## horizontal line



MOSAIC PS2

License Plate Recognition

TEAM : United

Member 1 : Yatharth Bhargava

Member 1 : Yash Jain

Member 1 : Deepak Kumar Meena

# Overview

In the problem statement we were given the task to implement an automatic Indian number plate recognizer in an unconstrained condition that considers occlusion, poor quality of images, and other spatial variations in image data. Furthermore there were Extra Points for reading the number plate from-

● Full image of car(s)

● Video of car(s)

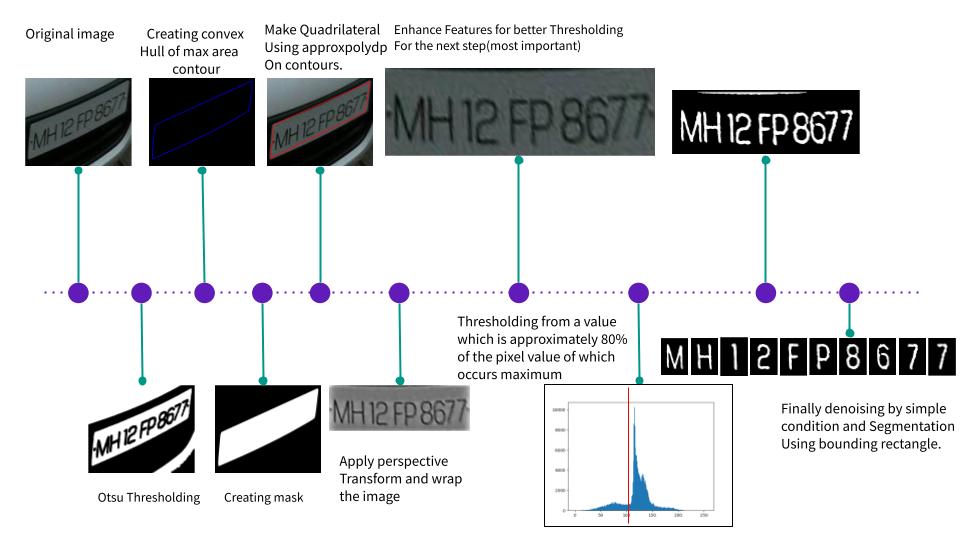
● Other practically useful & innovative additions.

Note:

* The first two letters on a number plate (DL, KL, HR, MH etc) denote the region or the state the vehicle is registered with. For a vehicle registered with Chhattisgarh RTO will bear the letters CG.
* The following digits indicate the district in which the vehicle is registered.
* The third part of the license plate is a set of numbers (typically four) which is unique to the vehicle. Vanity numbers, like 0001, 0786, 1111 are regarded as VIP numbers and can be bought at RTO auctions for a premium price.



# Workflow:



# Documentation

The folder contains 3 python files:

* main.py
* segmentation.py
* local\_utils.py

## main.py

This contains the main driver code for prediction.

* predict(image)

Inputs an original image and outputs the list which contains a list of characterters of corresponding number plates.

* test()

Takes path of image and send the corresponding images to predict function for predictions

## segmentation.py

This contains the utility function for segmentation and image processing.

* predict\_from\_model(image,model):

This function does the prediction for the segmented image with the mobilenet model.

* four\_point\_transform(image, pts):

This function returns the perspective Transform of the image with respect to the 4 points passed in the pts argument.

* order\_points(pts,shape):

Returns the pts points rearranged in clockwise manner for perspective transform.

* createMask(size, hull):

Gives the inner side of the hull high pixel(white) and outer side as low pixel(black).

* rectContains(rect, pt):

Checking if a rectangle is present inside other helps in selecting the outermost rectangle.

* refine\_contours(list\_coord):

Removes all the rectangles that are present inside a bigger rectangle.

* CreateHull(plate):

Gives the convex hull of the number plate in the form of a list, can be understood as

hull = []

hull.append(cv2.convexHull(final\_cnt, False))

return hull

* imclearborder(imgBW, radius):

Given a black and white image, removing the contours that are in certain radius of the image boundary helps in removing unnecessary noise.

* bwareaopen(imgBW, areaPixels):

Helps in removing the noise whose bounding rectangles area is small by simple area comparison and returns the corrected image.

* refine\_final(imgBW):

Helps in removing the noise in final segmented characters due to the neighbouring characters which comes while cropping the characters by bounding rectangles.

* ClearThresold(wrap):

This function makes use of the scipy and numpy library to increase the features in image and helps in accurate thresholding. Thresholding is also done in this function. For thresholding we used logic from histogram for finding the accurate Thresholding value. Max value in histogram represents the white part so we (80% +- 10%) of that value for thresholding.

Inspired from : <https://stackoverflow.com/questions/24731810/segmenting-license-plate-characters>

* refine\_answer(ans\_lis):

This function helps to increase the accuracy of answers for indian car license number plates by using the rules of license plate numbers in India as mentioned in the Note tag above documentation.

* Get\_Segmented(Iopen):

Takes input the binary image and which majorly contains characters only and no or very minimal noise and does segmentation while maintaining characters in sorted order. It returns the list of segmented letter images in sorted order which are ready for prediction.

## local\_utils.py

* get\_plate(image, Dmax, Dmin):

This function makes use of wpod net model for finding the number plate in car images.

* load\_model(path):

Simply loads model from the given path

* preprocess\_image(image,resize=False):

Converts the image from BGR to RGB and then converts the image pixel from [0-255] to [0-1]

# Accuracy metric on test-set:

We were given the set of images given below at the time of evaluation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Image | No. of characters originally in image | No. of segmented Characters | No. of Characters segmented correctly  (+2 marks) | No of noise in final segmented characters  (-1 marks) | No. of characters correctly recognised among segmented characters  (+2 marks) | Score |
| t1.png | 10 | 10 | 10 | 0 | 10 | 40 |
| t2.png | 9 | 9 | 9 | 0 | 9 | 36 |
| t4.png | 10 | 10 | 10 | 0 | 9 | 38 |
| t5.png | 10 | 10 | 10 | 0 | 10 | 40 |
| t6.png | 10 | 10 | 10 | 0 | 10 | 40 |
| t7.png | 10 | 10 | 10 | 0 | 10 | 40 |
| t8.png | 10 | 10 | 10 | 0 | 9 | 38 |
| t9.png | 10 | 10 | 10 | 0 | 10 | 40 |
| t10.png | 10 | 11 | 10 | 1 | 9 | 37 |
| t11.png | 9 | 9 | 9 | 0 | 9 | 36 |
| t12.png | 10 | 11 | 9 | 2 | 10 | 36 |
| t13.png | 10 | 12 | 10 | 2 | 10 | 38 |
| t14.png | 10 | 10 | 10 | 0 | 8 | 36 |
| t15.png | 10 | 9 | 9 | 0 | 7 | 32 |
| t16.png | 10 | 14 | 3 | 11 | 1 | skipped |
| t18.png | 10 | 16 | 10 | 6 | 9 | 32 |
| t19.png | 10 | 14 | 5 | 9 | 2 | 5 |

We have excluded t16.png in the below calculation as it was causing us negative marking

**Results:**

* **Accuracy of Correct segmentation of Characters: 151/158 = 0.955 = 95.5%**
* **Average no. of noise detected in an image Noise detected in image:20/16 = 1.25( Mainly due to last 2 images, excluding that the value is 5/14 = 0.35).**
* **Accuracy of prediction if character is segmented correctly = 141/151 = 0.934 = 93.4%**