“Vehicle License Plate Recognition and Text Extraction”

A dissertation submitted in the partial fulfillment of the requirements for the award of the degree of

Post Graduation Diploma
in
Big Data Analytics

Submitted by
Rishi Kumar Soni
Roll no. 210810039555

Under the Supervision of
Prof. Narinder Singh Sahni

जवाहरलाल नेहरू विश्वविद्यालय
Jawaharlal Nehru University

School of Computational and Integrative Sciences
Jawaharlal Nehru University
New Delhi - 110067

CERTIFICATE

This is to verify that the research work embodied in the dissertation entitled "***Vehicle License Plate Recognition and Text Extraction***" is the original work the candidate for the partial fulfillment of the requirement for the award of the degree of *Post Graduation Diploma* in *Big Data Analytics* and has been carried out under the supervision of of Prof. Nariender Singh Sahni in the center, School of Computational and Integrative Sciences (SC&IS), Jawaharlal Nehru University, New Delhi India.

Rishi Kumar Soni
(Candidate)
School of Computational and Integrative
Sciences,
Jawaharlal Nehru University,
New Delhi, India

Prof. Narinder Singh Sahni
(Supervisor)
School of Computational and Integrative
Sciences,
Jawaharlal Nehru University,
New Delhi, India

ACKNOWLEDGEMENT

Above all, I would like to express my sincere gratitude to my supervisor *Prof. Narinder Singh Sahni* for giving me this opportunity to build a project which aligns to my field of interest that is Machine Learning combined with Computer Vision. His constant guidance, support, and motivation helped me accomplish this project on time.

I am extremely grateful for the valuable insight provided by *Prof. Ayesha Choudhary* on computer vision technology and related articles. I am also thankful for my senior PhD scholars *Ms. Monika Phougat* and *Ms. Athokpam Langlen Chanu* for their suggestive tips and thoughtful approach which helped me understand many new things.

Last but not least, I would like to express my deepest gratitude to my family, friends and everyone who supported me directly or indirectly throughout this journey.

Rishi Kumar Soni
SC & IS
JNU, New Delhi
Date: 30/08/2022

TABLE OF CONTENTS

ABSTRACT

A number of Machine Learning have been developed. Since 1976 the recognition of license plates (also known as number plates). The current challenge is how to increase the accuracy of an existing model with high-speed input data or to find an entirely new way to solve this issue.

In this model, we have tried number plate localization so that even if the libraries read the number plate information incorrectly we can use cropped number plate images to double-check. Image processing techniques such as binary imaging, and connected component analysis are done by the OpenCV library, For optical character recognition an open source library named *EasyOCR* which supports more than 80 languages.

We have used open-source car images provided by *Kaggle*, one of the most famous open-source repositories for data science. Our proposed model can be divided into three major workflows, first is the preprocessing stage. Here, we have used filtering and noise reduction techniques. In the second stage, we used programming logic and data manipulation techniques to localize the number plate inside the input image. In the final stage, we used optical character recognition libraries to read the text from the croped number plate images.

1. INTRODUCTION

The problem statement “Vehicle License Plate Recognition” is more widely known as Automatic Number Plate Recognition (ANPR).
It was first invented in 1976 in Britain at the Police Scientific Development Branch. In November 2005, in Bradford, UK, the ANPR system first helped solve a murder case and also helped locate the killer. It was initially developed to monitor traffic, look for lost and suspected vehicles, identify the different motor vehicles and help in various investigations.

At present, many European and North American countries are using this technology at scale. Many of these countries are also using ANPR technology combined with radio antennas(similar to RFID technology) to make hassle-free road transportation systems. The radio antennas uniquely identify each vehicle entering and exiting the road, they also help in traffic analysis, unmanned toll collection, vehicle identification, and some other things.

Our country is currently trying to leverage this technology by using ANPR system included with RFID technology. According to Nitin Gadkari, Minister of Road Transport and Highways of India, the Indian highways will become toll-plaza free by 2025. My project aims to initially localize the number plate within the image. Once the number plate location is known the character recognition process will work without a problem. Having said that, it is important to capture a clear number plate image and apply a custom made neural engine based character recognition system. It is important to build a custom neural-network OCR because the characters and digits of a number plate are different from regular fonts used in different languages and scripts.

The future of this is both promising and achievable. From simple Car Parking, Traffic Analysis to enhanced security monitoring at borders, checkpoints and plazas this project can be very useful. Apart from that it can also help in vehicle theft detection and automations at harbor and logistics.

2. RESULT AND DISCUSSION

In this section I am going to thoroughly explain this project step by step. Starting from the libraries used, steps taken, logic implemented to the result. I shall further show the result along with the accuracy and then show some statistics. Please note that the term ‘License Plate’ is more commonly known as ‘Number Plate’ so I shall be referring to it interchangeably. Kindly refer to the appendix for code.

2.1 Libraries and Tools Used

In this project, I have used five libraries in total. There are two key libraries used for object detection and character recognition. Along with that three other libraries are used to help with visualization and manipulation of the contour/data. Refer to appendix [1] for code.

2.1.1 OpenCV

OpenCV is an open source cross-platform library for object detection. It was initially developed by Intel in 2000 and further joined by Willow Garage. It is written in C/C++. Apart from object detection, it can be used for 2D/3D feature toolkits, egomotion estimation, Facial recognition system, Gesture recognition, Human-computer interaction (HCI), Mobile robotics, Motion understanding, Segmentation and recognition, Stereopsis stereo vision: depth perception from 2 cameras, Structure from motion (SFM), Motion tracking and Augmented reality.

2.1.2 EasyOCR

It is an open-source character recognition library, published on GitHub under the name *JaidedAI.* It is a ready-to-use optical character recognition(OCR) library with more than 80 supported languages and popular writing scripts including Latin, Chinese, Arabic, Devanagari, Cyrillic, etc. It uses a PyTorch based deep learning network for character recognition.

2.1.3 Imutils

The imutils library contains a series of easy to use basic functions that can be applied to OpenCV functions and subfunctions. Basic operations like rotation, resizing, skeletonization, translation, and visualizing images.
In this project, we have used this library to detect, collect, and perform programming logic on contours.

2.1.4 Matplotlib

It is one of the most important libraries for visualization in python. It can create static, animated, and interactive visuals. In this project, every output image is shown using the *pyplot* function of this library.

2.1.5 Numpy

Numpy is a popular data manipulation python library. It is extensively used for numerical computing on large multi-dimensional arrays and metrics. Every image read by openCV is stored in the form of a 3D numpy array. In the project, image binarization and operations are some of the important operations done using numpy.

2.2 Flow Chart of The Proposed Approach

This project work is divided into nine major steps.Here I shall explain to you each step taken to achieve the final result .These steps are represented in the below chart.

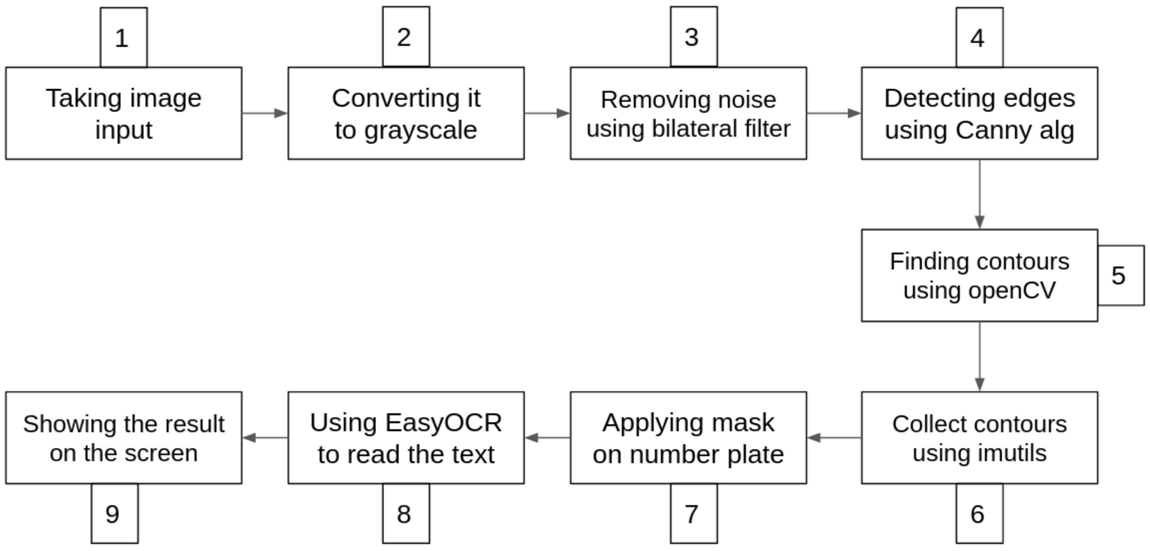

1
Taking image input
2
Converting it to grayscale
3
Removing noise using bilateral filter
4
Detecting edges using Canny alg
5
Finding contours using openCV
6
Collect contours using imutils
7
Applying mask on number plate
8
Using EasyOCR to read the text
9
Showing the result on the screen

Fig 1. Flow chart of the proposed approach

2.2.1 Taking Input Image

In this project we are mainly dealing with static image data. It can also be extended to video just by adding a few more lines of code. The storage of data with the database can also be done.

To read an image, use the **cv2.imread(img)** method. This method takes only one variable as an input and that is the path of the image. Every image read using this method is kept in the form of a 3-dimensional numpy array. It can be imagined as 3 layers of 2D array. Each layer represents blue, green and red color intensities.
Refer to appendix [3].

2.2.2 Conversion To Grayscale

The input image is an image with all three primary colors i.e. red, green and blue. Grayscaling speeds up the process since we no longer have to deal with RGB color details while preprocessing. So, we convert the image into grayscale. One of the reasons we do that is to reduce the file size since every pixel can now be represented as a 8-bit number i.e. a value from 0 to 255, where 0 represents black(lowest intensity) and 255 means white(highest intensity).

To convert an image into grayscale use the **cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)** method. This method takes two parameters as an input. First is the image itself and second is *cv2.COLOR_BGR2GRAY*. The second parameter helps the method know the image into grayscale. Refer to appendix [4].

2.2.3 Noise Reduction and Edge detection

This step includes two major operations, the first is noise reduction using a bilateral filter which converts the image into a binary image i.e. a black and white image. And the second one is edge detection using Canny algorithm which detects all the edges found in the image. It later helps in finding contour areas. In this project our primary focus is to locate the license plate. Everything other than the license plate is useless information. The useless information in any data is called noise. Using a binary filter smoothens the image and blurs the background which helps the program to focus on the foreground.

To remove noise from the image use **cv2.bilateralFilter(source_image, diameter_of_pixel, sigmaColor, sigmaSpace)** method. This method takes four parameters: *source_image*, *diameter_of_pixel*, *sigmaColor* and *sigmaSpace.* Here the source_image is our grayscale image. The dimatere_of_pixel is the diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace. *sigmaColor* and *sigmaSpace* are jointly used to adjust the blur effect on the image.

To detect edges in an image **cv2.Canny(img, threshold1, threshold2)** algorithm is used. It is a more effective edge detection algorithm than Sobel edge detection. The *Canny* algorithm takes three input parameters: *input_image*, *thresholdValue1* and *thresholdValue2*. Here the input image is the output of the ***binaryFilter()*** image. thresholdValue1 and thresholdValue2 are the minimum and maximum threshold values that help to find connected lines in the image. These two threshold values determine how connected or less connected lines are required.
Refer to appendix [5].

2.2.4 Contour Finding and Collecting

Contour is a curve that joins all the points with same pixel intensities, in other words contour is any close surface in image. In binarization (the process of making an image black and white) operations the image gets converted into black and white image. Once contour are detected then

To find all the contours use **findContours()** method of the openCV library. Store the detected contours in a variable using **grab_contours()** method of the **<u>imutis</u>** library. The findContours() method of imutils library takes three input parameters: edge_image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE. The edge image is the output of the Canny() function. The cv2.RETR_TREE retrieves all of the contours and reconstructs a full hierarchy of nested contours.

The cv2.CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points. Once the contour has been detected, sort them in the ascending order and take the first ten contours. This process is similar to slicing a list in python.
Refer to appendix [6].

2.2.5 Plate Contour Detection

The output of the grab_contours() contains all the curves with closed loops. Sorting(in ascending) and slicing helps to get the first ten contours. The license plate also has a well defined closed surface. The first ten contours have the highest probability to contain the contour image. To extract the number plate, loop through the contours and find the polygon with four corners.

Use **cv2.approxPolyDP(curve, epsilon, closed[, approxCurve])** to approximate a contour with another polygon but with less vertices so that a contour with jagged lines does not show many vertices. The takes three parameters: *curve* which is a polygon contour, *epsilon* specifies the approximation accuracy. This is the maximum distance between the original curve and its approximation. The *closed* parameter takes a boolean input if true the contour is closed and if false it can be open. With some programming logic we can extract the number plate contour which is a closed contour area with four vertices.
Refer to appendix [7].

2.2.6 Masking of Number Plate

Now we have the coordinates of the number plates. We can mask everything, other than the number plate, to pixel intensity. To do this, take an image size black background. Then draw the plates contour on top of the black background and in the final step use the bitwise operator to crop the number plate region from the original image.
To draw the black background, use **numpy.zeros(gray.shape, np.uint8).** Here the first parameter is the dimension of the image and the second parameter tells us to use 8-bit unsigned naming convention.Now to draw plate contour on the background use **cv2.drawContours(image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]])** method. It requires three essential and few non-essential parameters. The *image* takes the background image, *contour* takes the plate contour, *conoturIdx* takes contour index here it is zero, to draw all contours use negative value. The color takes the input of the contour curve, here 255 (that is white) is taken. The last parameter tells us to take the inner side of the contour. Now the plate contour curve is drawn on the black background, use **cv2.bitwise_and(img, img, mask=mask)** to mask the other than the number plate from the original image.
Refer to appendix [8].

2.2.7 Number Plate Cropping

Now we have masked the useless information, we can now crop the image using numpy library. Since every image is a numpy array and we know the coordinates of the number plates from the number plate contour. It is similar to slicing a two dimensional array.
Take the gray image, pass the xmin and xmax as row values and ymin and ymax as column values.
Refer to appendix [9].

2.2.8 Character Segmentation

This is the simplest yet important part of this project. In this section the number plate information is extracted, more formally known as ‘character segmentation’. Here **easyOCR** is used but other libraries like **pytesseract** can also be used.

To extract the number plate simply make an object of the Reader class of *easyocr*. Use the **readtext()** method of the object with a number plate image as input parameter. The output of the method is in the form of a tuple which contains three elements. First the number plate coordinates in a 2D array, second is the text extracted from the image and third is the confidence percentage that tells accuracy of the output.

Refer to appendix [10].

2.2.9 Output Rendering

This is the last stage in which the output is displayed along with the original image. This process is formally known as rendering. To render the output, a rectangle is drawn around the plate contour and the plate text is pasted at one of the corners of the plate.

To make a rectangle around the number plate use **cv2.rectangle(img, pt1, pt2, color[, thickness[, lineType[, shift]]])**. The parameters take the number plate image, first point of the rectangle, second point of the rectangle, color of the rectangle in RGB format and thickness of the rectangle respectively, rest of the parameters are optional.

To put the text on the rectangle use **cv2.putText(img, text, org, fontFace, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])**. Here the *img* is the original image, *text* is the extracted number plate characters, *org* is the bottom-left corner of the text string in the image, *fontFace* is the text font style for rendering, *fontScale* is the font scale factor that is multiplied by the font-specific base size, *color* is the font color in RGB format, *thickness* refers is the font weight or thickness of the line of the text, *linetype* is the line type is can be one of the four types provided by openCV, *bottomLeftOrigin* takes a boolean input. When true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.

Refer to appendix [11].

3. FUTURE SCOPE

This model has primarily used image-based data which can be easily extended to video(since the video is made up of several fast-moving images) data by adding a few lines of code. But apart from that, there are several other improvements that could be done in this program to achieve more accurate results. Some of the points are written below:

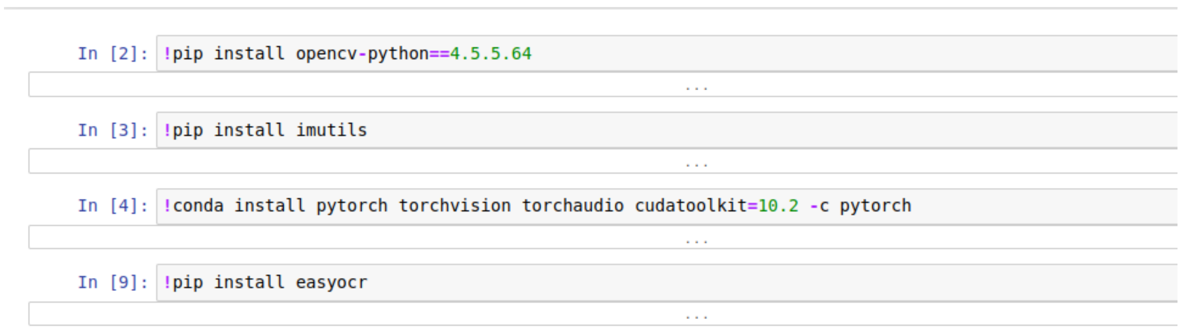
1. **To Automate Token System at JNU Entrance Gate:** At the main gate of our campus, every unknown vehicle that wants to enter is bound to register itself. Apart from capturing the name and purpose of the person, we can automate everything. The things we can store in a database are number plate, time of entering,

2. **Adding a Custom OCR:** The OCR or optical character recognition library we have mainly used for reading text and scripts of different languages. It is performing fairly well but in this case, we can make a custom neural OCR model. The text on cars is different from other languages and is different for different countries. If we can extract text from all the available number plates and make a custom model just to recognize car text then that would perform super. This work is a project on its own.

3. **Integration of RFID Technology:** To collect tolls at toll plazas on highways an antenna is used. The technology these antennas use is known as RFID. This project could be further extended to integrate with RFID technology and make toll plaza-free highways. Honorable Minister of Road Transport and Highways of India, Nitin Gadkari, is planning to make all roadways toll plaza-free by 2025.

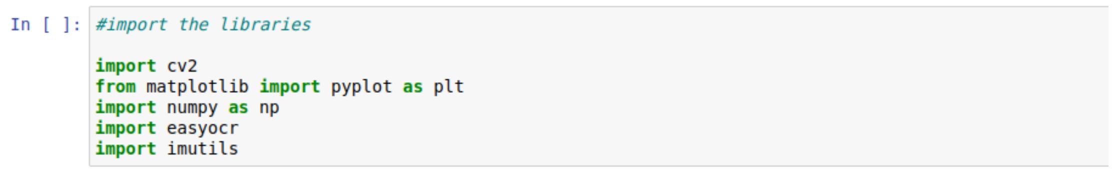
4. APPENDIX

1. Installation of dependencies
    - Use terminal or command prompt to install the dependencies .
    - Pytesseract can also be used to read optical characters.



```
In [2]: !pip install opencv-python==4.5.5.64
                                   ...
In [3]: !pip install imutils
                                   ...
In [4]: !conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
                                   ...
In [9]: !pip install easyocr
                                   ...
```

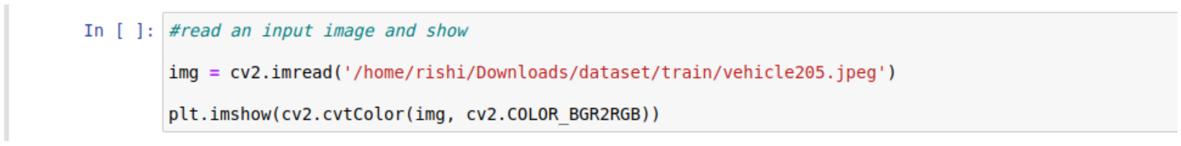
2. Import the required libraries



```
In [ ]: #import the libraries

        import cv2
        from matplotlib import pyplot as plt
        import numpy as np
        import easyocr
        import imutils
```

3. Reading the input image

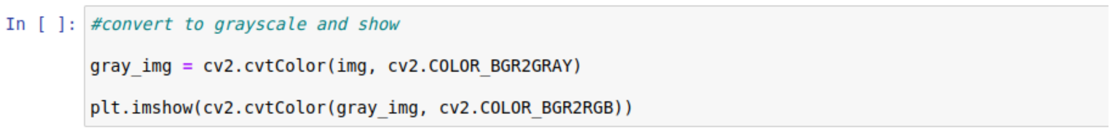


```
In [ ]: #read an input image and show

        img = cv2.imread('/home/rishi/Downloads/dataset/train/vehicle205.jpeg')

        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

4. Converting the original image into grayscale image

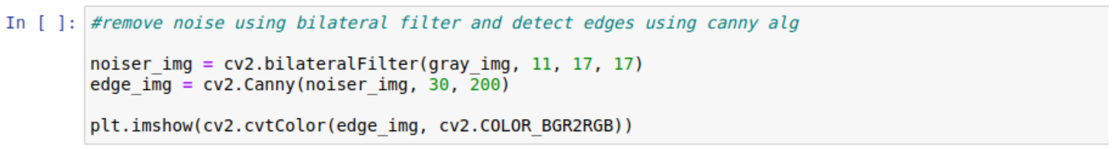


```
In [ ]: #convert to grayscale and show

        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        plt.imshow(cv2.cvtColor(gray_img, cv2.COLOR_BGR2RGB))
```

5. Noise Reduction and Edge Detection



```
In [ ]: #remove noise using bilateral filter and detect edges using canny alg

        noiser_img = cv2.bilateralFilter(gray_img, 11, 17, 17)
        edge_img = cv2.Canny(noiser_img, 30, 200)

        plt.imshow(cv2.cvtColor(edge_img, cv2.COLOR_BGR2RGB))
```

6. Contour Detection

```
In [ ]: #find contour in the edges and grab the regions using imutils library's grab_conotur method

        edge_points = cv2.findContours(edge_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

        contours = imutils.grab_contours(edge_points)
        contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]
```

7. Plate Contour Detection

```
In [ ]: #select the contour with 4 edges

        plate_points = None
        for contour in contours:
            points = cv2.approxPolyDP(contour, 10, True)

            if len(points)==4:
                plate_points = points
                break
        plate_points
```

8. Masking of Number Plate from Background

```
In [ ]: #mask the number plate and convert the background to black

        mask = np.zeros(gray_img.shape, np.uint8)

        new_img = cv2.drawContours(mask, [plate_points, 0, 255, -1)
        new_img = cv2.bitwise_and(img, img, mask=mask)

        plt.imshow(cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB))
```

9. Cropping Number Plate

```
In [ ]: #grab the plate area using coordinates

        (x, y) = np.where(mask==255)
        (x1, y1) = (np.min(x), np.min(y))
        (x2, y2) = (np.max(x), np.max(y))

        plate_img = gray_img[x1:x2+1, y1:y2+1]

        plt.imshow(cv2.cvtColor(plate_img, cv2.COLOR_BGR2RGB))
```

10. Character Segmentation: Reading Plate Text

```
In [ ]: #make a Reader class object, pass the plate image and read the text

        reader = easyocr.Reader(['en'])
        plate_info = reader.readtext(plate_img)
        plate_info
```

11. Rendering The Output

```
In [ ]: # render the outplut on image

        plate_text = plate_info[0][1]

        font = cv2.FONT_HERSHEY_SIMPLEX

        result = cv2.putText(img, text=plate_text, org=(points[0][0][0], points[1][0][1]+60),
                    fontFace=font, fontScale=3.5, color=(0, 255, 255), thickness=4, lineType=cv2.LINE_AA)

        result = cv2.rectangle(img, tuple(points[0][0]), tuple(points[2][0]), (0, 255, 255), 3)

        plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
```

5. REFERENCES

1. Kaur, S. and Kaur, S. “An Efficient Approach for Number Plate Extraction from Vehicles Image under Image Processing”, International Journal of Computer Science and Information Technologies, 2014
2. https://muthu.co/algorithm-for-detecting-and-extracting-number-plates-from-images-of-cars/
3. https://medium.com/data-science-in-your-pocket/vehicle-number-plate-detection-and-ocr-tcs-humain-2019-a253019e52a1
4. https://blog.devcenter.co/developing-a-license-plate-recognition-system-with-machine-learning-in-python-787833569ccd
5. https://circuitdigest.com/microcontroller-projects/license-plate-recognition-using-raspberry-pi-and-opencv
6. https://en.wikipedia.org/wiki/Automatic_number-plate_recognition
7. https://www.jaided.ai/easyocr/documentation/
8. https://github.com/PyImageSearch/imutils
9. https://matplotlib.org/stable/index.html
10. https://devdocs.io/numpy~1.14/