The team has come to a conclusion that without a user friendly mobile or web application, the above ML model wouldn't reach its full perks. Our Web Application which is going to implemented with Django using Python as the backend part with an inbuilt SQLite database and ReactJS for the frontend is going to consider the following real-life scenarios ,

Problem statements adhering to the project:

1. Imagine an accident taking place on the highway and the victim has to inform the police/hospital of his/her location and find the nearest police station/hospital. Also, the victim's family members should be informed of the same.

In this scenario, we can uniquely identify each user by his or her car plate number, and extract their information and contacts of his family members if he wishes to register in our application.

If the user is met in an accident, since he is a registered user, he can directly enter location of accident and get the full advantages of our application.

If he is an unregistered user, he can still get features such as the nearest hospital and police stations and later register on our application after help is provided.

If the victim is conscious, a passer by can avail the features directly through the HomePage and provide help to the victim

We can also add an ML model API in which the user scans his/her wounds and gets immediate wound detection remedies while professional help arrives.

We plan to use the following technologies:

- Google Maps API for live tracker
- Wound Detection Model API
- Folium and Leaflet JS for rendering nearest hospitals and police stations

5. Result

Although integration of all the components was not realized, we were moderately successful with both our vehicle detection and character recognition. For vehicle detection, we were able to predict vehicle with up to 25% confidence, and our training showed a final loss of 0.287. For character recognition, our model had a final test loss of 0.225 and claimed an accuracy of 93%. It worked well for most fonts that we tested, but it is apparent that further work is required to improve it.

6. Conclusion and Summary

In summary, we were able to understand the architecture of the convolutional neural network (CNN) and its training procedure and understand the whole procedure of YOLO custom model. Furthermore, we were able increase the prediction precision by training a small dataset. The aspiration is detecting foreign license plates with the trained international dataset.

In conclusion, there are multiples of applications in computer vision that deep learning could be used for such.

7. REFERENCES

- <https://ieeexplore.ieee.org/abstract/document/8748287>
- <https://arxiv.org/pdf/1802.09567.pdf>
- <https://pjreddie.com/darknet/yolo/>
- https://www.researchgate.net/publication/236888959_Automatic_Number_Plate_Recognition_System_ANPR_A_Survey