EE417 – Term Project

Project Report

Project Title:

Vehicle License Plate Detection and Recognition

Group Members:

İrem Gürak - 25442
Zeynep Tandoğan - 25200

Semester: 2021 Fall
Instructor: Mustafa Ünel
Sabancı University
Faculty of Engineering and Natural Sciences

Table of Contents

1. Problem Definition

License plate recognition is a field that gains importance thanks to its usages in investigation of stolen vehicles, traffic monitoring and control. It needs to be used in daily life activities to regulate life in the traffic. License plate recognition includes license plate detection, character segmentation and recognition stages. In this project, we aimed to perform license plate recognition for the vehicles in the images. It is decided to use Python as a programming language and Google Colaboratory is used as an execution environment. Since Google Colab is a cloud environment, the tasks that requires more time in the local machines such as the training task in the project are completed quickly.

2. Problem Formulation and Solution Methods

This project aims to detect the license and recognize the characters on license plate. There are mainly four steps to complete this task successfully.

These are:

1) Detecting the license plate
2) Processing the cropped license plate image to prepare it for recognition
3) Segmenting and extracting the characters on the plate
4) Applying Optical Character Recognition (OCR) to recognize the characters.

For the first task, it is decided to use yolov4 to perform detection. YOLO (You Only Look Once) is a state-of-the-art, real time object detection algorithm. This algorithm detects various objects by employing convolutional neural networks. This is a common algorithm in computer vision field thanks to its speed, high accuracy, and excellent learning capabilities ("Overview of the YOLO Object Detection Algorithm", 2018). With yolov4, it is aimed to train custom model using a car plate dataset in Kaggle and then detect the plate locations in a frame with this model for further processing. The license plate detection is also performed without using yolov4 in the project, the difference between the performances will be explained in next sections.

To prepare the license plate for the segmentation part, some processing should be performed such as noise smoothing and thresholding. For this mission, OpenCV is used which is a real-time optimized computer vision library. With OpenCV, various tasks can be performed like binarization, blur and edge detection.

For segmentation, there are two approaches that are used in the project. The first one is to detect contours and use them for segment the characters from each other. The other one is histogram approach. By following the intensity changes in the cropped image, which is a binarized license plate, the characters are separated. Both approaches are used together in the implementation.

Optical Character Recognition systems transform two dimensional images of text into machine-readable text. OCR is performed by using EasyOCR library. EasyOCR is a Python package that can be used for 58 languages (Rosebrock, 2022).

At the beginning of the project, the steps and solution methods are planned, and the implementation was done in accordance with the plan to a large extent.

3. Implementation Details

3.1. Detecting the License Plate

Two different implementations are done for this section. While one of them does the detection by training with yolov4, the other provided detection by using computer vision techniques without yolov4.

3.1.1. Detection by Training Yolov4

3.1.1.1. Overview

The YOLO algorithm was first addressed by Joseph Redmon, et al. (2016) in the paper titled "You Only Look Once: Unified, Real-Time Object Detection." Different from other algorithms, this approach applied a single neural network of multiple convolutional networks to the full image, the authors have completely abandoned the previous detection paradigm of "proposal detection + verification". The system produced predictive vectors corresponding to each object appearing in the image and instead of iterating the process of classifying different regions on the image, it computed all the features of the image and made predictions for all objects at the same time. This algorithm outperformed the other detection methods in terms of time and although it was not accurate as much as two stage detectors, still achieved a good performance (Redmon et al., 2016). Since one stage process consumes less time, YOLO algorithm was more suitable to be used in real-time applications. There were different versions of this algorithm, the one we used, was "YOLOv4". The original YOLO algorithm was written by Joseph Redmon, who is also one of the authors of a custom framework called Darknet. After 5 years of research and development to the 3rd generation of YOLO (YOLOv3), Joseph Redmon announced his withdrawal from the field of computer vision and discontinued

developing the YOLO algorithm. However, he does not dispute the continuation of research by any individual or organization based on the early ideas of the YOLO algorithm. In April 2020, Alexey Bochkovsky, a Russian researcher and engineer who was also one of the authors of the Darknet framework and builder of the 3 previous YOLO architecture on C-based language, has cooperated with Chien Yao and Hon-Yuan and published "YOLOv4: Optimal Speed and Accuracy of Object Detection". The implementation related to this paper was released on GitHub Darknet Repository. It was built upon the same idea with the other versions, and was more accessible and easier to use, especially in model training with good pre-trained models. There was also a library called "yolov4" in Python, but it was not yet stable for custom training, which was needed in this project, so related code frame is cloned from GitHub repository instead of directly using the library.

3.1.1.2. Object Detector Architecture

Object detectors typically compose of several components, which are the:

- **Input –** this is where the input image is taken
- **Backbone –** Which refers to the network which takes as input the image and extracts the feature map.
- **Neck –** Neck and head are sub-sets of the backbone, and the neck serves as an enhancer of the feature discriminability and robustness.
- **Head –** Which handles the prediction. This can be either using a one stage detector for dense prediction like Yolov4 or a two-stage detector also known as Sparse Prediction - with FRCNN and Mask RCNN, as seen in Figure 1.

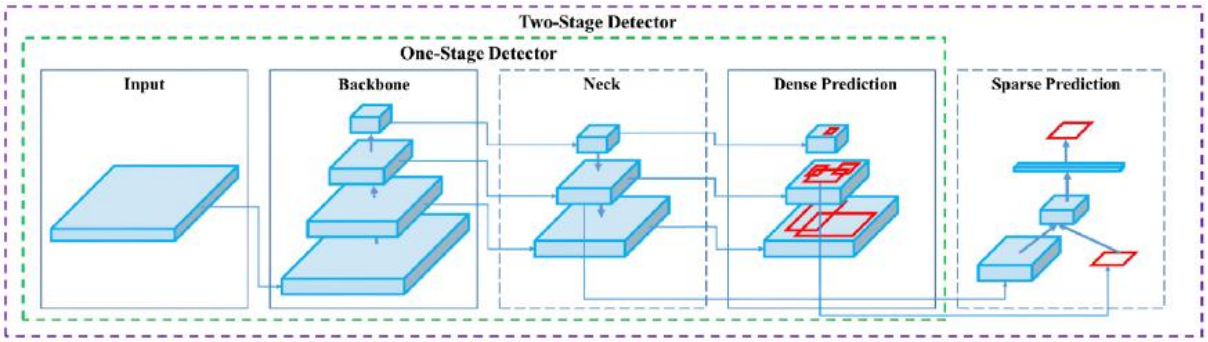

Figure 1: Detector Schema

In Yolov4 architecture, the chosen models for these components are as follows:

- **Backbone -** There was a choice Between CSPResNeXt50, CSPDarknet53, and EfficientNet B3; based on theoretical justification and several experiments

CSPDarknet53 neural network (can be seen in Figure 2) was shown to be the most optimal model (Bochkovskiy et al., 2020). Also, other optimizations to the training method are performed to yield better accuracy without increasing the inference cost (referred as "bag of freebies"). Moreover, authors introduced some set of modules that only increase the inference cost by a small amount but significantly improve the accuracy of object detection (referred as "bag of specials").

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

Figure 2: Darknet-53 Layers

- **Neck** - SPP (Spatial pyramid pooling) block over the CSPDarknet53 is utilized, since it significantly increases the receptive field, separates out the most significant context features and causes almost no reduction of the network operation speed (Bochkovskiy et al., 2020). They also used PANet (Path Aggregation Network, can be seen in Figure 3) as the method of parameter aggregation from different backbone levels for different detector levels, instead of the FPN (Feature Pyramid Network) used in YOLOv3 (Bochkovskiy et al., 2020).

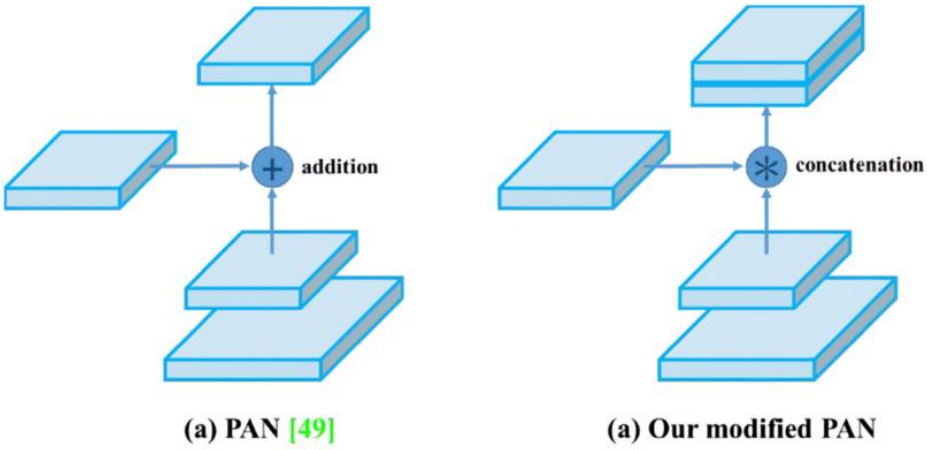

Figure 3: Modified PANet explanation

- **Head** - The dense prediction is the final prediction composed of a vector containing the predicted bounding box coordinates (center, height, width), the prediction confidence score, and the probability classes. YOLOv4 deploys the identical head as YOLOv3 for detection with the anchor-based detection steps, and three levels of detection granularity (Thuan, 2021).

3.1.1.3. License Plate Detection

As mentioned earlier, it was aimed to train a custom Yolov4 model using a car plate dataset in Kaggle and then detect the plate locations in a frame with this model for further processing. For this aim, we used Yolov4's train option with custom written configuration files and a pre-trained model (yolov4.conv.137) for increasing speed. Related implementation can be found in Appendix "License Plate Detection with YOLOV4 – Training YOLOV4". After training is successfully done, the weights file is saved to our drive folder, under "backup". This newly generated weights file is used for license plate detection, and the results are saved in a txt file. The implementation for this extraction is present in Appendix "License Plate Detection with YOLOV4 – Extracting the License" section. By reading the coordinates of the detections performed by our model, we cropped the license plate(s) for further processing. The results can be seen Figure 4 and 5.

Figure 4: Detection results

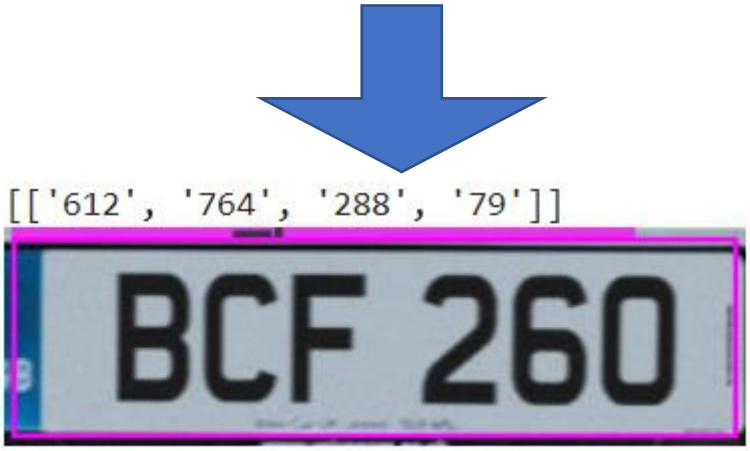

Figure 5: Cropped result of detection

3.1.2. Detection without Yolov4

In the first step of this detection implementation, the image is read by using cv2.imread function and preprocessing is performed.

Figure 6: Original Image

As a preprocessing, the image is converted to gray scale and Gaussian filter which uses 5x5 as a kernel size and has its border at 0.

Figure 7: Gray-scaled Image

As it is known, there are 4 filter options that can be used in noise smoothing; these are box filter, gaussian filter, median filter and sigma filter. Gaussian filter is more successful compared to box filter since it gives more weight to the center and the weight are given to the neighboring pixel decrease as it is moved away from the center of the window. Median and sigma filters are suitable for the images with salt and pepper noise. Gaussian filter is chosen since it is the one that most appropriate for most of the images.

Figure 8: Gaussian Filter applied version

After removing the outliers, edge detection is done by using Canny edge detection method. This algorithm is called as optimal edge detector, thanks to its thresholding approach. This thresholding approach especially differs it from the other techniques. It uses two threshold values and label the pixels as weak or strong edge. If a pixel passes the low threshold and it is near to strong edges, it is taken as weak edges. Thanks to this, it finds all edges better than other edge detectors. The minimum threshold is given as 150 and maximum threshold is 200 in the function.

Figure 9: Canny Edge Detector

The preprocessing step is completed with these, the function that is written and called as “preprocessTheImage” is shown in Appendix part “License Plate Detection without YOLOV4” section.

In the function that is called as “extractTheLicense”, the contours in the edge detected image is found with findContours function of OpenCV. findContours detects the changes in the image colors and label them as contours. It takes 3 input parameters: image, retrieval mode and contour approximation method. There are 4 options for retrieval modes, these are RETR_EXTERNAL, RETR_LIST, RETR_CCOMP and RETR_TREE. While RETR_EXTERNAL finds only the outer contours, RETR_CCOMP finds inner and outer contours separately. RETR_TREE calculates the full hierarchy of the contours whereas RETR_LIST gives all contours without considering the hierarchy. Since we just want to detect contours, cv2.RETR_LIST given as the retrieval mode. As a contour approximation method, there are two options: CHAIN_APPROX_NONE which stores all the contours points and CHAIN_APPROX_SIMPLE that stores only the endpoints of the contour lines (Shaikh, 2020). Since CHAIN_APPROX_SIMPLE saves memory because it stores a smaller number of points, it is chosen as contour approximation method.

Figure 10: All contours detected with findContours

The contours are sorted according to their areas and the contours with area equal to 30 and greater than 30 are taken into consideration. This is limited to 30 to reduce the number of contours examined in the for loop and not accidentally select the contour area, which is a small rectangle.

Figure 11: The contours that has area greater than 30

After getting the selected contours, it is tried to find a rectangular area by iterating for loop for each contour. This is done with the help of arcLength and approxPolyDP functions of cv2. arcLength calculates the perimeter of the contour, if "True" is given as argument to this function, it returns the closed contour perimeter. approxPolyDP takes this peripheral value as argument and calculates how many edges this closed area has. If it is equal to 4, it means that the rectangle area which belongs to the license plate is found. The coordinate information is stored, and the image is cropped in accordance with the coordinates. The stored coordinates will be used to put the result of the reading of plate on the image. For this reason, this function returns cropped image and coordinates. The function is put in the Appendix part "License Plate Detection without YOLOV4".

Figure 12: License Plate Detection

Figure 13: Cropped Image

There are some restrictions in this detection application. Since the detection is based on the contours and the area of the rectangle plays an important role in finding a rectangle, it cannot

perform license detection on a vehicle photograph taken from a distance. It also cannot perform plate detection when there is more than one car in the same image. For these reasons, it is more convenient to use yolov4 for license plate detection.

3.2. Preprocessing for Segmentation

After performing the license plate detection with or without yolov4, the next step is to perform processing to prepare the license plate for segmentation.

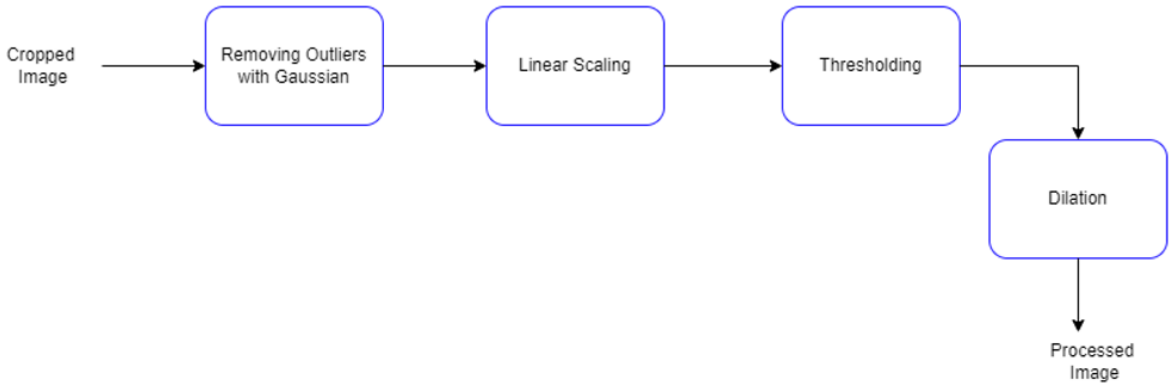

Figure 14: Processing stages

As a first step Gaussian filter is applied with 5x5 kernel size in the function called gaussFilter. This function returns the gaussian filtered version of the license plate. Gaussian filter is chosen because it is most convenient filtering for all images compared to other noise removal methods. While box filter is less successful than Gaussian filter because it gives same weight to each pixel, sigma and median filters are more appropriate for the images that include salt and pepper noise. This step is not performed after the detection without yolov4 method since the image is already filtered with Gaussian in detection stage.

Figure 15: Gaussian filtered License Plate - car2.jpg

In the second step, it is chosen to implement linear scaling. Linear scaling is done in order to spread the distribution of the pixels. Linear scaling is a point operator that maps the histogram values of the image (umin- umax) to [0, Gmax] where Gmax is equal to 255. The functions that are used to transform an image to a new image is called gradation functions. The gradation function for linear scaling:

$$g(u) = b(u + a) \;\; a = -u_{min} \;\; b = \frac{G_{\max}}{u_{max} - u_{min}}$$

The change in the histogram after performing linear scaling is observed in the following figure 16:

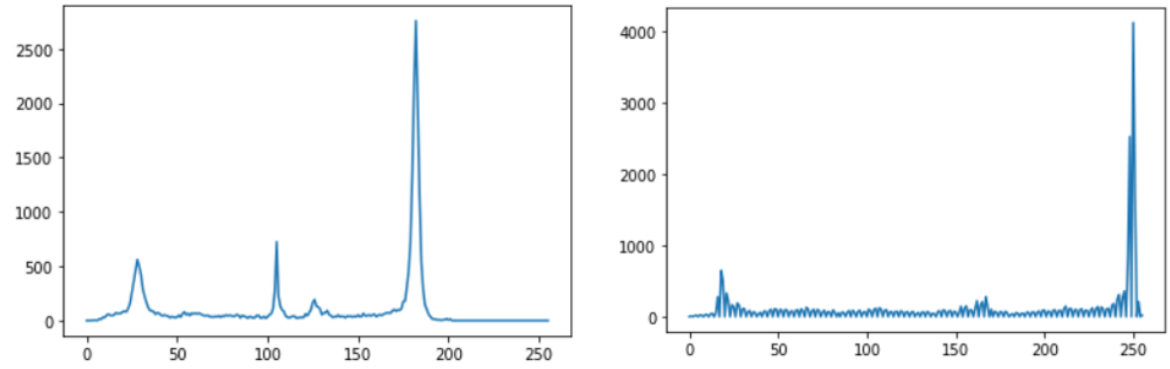

Figure 16: Before vs After Linear Scaling

After linear scaling, it is desired to convert the image to a binary image. Firstly, characters are made black, and the remaining part is left as white. In order to make the characters more visible and bigger, erosion is applied. Erosion reduces the brightness of the bright objects. Its effect can be seen from the following:

Figure 17: Thresholding (threshold=100)

Figure 18: Erosion

These implementations are done in the functions that named as thresh and erosion. While researching how binarization is performed before findContours in such implementation, it is observed that the image is binarized such that the characters are white and remaining part is black because findContours can find white objects more successfully. This is important factor for the implementation since findContours will be used in the segmentation section. For this reason, pixels with an intensity value greater than 100 were made black and the others were made black. After that dilation is applied. Dilation is the opposite of erosion; it increases the brightness of the objects.

Figure 19: Thresholding

Figure 20: Dilation

However, the threshold value that determined by us beforehand was not suitable for every image, it is seen that it is a problem for the recognition section. At this point, we have to find a way to automatically determine the threshold.

One way of achieving automatic thresholding is called as Otsu's method. This method was proposed by Nobuyuki Otsu. The algorithm tries to find the value that has minimum within-class variance which means the weighted sum of variances of two classes, the part that has lower intensity than threshold (background) and other part that has higher intensity (foreground). The algorithm iteratively calculates this variance for every intensity value with the following formula:

$$\sigma^2(t) = w_{bg}(t)\sigma^2_{bg}(t) + w_{fg}(t)\sigma^2_{fg}(t)$$

where $w_{bg}(t) = \frac{P_{bg}(t)}{P_{all}}$ *and* $w_{fg}(t) = \frac{P_{fg}(t)}{P_{all}}$

$$\sigma^2(t) = \frac{\sum(x_i - \mu)}{N-1}$$

P_{all} : The total count of pixels in the image

P_{BG} (t): the count of background pixels at threshold t

P_{FG} (t): the count of foreground pixels at threshold t

x_i : the value of pixel at i in the group

μ: the means of pixel values at the group

N: number of pixels

The threshold value, which has a minimum within-class variance, is selected as threshold and binarization is performed accordingly with Otsu's method ("Otsu's method for image

thresholding explained and implemented – Muthukrishnan", 2020). The figures 21 and 22 can be given as examples where manual thresholding gives bad result and Otsu gives better result.

Figure 21: Bad result from Manual Thresholding

Figure 22: Thresholding with Otsu

The function that is mentioned in this section is present in Appendix “Processing of License Plate” section.

3.3. Segmentation of the Characters and Applying OCR

Two approaches used here interchangeably, since depending on the properties of the images, sometimes our preferred method with histograms caused an error, especially when the characters are too close to each other. The reason behind the preference is explained in the “Discussion” section.

3.3.1. Histogram Approach for Segmentation

In this approach, we implemented below algorithm to obtain characters one by one.

The steps of algorithm:

1)Calculate the sum for each row and for each column separately by using cv2.reduce

2)Take width (img.shape[1]) and height (img.shape[0])

3)Get the averages of each column and row by dividing sums with width or height

4)Form an array sized to columns (X_temp) and rows (Y_temp) and fill it with 0's and 1's

4.1) Put 0 where the average sum is less than 20 and put 1 otherwise

5)Loop over rows and columns in separate loops and store the indices where there is a change from 0 to 1 or 1 to 0

6)If the distance between the adjacent indices is greater than 5 and the value of that index is equal to 1 in column array (X_temp), cut the image accordingly and put it into a list

In step 6, only the columns are checked since the characters can be separated only by their width in regular license plates, because generally all characters have the same height. An example result can be seen in Figure 23.

Figure 23: Extracted Characters with Histogram Approach

Figure 24: Cropped License

3.3.2. Segmentation with Contours

As explained in section “Detection without Yolov4” there is a built-in function for finding contours in an image, findContours. In order to utilize this function for extracting characters, we performed some checks on all contours that are found by findContours.

First, we checked if the ratio of the height of the cropped image to the height of the contour is more than 6. This check is also performed for width, with the threshold 15. The ratio of height versus width of the contour is also calculated and checked to see if it is more than 1.5. Lastly, the area of the contour is checked to be more than 100 pixels. These thresholds are determined and optimized after it is compared with other values selected with a binary search approach.

After the checks are done, the character is cropped and appended into a list. An example results can be seen in Figure 26.

Figure 25: Cropped License

Figure 26: Extracted Characters with FindContours

3.3.3. OCR Implementation

For Optical Character Recognition, there are several libraries that can be used in Python such as Pytesseract, Keras-OCR and EasyOCR. While planning the project, it is decided to use Pytesseract as OCR library. In the implementation, we firstly used Pytesseract. However, it is seen that it gives some errors especially in numbers. After observing that we have found EasyOCR. When EasyOCR library is used, we have seen that it gives more accurate result for converting license plate to text. This will be discussed in detail in the discussion section.

We switched to EasyOCR by looking at the analyzes in the referenced page. When EasyOCR and Pytesseract are compared with each other, it is seen that while EasyOCR is more

successful to convert numbers than Pytesseract, it is worse in terms of recognizing alphabets (Liao, 2020). Since the number of letters used on the plates is less than the number of numbers, easyOCR has been tried.

The figure 27 is an example to show how EasyOCR gives the results.

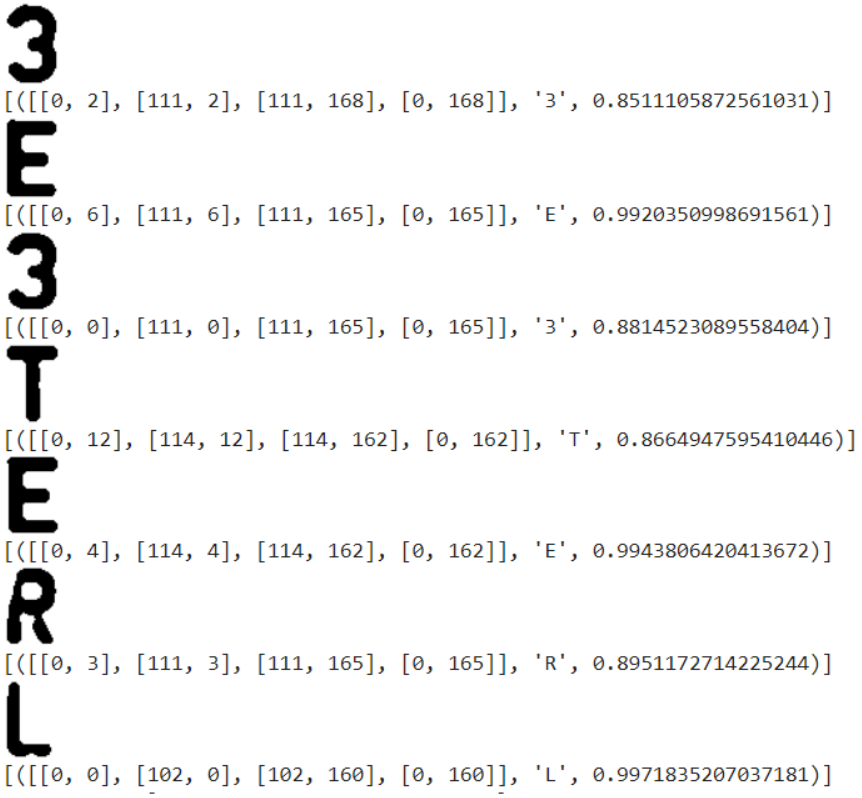

Figure 27: EasyOCR results

In some cases, the segmentation with contours and histogram cannot parse the whole characters depends on the image. In these cases, it is checked that the number of parsed characters is at least equal to 5, if it is less than 5 the license plate is given directly to the EasyOCR. It should also be noted that easyOCR works more accurately when it receives characters individually rather than as a whole. In addition to that, we checked the validity of found characters by easyOCR by looking whether found character is alphanumeric or not. These are performed in the function called as findChar. All the functions mentioned in this part is located in Appendix “Segmentation and OCR Implementation” section.

3.4. Results

In this section the results that is taken from the implementations will be represented. The processes will be shown step by step.

3.4.1. Results from the implementation done without yolov4

- Steps for image “car1.PNG”

Figure 28: Gray-scaled

Figure 29: Gaussian Filtered

Figure 30: Canny Edges

Figure 31: Selected Contours

Figure 32: Cropped Image

Figure 33: Linear Scaled Image

Figure 34: Otsu's Thresholding

Figure 35: Result - LRJJTEE

- Steps for image “car3.PNG”

Figure 36: Gray-scaled

Figure 37: Gaussian Filtered

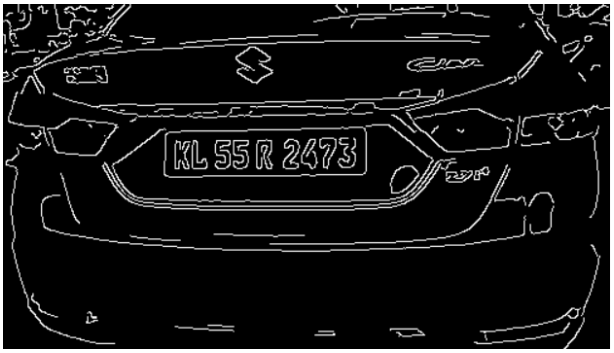

Figure 38: Canny Edges

Figure 39: Selected Contours

Figure 40 Cropped Image

Figure 41: Linear Scaled

Figure 42: Otsu's Thresholding

Figure 43: Result -KLSSR2473

3.4.2. Results from the implementation done with yolov4

- Steps for image “car2.jpg”

Figure 44: Cropped Image

Figure 45: Gaussian Filtered

Figure 46: Linear Scaled

Figure 47: Otsu's Thresholding

Figure 48: Result- BCF 260

- Steps for image “img7.jpeg”

Figure 49: Cropped Image

Figure 50: Gaussian Filtered

Figure 51: Linear Scaled

Figure 52: Otsu's Thresholding

Figure 53: Result- 81 FA 077

- Steps for image “car_2.jpeg” – two cars in one image

Figure 54: Cropped Image for the first car

Figure 55: Gaussian Filtered for the first car

Figure 56: Linear Scaled for the first car

Figure 57: Otsu's Thresholding for the first car

Figure 58: Cropped Image for the first car

Figure 59: Gaussian Filtered for the first car

Figure 60: Linear Scaled for the first car

Figure 61: Otsu's Thresholding for the first car

Figure 62: Result - TR 81 AAE 817 and PNII UNF

- Result for image “fourcars.jpeg” – four cars in one image

Figure 63: Result - IK33 PIT and R574 INS and MY97 ATE and RE69 LTE

As it can be seen from the result, EasyOCR successfully converts character images to text most of the time. There are a few errors in converting the image it to text, for example, it converts 3 as J in Figure 35 and 5 as S in Figure 43.

Also, this implementation can be performed on videos. In the following link, you can see the result of the sample video.

Video Link:
https://drive.google.com/file/d/1-6v3yV6CcE9l78ji8YDeqBi3MQx0HyY0/view?usp=sharing

4. Discussion

As it is mentioned in the license plate detection without yolov4 section, this implementation has some limitations compared to the detection that is performed with yolov4 training. Since it cropped the license plate considering to the contours and it takes the one that has four corners, there should be only one visible rectangle shaped object in the image. For this reason, when the image of car is taken from far away, it fails to crop the license part since it finds another contour. Due to this fact, whether this implementation works or not depends on how the image was captured. The second disadvantage of it is that it is not capable of finding more than one car's license plate. It can find only one license plate in one image. Due to these factors, using yolov4 after training it for license plate detection is much more useful and appropriate for the project. Thanks to YOLOV4 training, the license plate recognition can be performed for videos too; if this is tried to be done without YOLOV4, this will take so much time.

In the processing section, it is mentioned that after applying Otsu's thresholding we have decided to apply dilation to increase the brightness of the objects, it helps use to get bigger sized characters. However, dilation could not provide us the help we expected in segmentation section. On license plate with too many characters, the characters are combined with each other, and this prevent us to perform segmentation correctly. For this reason, it is decided to give Otsu's thresholding version to the segmentation section.

Figure 64: Comparison for dilation problem

It is mentioned in the OCR Implementation section that with the same given characters easyOCR gives more correct results than Pytesseract. Pytesseract works well on high quality images. Since the image quality is not guaranteed while performing license plate recognition, it is better to use EasyOCR. Below, some examples are given to compare the performances of pytesseract and EasyOCR.

```
Pytesseract result:  81 FA 077)
EasyOCR result:  81 FA 077
```


```
Pytesseract result:  mO1 AAE 81
EasyOCR result:  tr 81 AAE 817
```

In the segmentation section, contours of some characters are not enough to detect the characters; for this reason, histogram approach is also added to segmentation part. However, when the characters are close each other some of characters gives error. In order to overcome this issue, we combined these two methods to segment the characters.

Figure 65: License Plate

Figure 66: Segmentation with Contours

Figure 67: Segmentation with Histogram

As it is seen from the figure 66 and 67; while contours cannot find all characters, it could not find “I”, the segmentation that is performed by following the changes in the intensities can segment all the characters successfully.

You can find all the images that is tested with the implementations, training files and codes in the zip file that is submitted in Sucourse+ and also you can find the drive link of the folder that includes these in below.

Drive folder link:
https://drive.google.com/drive/folders/1j5SEJgQgR9a5vNFB1SrlsegcYaZezVty?usp=sharing
The dataset link used for training:
https://www.kaggle.com/andrewmvd/car-plate-detection

5. References

Bochkovskiy, A., Wang, C., & Liao, H. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv.org. Retrieved 10 January 2022, from https://arxiv.org/abs/2004.10934.

Liao, C. (2020). OCR Engine Comparison—Tesseract vs. EasyOCR. Retrieved 10 January 2022, from https://medium.com/swlh/ocr-engine-comparison-tesseract-vs-easyocr-729be893d3ae

Otsu's method for image thresholding explained and implemented – Muthukrishnan. (2020). Retrieved 9 January 2022, from https://muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/

Overview of the YOLO Object Detection Algorithm. (2018). Retrieved 8 January 2022, from https://odsc.medium.com/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv.org. Retrieved 10 January 2022, from https://arxiv.org/abs/1506.02640v1.

Rosebrock, A. (2022). Getting started with EasyOCR for Optical Character Recognition - PyImageSearch. Retrieved 8 January 2022, from https://www.pyimagesearch.com/2020/09/14/getting-started-with-easyocr-for-optical-character-recognition/

Shaikh, R. (2020). OpenCV (findContours) Detailed Guide. Retrieved 9 January 2022, from https://medium.com/analytics-vidhya/opencv-findcontours-detailed-guide-692ee19eeb18

Thuan, D. (2021). *Evolution of yolo algorithm and yolov5: the state-of-the-art object detection algorithm.* Retrieved 10 January 2022, from https://www.theseus.fi/handle/10024/452552

6. Appendix

6.1. License Plate Detection with YOLOV4

6.1.1. Training YoloV4

```
[ ]  import os
     def arrangeTxt(name):
         image_files = []
         os.chdir(os.path.join("data", name))
         for filename in os.listdir(os.getcwd()):
             if filename.endswith(".jpg"):
                 image_files.append("data/" + name +"/" + filename)
         os.chdir("..")
         if name=="train":
             with open("train.txt", "w") as outfile:
                 for image in image_files:
                     outfile.write(image)
                     outfile.write("\n")
                 outfile.close()
             os.chdir("..")
         elif name=="test":
             with open("test.txt", "w") as outfile:
                 for image in image_files:
                     outfile.write(image)
                     outfile.write("\n")
                 outfile.close()
             os.chdir("..")
         else:
             with open("valid.txt", "w") as outfile:
                 for image in image_files:
                     outfile.write(image)
                     outfile.write("\n")
                 outfile.close()
             os.chdir("..")
```

```
[ ]  !cp /content/gdrive/MyDrive/EE417/ee417-obj.cfg ./cfg
     !cp /content/gdrive/MyDrive/EE417/obj.names ./data
     !cp /content/gdrive/MyDrive/EE417/obj.data  ./data
```

```
[ ]  !ls data/
```

```
9k.tree     eagle.jpg    imagenet.labels.list      obj.names         voc.names
coco9k.map  giraffe.jpg  imagenet.shortnames.list  openimages.names
coco.names  goal.txt     labels                    person.jpg
dog.jpg     horses.jpg   obj.data                  scream.jpg
```

```
[ ] !./darknet detector train data/obj.data cfg/ee417-obj.cfg yolov4.conv.137 -dont_show -map
```

```
Streaming output truncated to the last 5000 lines.
 total_bbox = 396572, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.829581), count: 9, class_loss = 1.253952, iou_loss = 9.124035, total_loss = 10.377987
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.704978), count: 11, class_loss = 1.178283, iou_loss = 2.473377, total_loss = 3.651660
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.309149, iou_loss = 0.000000, total_loss = 0.309149
 total_bbox = 396592, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.842271), count: 6, class_loss = 0.298259, iou_loss = 5.034066, total_loss = 5.332325
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.752971), count: 10, class_loss = 0.257628, iou_loss = 2.784932, total_loss = 3.042561
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.472285), count: 2, class_loss = 0.066473, iou_loss = 0.027278, total_loss = 0.093751
 total_bbox = 396610, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.784128), count: 6, class_loss = 1.099326, iou_loss = 8.008502, total_loss = 9.107828
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.768533), count: 6, class_loss = 0.472592, iou_loss = 1.280005, total_loss = 1.752596
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.621416), count: 3, class_loss = 0.150729, iou_loss = 0.106705, total_loss = 0.257434
 total_bbox = 396625, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.851105), count: 2, class_loss = 0.111326, iou_loss = 2.394600, total_loss = 2.505926
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.723161), count: 8, class_loss = 0.901396, iou_loss = 1.707427, total_loss = 2.608824
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.634979), count: 4, class_loss = 0.285179, iou_loss = 0.061163, total_loss = 0.346342
 total_bbox = 396639, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.767199), count: 10, class_loss = 0.306406, iou_loss = 13.545672, total_loss = 13.852078
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.799042), count: 5, class_loss = 0.512382, iou_loss = 2.067921, total_loss = 2.580303
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.000033, iou_loss = 0.000000, total_loss = 0.000033
 total_bbox = 396654, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.662033), count: 12, class_loss = 2.209750, iou_loss = 13.098273, total_loss = 15.308023
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.697210), count: 8, class_loss = 1.421515, iou_loss = 1.546907, total_loss = 2.968423
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.000000), count: 1, class_loss = 0.245943, iou_loss = 0.000000, total_loss = 0.245943
 total_bbox = 396674, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.692545), count: 14, class_loss = 2.520789, iou_loss = 11.801228, total_loss = 14.322017
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.734258), count: 13, class_loss = 1.848274, iou_loss = 3.057711, total_loss = 4.905986
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.499608), count: 1, class_loss = 0.007138, iou_loss = 0.020049, total_loss = 0.027187
 total_bbox = 396702, rewritten_bbox = 0.000000 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.653829), count: 18, class_loss = 2.836256, iou_loss = 23.105938, total_loss = 25.942194
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.722535), count: 18, class_loss = 0.902270, iou_loss = 3.155318, total_loss = 4.057588
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.680216), count: 6, class_loss = 0.161148, iou_loss = 0.206797, total_loss = 0.367944
 total_bbox = 396744, rewritten_bbox = 0.000000 %
```

6.1.2. Extracting the results

```
[ ]  def readResult():
       allLines=[]
       with open("result.txt") as f:
           lines = f.readlines()
           allLines.append(lines)

       count=0
       licenseInfo=[]
       for i in range(len(allLines[0])):
         count=count+1
         if (count>14):
             licenseInfo.append(allLines[0][i].strip())
       licenseCoordinates=[]
       for i in range(len(licenseInfo)):
         #print(licenseInfo[i])
         temp=licenseInfo[i].split("\t")
         #print(temp)
         info=temp[1].split()
         #print(info)
         tmp_coord=[]
         tmp_coord.append(info[1])
         tmp_coord.append(info[3])
         tmp_coord.append(info[5])
         tmp_coord.append(info[7].split(")")[0])


         licenseCoordinates.append(tmp_coord)
       #print(licenseCoordinates)
       return licenseCoordinates
```

```
[ ]  def getCoordinates(licenseCoordinates, img):
       crop_gray_images = []
       coord = []
       for i in range (len(licenseCoordinates)):
         x1 = int(licenseCoordinates[i][0])
         y1 = int(licenseCoordinates[i][1])
         w1 = int(licenseCoordinates[i][2])
         h1 = int(licenseCoordinates[i][3])

         crop_img = img[y1-4:y1+h1+4, x1-4:x1+w1+4]
         #cv2_imshow(crop_img)
         gray = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)
         crop_gray_images.append(gray)
         coord.append((x1+w1,y1+h1))
         # hist = cv2.calcHist([gray],[0],None,[256],[0,256])
         # plt.plot(hist)
         # plt.show()
       return crop_gray_images, coord
```

6.2. License Plate Detection without YOLOV4

```
[ ]  def preprocessTheImage(img_name):
       #read the image and convert to gray scale
       img = cv2.imread(img_name,cv2.IMREAD_COLOR)
       img = imutils.resize(img, width=500 )
       gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
       cv2_imshow(gray_img)

       #reduce noise - Gaussian filter
       gray_img = cv2.GaussianBlur(gray_img,(5,5),0)
       cv2_imshow(gray_img)

       #Perform Edge detection with Canny
       cannyEdges = cv2.Canny(gray_img, 150, 200)
       cv2_imshow(cannyEdges)

       return img,cannyEdges
```

```
[ ]  def extractTheLicense(img,cannyEdges):
       #find contours
       contours,_ = cv2.findContours(cannyEdges.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
       img_copy=img.copy()
       cv2.drawContours(img_copy,contours,-1,(0,255,0),3)
       cv2_imshow(img_copy)

       contours = sorted(contours, key = cv2.contourArea, reverse = True)[:30]

       img2 = img.copy()
       cv2.drawContours(img2,contours,-1,(0,255,0),3)
       cv2_imshow(img2) # contours
       new_img=[]
       coord=[]
       for c in contours:
             peri = cv2.arcLength(c, True)
             approx = cv2.approxPolyDP(c, 0.02 * peri, True)
             # print ("approx = ",approx)
             if len(approx) == 4:  # Select the contour that has 4 corners
                 LicensePlate = approx
                 x, y, w, h = cv2.boundingRect(c)
                 coord.append((x+w,y+h))
                 new_img = img[y:y + h, x:x + w] #Crop the new image
                 break

       cv2.drawContours(img, [LicensePlate], -1, (0,255,0), 3)
       cv2_imshow(img)
       return new_img,coord
```

6.3. Processing of License Plate

```
[ ]  def gaussFilter(gray):
       imGaussFiltered = cv2.GaussianBlur(gray,(5,5),0)
       return imGaussFiltered
       #cv2_imshow(imGaussFiltered)
```

```
[ ]  def linScale(crop_img):
       if crop_img.ndim==3:
         crop_img = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)
       img = np.double(crop_img)

       min = np.amin(img)
       max = np.amax(img)

       alpha = -1*min
       Gmax = 255
       beta = Gmax/(max-min)
       imLinScaled = (img+alpha)*beta

       imOut = np.uint8(imLinScaled)
       return imOut

     # imOut = linScale(imGaussFiltered)
     # cv2_imshow(imOut)
     # hist = cv2.calcHist([imOut],[0],None,[256],[0,256])
     # plt.plot(hist)
     # plt.show()
```

```
[ ]  def thresh(imOut):
       th, im_th = cv2.threshold(imOut, 40, 255, cv2.THRESH_BINARY)
       return im_th
       #cv2_imshow(im_th)
```

```
[ ]  def erosion(im_th):
       kernel=cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
       #apply erosion to make region clearer
       erodedVersion=cv2.erode(im_th,kernel,iterations=1)
       return erodedVersion
       #cv2_imshow(erodedVersion)
```

```
[ ]  def convertWhite(imOut):
       h,w=imOut.shape
       img = np.double(imOut)
       imgBinarization=np.zeros((h,w))
       for i in range(h):
         for j in range(w):
           if imOut[i][j]>100:
             imgBinarization[i][j]=0
           else:
             imgBinarization[i][j]=255
       imgBinarization=np.uint8(imgBinarization)
       return imgBinarization

     # imgBinarization=convertWhite(imOut)
     # cv2_imshow(imgBinarization)
```

```
[ ]  def dilation(imgBinarization):
       kernel=cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
       #apply dilation to make region clearer
       dilatedVersion=cv2.dilate(imgBinarization,kernel,iterations=1)
       return dilatedVersion
       #cv2_imshow(dilatedVersion)
```

```
[ ]  def otsu(imOut):
       ret, thresh = cv2.threshold(imOut, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
       cv2_imshow(thresh)
       return thresh
```

6.4. Segmentation and OCR Implementation

```
[ ]  def histogramApproach(img):

        characters = list()
        np_img = np.array(img)

        # calculate the summation for each row and col
        X_summation = cv2.reduce(np_img, 0, cv2.REDUCE_SUM, dtype=cv2.CV_32S)
        Y_summation = cv2.reduce(np_img, 1, cv2.REDUCE_SUM, dtype=cv2.CV_32S)

        # rotate the vector for computations
        X_summation = X_summation.transpose()
        width = img.shape[1]
        height = img.shape[0]

        # calculate the average
        X_sum_avg = X_summation / height
        Y_sum_avg = Y_summation / width

        # convert them to np arrays
        X_temp = np.array(X_sum_avg)
        Y_temp = np.array(Y_sum_avg)

        # convert to zero small details and others 1
        X_temp[X_temp < 20] = 0
        X_temp[X_temp > 20] = 1
        Y_temp[Y_temp < 20] = 0
        Y_temp[Y_temp > 20] = 1
```

```
    #store the indices where we go from 0 to 1 and 1 to 0

    comp1 = X_temp[0]
    temp1 = list()
    temp2 = list()
    for i in range(X_temp.size):
        if X_temp[i] != comp1:

            comp1 = X_temp[i]
            temp1.append(i)
    horizontal = np.array(temp1)


    comp2 = Y_temp[0]
    for i in range(Y_temp.size):
        if Y_temp[i] != comp2:

            comp2 = Y_temp[i]
            temp2.append(i)
    vertical = np.array(temp2)

    # calculate the height
    rectv = []
    rectv.append(vertical[0])
    rectv.append(vertical[1])
    max = int(vertical[1]) - int(vertical[0])
    for i in range(len(vertical) - 1):
        difference = int(vertical[i + 1]) - int(vertical[i])
        if difference > max:
            rectv[0] = vertical[i]
            rectv[1] = vertical[i + 1]
            max = difference
```

```
    #calculate the width and crop it
    for i in range(len(horizontal) - 1):
        diff = int(horizontal[i + 1]) - int(horizontal[i])

        if (diff > 5) and (X_temp[horizontal[i]] == 1):
            characters.append(np_img[int(rectv[0])-5:int(rectv[1])+5, horizontal[i]-5:horizontal[i + 1]+5])
            cv2.rectangle(img, (horizontal[i]-5, rectv[0]-5), (horizontal[i + 1]+5, rectv[1]+5), (0, 255, 0), 1)

    image = plt.imshow(img)
    plt.show(image)

    return characters
```

```
[ ]  def findCont(processedIm):
       contours, hierarchy = cv2.findContours(processedIm, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

       # sort contours
       #contours = sorted(contours, key=cv2.contourArea, reverse=True)

       # loop through contours and find letters in license plate
       selected_contours = []
       for cnt in contours:
           x,y,w,h = cv2.boundingRect(cnt)
           height, width = processedIm.shape

           # if height of box is not more than 6 pixels skip
           if height / float(h) > 6: continue
           ratio = h / float(w)
           # if height to width ratio is less than 1.5 skip
           if ratio < 1.5: continue
           # if width is not more than 15 pixels skip
           if width / float(w) > 15: continue
           area = h * w
           # if area is less than 100 pixels skip
           if area < 100: continue

           crp = processedIm[y-3:y+h+3, x-3:x+w+3]
           crp = cv2.bitwise_not(crp)
           crp = cv2.medianBlur(crp, 5)
           selected_contours.append(crp)
       selected_contours = selected_contours[::-1]
       return selected_contours
```

```
[ ]  def isValidChar(txt):
       result=""
       for i in range(len(txt)):
         if (txt[i]==" "):
           result=result+" "
         elif (txt[i].isalnum()):
           result=result+txt[i]
       return result
```

```
[ ]  def findChar(processedIm):
       try:
         contours = histogramApproach(processedIm)
       except:
         contours = findCont(processedIm)
       reader=easyocr.Reader(['en'],gpu=True)
       plate_num = ""
       # loop through contours and find letters in license plate
       for cnt in contours:
         if (cnt is not None and len(cnt)!=0):
           cv2_imshow(cnt)
           cnt = cv2.resize( cnt, None, fx = 3, fy = 3, interpolation = cv2.INTER_CUBIC)
           cnt= cv2.cvtColor(cnt, cv2.COLOR_GRAY2RGB)
           text=reader.readtext(cnt)
           #text = pytesseract.image_to_string(cnt, config='-c tessedit_char_whitelist=0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ --psm 10 --oem 3')
           if (len(text)!=0):
             print(text)
             plate_num += text[0][1].strip()
       #print(plate_num, end=" ")
       if(len(plate_num))<5:
         #cv2_imshow(processedIm)
         text=reader.readtext(processedIm)
         text_easyocr=""
         for res in text:
            text_easyocr=text_easyocr+res[1]+" "
         text_easyocr = isValidChar(text_easyocr)
         return text_easyocr
       else:
        plate_num = isValidChar(plate_num)
        return plate_num
```

```
[ ]  def writeOnImage(img, coor, number):
       font = cv2.FONT_HERSHEY_SIMPLEX

       # org
       org = (coor[0], coor[1])

       # fontScale
       fontScale = 1

       # Blue color in BGR
       color = (255, 0, 0)

       # Line thickness of 2 px
       thickness = 2

       # Using cv2.putText() method
       if(len(number) == 0):
         image = cv2.putText(img, "not found", org, font,
                             fontScale, color, thickness, cv2.LINE_AA)
       else:
         image = cv2.putText(img, number, org, font,
                             fontScale, color, thickness, cv2.LINE_AA)

       return image
```

```
[ ] def detect():
      !./darknet detector test data/obj.data cfg/ee417-obj.cfg /content/gdrive/MyDrive/EE417/backup/ee417-obj_last.weights frame.jpg -thresh 0.7 -ext_output > result.txt -dont_show
      pred = cv2.imread("predictions.jpg")
      #cv2_imshow(pred)
      results = readResult()
      gray_crop_images, coords = getCoordinates(results, pred)
      processed_images = []
      for i in range (len(gray_crop_images)):
        resImg = gaussFilter(gray_crop_images[i])
        resImg = linScale(resImg)
        resImg = otsu(resImg)
        #resImg = dilation(resImg)
        processed_images.append(resImg)
        plate_num = findChar(processed_images[i])
        #print(plate_num)
        pred = writeOnImage(pred, coords[i], plate_num)
      #cv2_imshow(pred)
      return pred
```

The following code is for video processing.

```
[ ]  cap = cv2.VideoCapture("/content/gdrive/MyDrive/EE417/videoTrial/test.mp4")

     # Check if camera opened successfully
     if cap.isOpened() == False:
         print("Error opening video stream or file")

     fps = cap.get(cv2.CAP_PROP_FPS)
     w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
     h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

     save_path = "/content/gdrive/MyDrive/EE417/videoTrial/result1.mp4"
     out = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
     while (cap.isOpened()):
         # Capture frame-by-frame
         ret, frame = cap.read()
         if ret == True:
           #cv2_imshow(frame)
           cv2.imwrite("frame.jpg", frame)
           pred = detect()
           out.write(pred)
           print("SAVED")
         else:
           break
     cap.release()
     out.release()
```