

A Robust Real Time License Plate Recognition System

Introduction

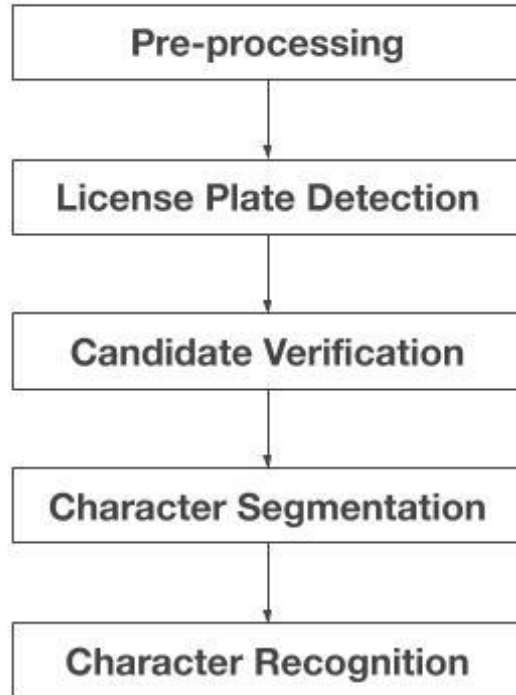
There has been a lot of research in the field of Automatic License Plate Recognition (ALPR) due to its widespread applications in vehicle monitoring and security, identification of traffic rule violators, identification of stolen vehicles and in toll booths.

We propose to develop an ALPR system based on morphological image processing techniques that can run on a Raspberry Pi. It is followed by the development of a more robust solution based on the state-of-the-art YOLOv4 object detector.

Motivation

- Current developments in the fields of computer vision and powerful small-sized devices (like the Nvidia Jetson Nano and Intel Movidius Neural Compute Stick) have enabled the development of edge-based techniques for building fast and reliable solutions for latency-sensitive applications.
- Though ALPR has been a hot topic of research in the recent years, not many robust solutions exist that can identify and recognize license plates in real time under varying environmental conditions, camera viewing angles and brightness variations. There aren't many solutions that work well in the context of Indian license plates, where ALPR is a big challenge in itself arising from the use of non-standardized license plates.

License Plate Recognition Process



Connected Component Extraction after Binarization



Original Image



Resized Image



Denoised Grayscale Image



Otsu Thresholding of Grayscale Image



Adaptive Thresholding in HSV Space

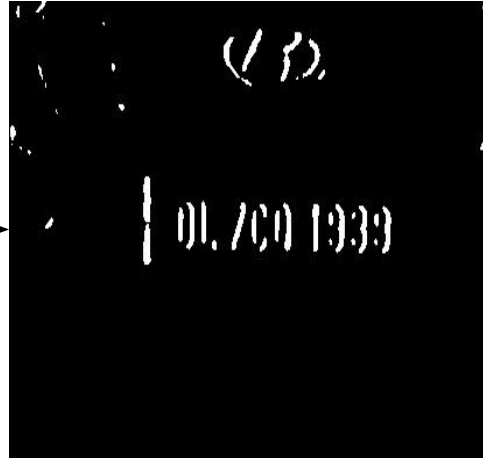


Our Approach

Edge-Detection Based Approach



Denoised Grayscale Image



Vertical Sobel Operator + Otsu
Thresholding



Closing Operation using 17 x 4
Structuring Element

Our Approach



Denoised Grayscale Image



Resized Image



Averaging using 20 x 20 Mean
Filter



Subtracting with Original Image



Otsu Thresholding at a Low
Value



Plate Extraction using CCA and
Candidate Verification

Character Segmentation



Extracted License Plate



Image Sharpening using Unsharp Masking



Adaptive Thresholding in HSV Space



Character Segmentation using Contour Extraction

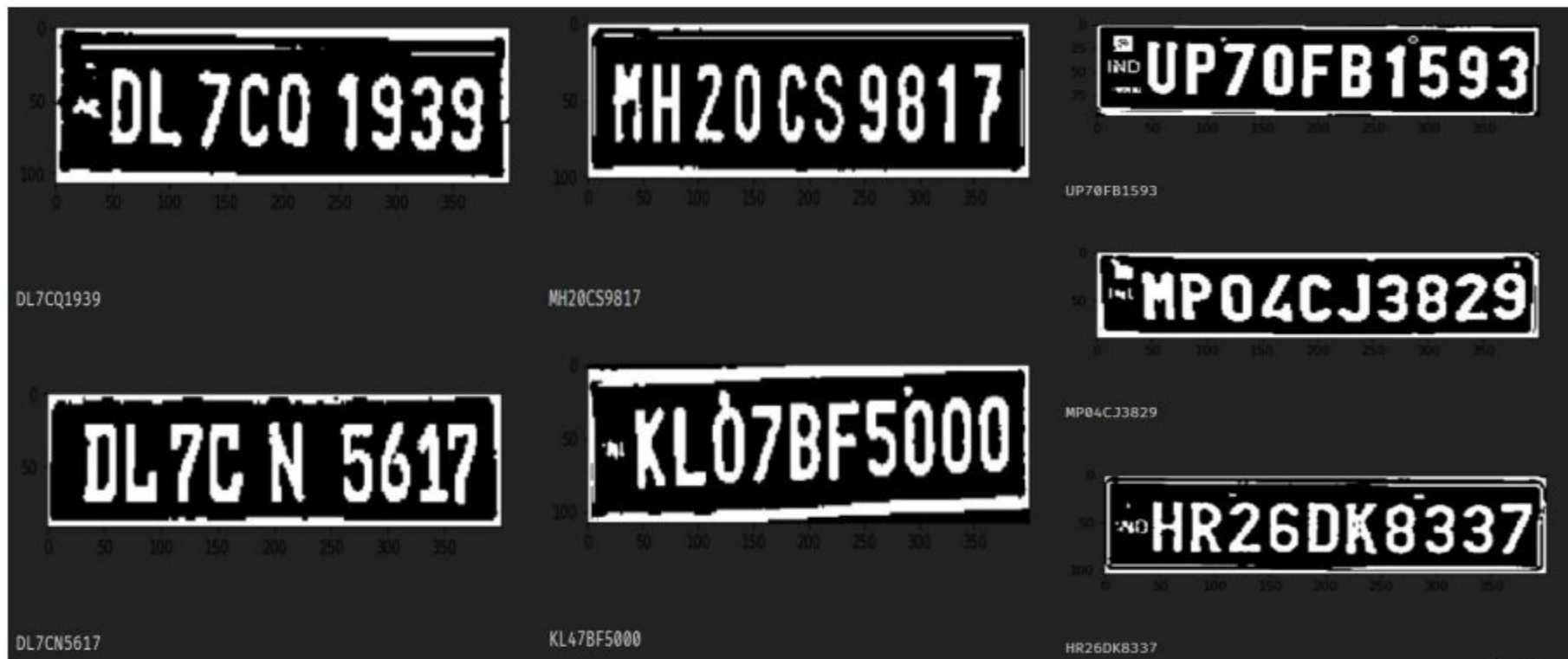


Resizing Characters for Recognition

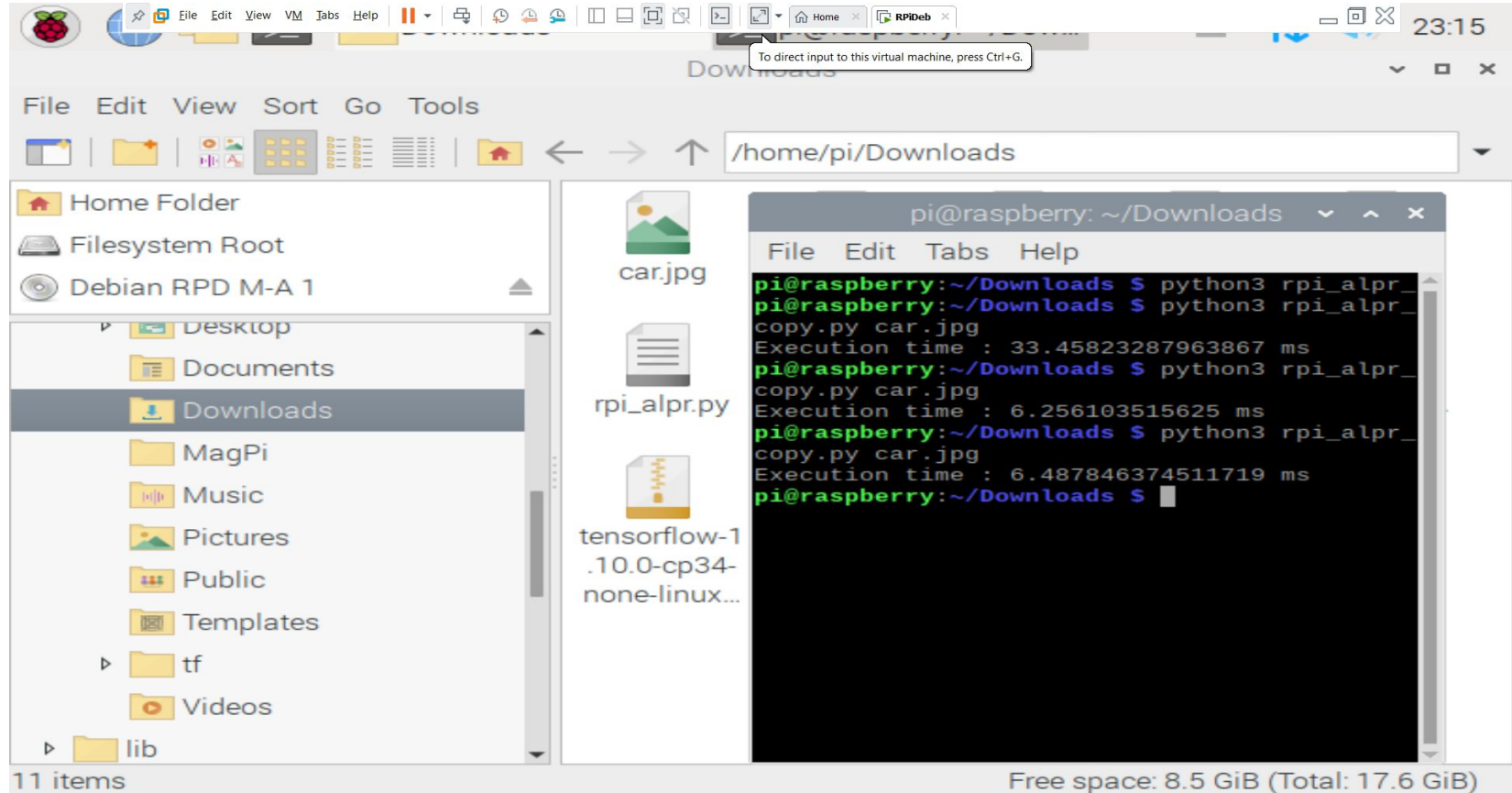
Character Recognition Model Architecture

#	Layer	Filters	Filter Size	Activation
0	convolutional	6	3×3/1	
1	maxpool		2×2/1	
2	convolutional	16	5×5/1	
3	maxpool		2×2/1	
4	convolutional	120	5×5/1	
5	flatten			
5	dense			relu
6	dense			relu
7	dense			softmax

Character Recognition : Results



Execution on Raspberry Pi (Emulated)



The screenshot displays a virtual machine environment. At the top, a menu bar includes 'File', 'Edit', 'View', 'VM', 'Tabs', and 'Help'. Below this is a toolbar with various icons. The main window is a file manager showing the '/home/pi/Downloads' directory. The left sidebar lists the 'Home Folder', 'Filesystem Root', and 'Debian RPD M-A 1'. The main pane shows three files: 'car.jpg', 'rpi_alpr.py', and 'tensorflow-1.10.0-cp34-none-linux...'. A terminal window is open in the foreground, showing the execution of a Python script 'rpi_alpr.py' three times. The terminal output shows the execution time for each run. A tooltip at the top center reads 'To direct input to this virtual machine, press Ctrl+G.' The bottom status bar indicates 'Free space: 8.5 GiB (Total: 17.6 GiB)'.

File Edit View VM Tabs Help

Downloads

To direct input to this virtual machine, press Ctrl+G.

File Edit View Sort Go Tools

/home/pi/Downloads

Home Folder

Filesystem Root

Debian RPD M-A 1

Desktop

Documents

Downloads

MagPi

Music

Pictures

Public

Templates

tf

Videos

lib

11 items

car.jpg

rpi_alpr.py

tensorflow-1.10.0-cp34-none-linux...

pi@raspberrypi: ~/Downloads

File Edit Tabs Help

```
pi@raspberrypi:~/Downloads $ python3 rpi_alpr.py
pi@raspberrypi:~/Downloads $ python3 rpi_alpr.py
copy.py car.jpg
Execution time : 33.45823287963867 ms
pi@raspberrypi:~/Downloads $ python3 rpi_alpr.py
copy.py car.jpg
Execution time : 6.256103515625 ms
pi@raspberrypi:~/Downloads $ python3 rpi_alpr.py
copy.py car.jpg
Execution time : 6.487846374511719 ms
pi@raspberrypi:~/Downloads $
```

Free space: 8.5 GiB (Total: 17.6 GiB)

Optimizations and Results

- Resizing and cropping images before performing the localization task (since the license plate will be present in the lower 60% of the image).
- Using array computations.
- Structuring the recognition code according to the probability of license plate formats.

Results:

- Execution time on Raspberry Pi : **< 10 ms** (for localization and character segmentation)
- Expected execution time on Raspberry Pi : **~350 ms** (for character recognition).

Towards the YOLO-based approach

Our Contribution

- Due to unavailability of publicly-available license plate datasets and car datasets for India, we created and manually annotated our own dataset using drive-through videos of various cities of India from YouTube videos.
- We labelled a dataset of around 6000 images for training the localization stage.
- For training the character segmentation and recognition stages of the YOLO-based approach, we labelled around 1000 images (~ 7000 annotations).
- For the latter stages, we also utilised standard publicly-available license plate datasets of other countries since the dataset size was small.

YOLOv4 based License Plate Localization

Image Augmentation for Increasing Robustness

- Varying Brightness
- Varying Color Temperatures
- Varying Color
- Shear
- Rotation
- Shuffling Channels
- Simulating Rain and Clouds
- Adding Noise



YOLO based Architecture for License Plate Localization

#	Layer	Filters	Size	Stride	Activation
0	convolutional	16	3×3	1	swish
1	maxpool		2×2	2	
2	convolutional	32	3×3	1	swish
3	maxpool		2×2	2	
4	convolutional	64	3×3	1	swish
5	maxpool		2×2	2	
6	convolutional	128	3×3	1	swish
7	maxpool		2×2	2	
8	convolutional	256	3×3	1	swish
9	maxpool		2×2	2	
10	convolutional	512	3×3	1	swish
11	maxpool		2×2	1	
12	convolutional	1024	3×3	1	swish
13	convolutional	512	3×3	1	swish
14	convolutional	1024	3×3	1	swish

Localization Results



YOLO based Architecture for Segmentation and Recognition

#	Layer	Filters	Size	Stride	Activation
0	convolutional	32	3×3	1	swish
1	maxpool		2×2	2	
2	convolutional	64	3×3	1	swish
3	maxpool		2×2	2	
4	convolutional	128	3×3	1	swish
5	maxpool		2×2	2	
6	convolutional	64	1×1	1	swish
7	convolutional	128	3×3	1	swish
8	maxpool		2×2	2	
9	convolutional	256	3×3	1	swish
10	convolutional	128	1×1	1	swish
11	convolutional	256	3×3	1	swish
12	maxpool		2×2	1	
13	convolutional	512	3×3	1	swish
14	convolutional	256	1×1	1	swish
15	convolutional	512	3×3	1	swish
16	convolutional	120	1×1	1	linear

Evaluation Results for YOLOv4-based Localization

Name	Images	Recall	FP	Misorientations
Brazilian Cars	503	1.00	2	4
OpenALPR-Brazil	78	1.00	0	2
OpenALPR-Europe	90	1.00	0	0
Our Dataset	454	0.99	12	5

Heuristics Based Character-Digit Matching

For increasing accuracy of character recognition results, we deploy a heuristic approach to convert commonly misclassified characters to digits and vice versa based on:

Types of number plates in India : **LLDLLDDDD**, **LLDLLLDDDD**, **LLDDLDDDD**, **LLDDLDDDD**, where **L** denotes a letter and **D** denotes a digit.

Digit	Letter
1	I, J
2, 7	Z
0	O, Q
4	A, L
6	G
3, 8	B
5	S

Common Misclassifications

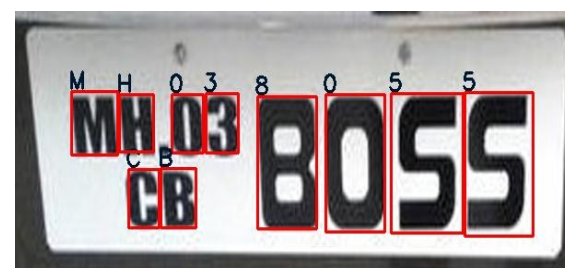
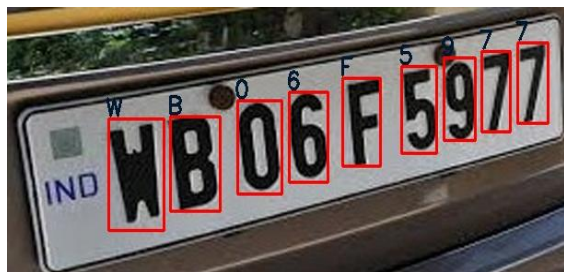
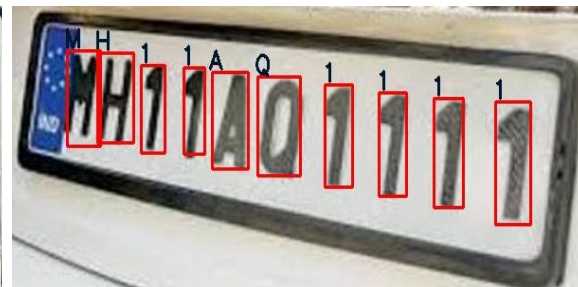
Evaluation Results for YOLOv4-based Segmentation

Dataset	Images	All	>80%	FP
Czech	652	638	7	30
Reld	475	452	14	3
Our Dataset	101	94	3	3

Character Segmentation Results



YOLOv4-based Character Recognition Results



Evaluation Results for YOLOv4-based Character Recognition

Dataset Name	Dataset Size	Incorrect Recognitions	Accuracy
Czech	652	22	0.995
Reld	475	70	0.978
Our Dataset	101	9	0.987

Conclusion and Scope of Improvement

- The morphology-based approach works well in cases where the probability of encountering a skewed license plate is not very high like in toll booths or for surveillance purposes.
- The YOLO v4-based approach is capable of detecting characters in any kind of license plates, although our prime focus while working on the project was to create an high-accuracy ALPR for single-line license plates. The evaluations have hence been done majorly on single-line license plates.
- The accuracy achieved in case of low resolution images is good, however more work can be done for decreasing the false positive rate and misclassifications in the segmentation-recognition stage, especially for images having low resolution or high skew (more insights to improve the accuracy can be achieved by using more challenging datasets like CCPD and Reid to reduce the bias of the model towards certain license plate layouts).

References

- Chen, R., and Luo, Y. An improved license plate location method based on edge detection. *Physics Procedia* 24 (12 2012), 1350–1356.
- Laroca, R., Zanlorensi, L. A., Gonçalves, G. R., Todt, E., Schwartz, W. R., and Menotti, D. An efficient and layout-independent automatic license plate recognition system based on the YOLO detector. *arXiv preprint arXiv:1909.01754* (2019), 1–14.
- Yang, X., and Wang, X. Recognizing license plates in real-time. *CoRR abs/1906.04376* (2019).
- Xu, Z., Yang, W., Meng, A., Lu, N., and Huang, H. Towards end-to-end license plate detection and recognition: A large dataset and baseline. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 255–271.

Thank You