AUTOMATIC LICENSE PLATE DETECTION AND RECOGNITION SYSTEM

A CREATIVE AND INNOVATIVE PROJECT REPORT

Submitted by

ANIKA ARIVARASHAN S (2019103506)

JEYASRI MEENAKSHI K (2019103530)

OMAR HAJA MOIDEEN (2019103044)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY:: CHENNAI 600 025

JUNE 2022

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **AUTOMATIC LICENSE PLATE DETECTION AND RECOGNITION SYSTEM** is the bonafide work of **ANIKA ARIVARASHAN S (2019103506), JEYASRI MEENAKSHI K (2019103530) and OMAR HAJA MOIDEEN (2019103044)** who carried out the project work under my supervision.

PLACE: CHENNAI **SIGNATURE**

DATE: **Mr. T.M. Thiyagu**

SUPERVISOR

Teaching Fellow

Department of Computer Science and Engineering

Anna University Chennai,

Chennai 600025

ABSTRACT

Automatic vehicle license plate detection and recognition is a key technique in most traffic related applications and is an active research topic in the image processing domain. Automatic License Plate Recognition (ALPR), is software used to recognize the number plates automatically by performing sophisticated optical character recognition on images to read the license plates of vehicles and it aims at exploiting image processing and pattern recognition techniques to extract and recognize the characters on license plates (LPs) from vehicle images or videos. ALPR approaches are crucial for intelligent transport systems (ITS) in a wide variety of several real-time applications. The proposed model involves two main stages namely, license plate detection and Tesseract-based character recognition. The detection of alphanumeric characters present in license plates takes place with the help of an advanced real-time object detection system YOLOv5 which divides all the given input images into the SxS grid system. Each grid is responsible for object detection. Now those Grid cells predict the boundary boxes for the detected object. CSP(Cross Stage Partial Networks) are used as a backbone in YOLO v5 to extract useful characteristics from an input image. Then the characters in the regions are recognized one-by-one via optical character recognition (OCR) techniques are extracted using the Tesseract Optical Character Recognition (OCR) model.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ALPR	Automatic License Plate Recognition
ANPR	Automatic Number Plate Recognition
CCTV	Closed Circuit Television
cfg	Configuration
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSP	Cross Partial Network
et al	And Others
GPU	General Processing Unit
img	Image
IoU	Intersect over Union
ITS	Intelligent Transport System
IWPOD-NET	Improved Warped License Planar Object Detection Network
LP	License Plate
mAP	Mean Average Precision
OCR	Optical Character Recognition
PIP	Package Installer for Python
RBF	Radial Basis Function
RoI	Region of Interest
RTA	Roads Transport and Authority
SVM	Support Vector Machine
SOTA	Special Operations and Training Area.
YAML	Yet Another Markup Language
YOLO	You Look Only Once

1. INTRODUCTION

Automatic license plate recognition (ALPR) on stationary to fast-moving vehicles is one of the common intelligent video analytics applications for smart cities. Some of the common use cases include parking assistance systems, automated toll booths, vehicle registration and identification for delivery and logistics at ports, and medical supply transporting warehouses. Being able to do this in real time is key to servicing these markets to their full potential. Traditional techniques rely on specialized cameras and processing hardware, which is both expensive to deploy and difficult to maintain.

The pipeline for ALPR involves detecting vehicles in the frame using an object detection deep learning model, localizing the license plate using a license plate detection model, and then finally recognizing the characters on the license plate. Optical character recognition (OCR) using deep neural networks is a popular technique to recognize characters in any language.

1.1. PROBLEM STATEMENT

Although ALPR has been successfully applied to environment-controlled smart parking systems, it still faces many challenging in the surveillance system such as it deals with highly degraded, noisy, low illuminated, cross angled, surrounding effects and non-standard font number plates. These variations result in false positives on plate detection, recognition and poor ALPR accuracy. Depending on the car class, the license plates can’t be placed in the right position which will lead to the plate image in the vehicle and the vehicles being misclassified. The appearance of odd characters can be misclassified as Number Plate

Characters in the License Plate, the non-uniformity of license plates depends on the car type, Single and multi-line characters, Dust and Unused License Plates, Non-Uniformity present in Font Types and Font Size.

1.2. PROPOSED MODEL

The focus of the proposed work to apply novel techniques in the context of registration number plate identification and realization. In a vast country like India, most of the plates are not standard, so a solution is needed that targets multiple formats and fonts. Firstly, Number Plate region (the Region-of-Interest) is detected using YOLOv5, which is now the most advanced object identification algorithm available. It is a novel convolutional neural network (CNN) that detects objects in real-time with great accuracy. This approach uses a single neural network to process the entire picture, then separates it into parts and predicts bounding boxes and probabilities for each component. These bounding boxes are weighted by the expected probability. and extracted amongst the larger captured image of the vehicle. Later, the task is to recognize the numbers and characters present in the region of interest. Here optical character recognition (OCR) is applied to the new image and it extracts all the characters. These characters may be the numbers and alphabets present in the number plate. Then the obtained image is converted to text and it is stored in a string variable which will be used to obtain the vehicle identification details.

2. LITERATURE SURVEY

Sergio M. Silva & Cláudio Rosito Jung, 2021. In [6] the core of the proposed method is the introduction of an Improved Warped License Planar

Object Detection network (IWPOD-NET), which is able to detect car and motorcycle LPs in a variety of capture situations and then rectify them to a fronto-parallel view. It presents accuracy comparable to other SOTA methods. Highly degraded LPs (e.g., low-resolution or motion blurred) were not recognized.

Hendry & Rung Ching Chen, 2019. [2] YOLO's 7 convolutional layers are used to detect a single class. The sliding window detects each digit of the license plate, and each window is then detected by a single YOLO framework. The system can detect license plates with high precision. To handle this trade-off, the number of layers in the YOLO model are decreased.

Farheen Ali et al, 2021. [1] The number plate is identified using Text-line Construction and Multilevel RBF Neural Network from the picture taken in the plate number detection process and then the segmented plate is transferred to the plate recognition in the second phase to determine the characters and numbers. Identifying and recognizing multiple license plate cars from a single frame is possible. The license plates can’t be placed in the right position. This will lead to the plate image in the vehicle and the vehicles being misclassified and requiring manual surveillance.

Vijaya Vardhan Reddy SP et al, 2021. [7] The recognized number plate is then compared with the database and checks whether the vehicle belongs to the organization. A webcam is connected to a raspberry in order to capture the image at the entrance of the organization. Raspberry pi 3 is connected to the network to transfer the extracted information to the database server. Initially the image is captured when the licensed plate is detected. Better performance is not achieved. Manual surveillance is required to oversee this system.

Naren Babu R et al, 2019. [5] The main objective of this work is to create a robust number plate recognition model that works under different illuminations and angles. This recognition model was created by training on our manually collected car number plate dataset using YOLO V3. An input image is fed into the YOLO model, then if the number plate is detected, the corresponding number plate region of interest (ROI) is extracted and this image is again fed into the YOLO model to get recognized. Finally, the recognized output is sorted from left to right so that it's in the correct order as in the number plate.

Chao Xu et al, 2020. [9] In this paper, OpenCV is used to locate the license plate area, the characters are cut by projection, and the characters are recognized by convolution neural networks. The disadvantage of literature is that vertical projection is used to avoid license plate tilt, which is not ideal.

Cheng-Hung Lin et al, 2018. [4] An efficient hierarchical methodology for license plate recognition system with integration of YOLOv2 model and SVM can capture license plates with high accuracy that first detects vehicles using deep learning techniques and then retrieves license plates from detected vehicles to reduce false positives on plate detection. This methodology shows the superiority in both accuracy and performance in comparison with traditional license plate recognition systems.

Monika Khinchi & Chanchal Agarwal, 2019. [3] Raspberry pi is used for the recognition of the Number plate automatically. This system is used by the OpenCV Platform. For character detection Optical Character Recognition(OCR) method is used. For distance measurement ultrasonic sensors are used. This system reduces manual work and can also be extended by taking broken number plates, trying to recognize the number plate and taking

real time input in the form of video but only 67.98% accuracy was obtained under this method.

Ravi Kiran Varma P et al, 2019. [8] The python OpenCV library is used for implementing the image processing tools and for number plate segmentation, contours are applied by border following and contours are filtered based on character dimensions and spatial localization. Finally, after the region of interest filtering and de-skewing, the K-nearest neighbour algorithm is used for character recognition. Sophisticated models such as Gradient Boosting tend to overfit the data as not much data was available which resulted in poor predictions.

3. METHODOLOGY

3.1. OVERALL ARCHITECTURE

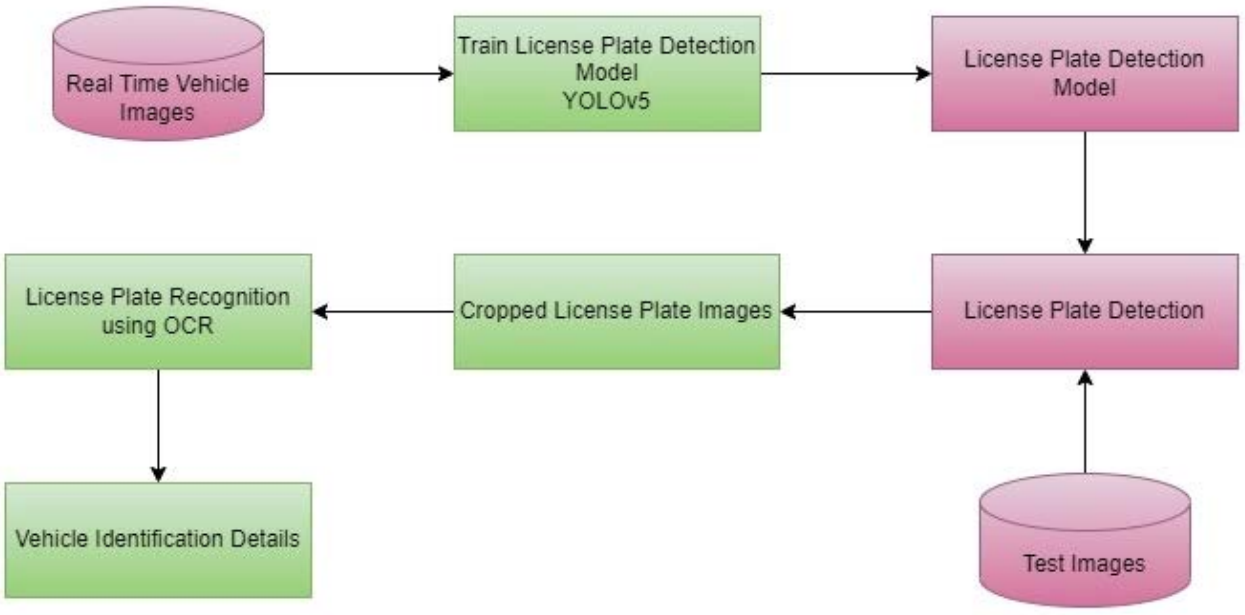


Fig. 3.1.1. Overall Architecture of the developed model.

3.2. SYSTEM DESIGN

The model that we have designed automatically detects the license plates from the input images using YOLOv5, a recent release of the YOLO family models that is 88% smaller and 180% faster than the previous model. It is also the first of the YOLO models to be written in the PyTorch framework being much more lightweight and easy to use. YOLO algorithms divide all the given input images into the SxS grid system. Each grid is responsible for object detection. Now those Grid cells predict the boundary boxes for the detected object. For every box, we have five main attributes: x and y for coordinates, w and h for width and height of the object, and a confidence score for the probability that the box containing the object.

Its architecture mainly consisted of three parts, namely-

1. Backbone: Model Backbone acts as a feature extractor from an input image. Feature extractor is nothing but using convolutional layers such as kernel, stride, batch normalization applies on input images to extract important features such as edges, shapes, and so on. CSP(Cross Stage Partial Networks) are used as a backbone in YOLO v5 to extract useful characteristics from an input image.
2. Neck: In the Neck layer, the network was designed to perform Multi-scale prediction in addition to the Feature pyramid network. Multi-scale prediction helps to detect objects of different sizes by sending images into three different grid values. So small grid images detect large objects and large grid images detect small objects.

3. Head: The model Head is mostly responsible for the final detection step. It uses anchor boxes to construct final output vectors with class probabilities, object scores, and bounding boxes.

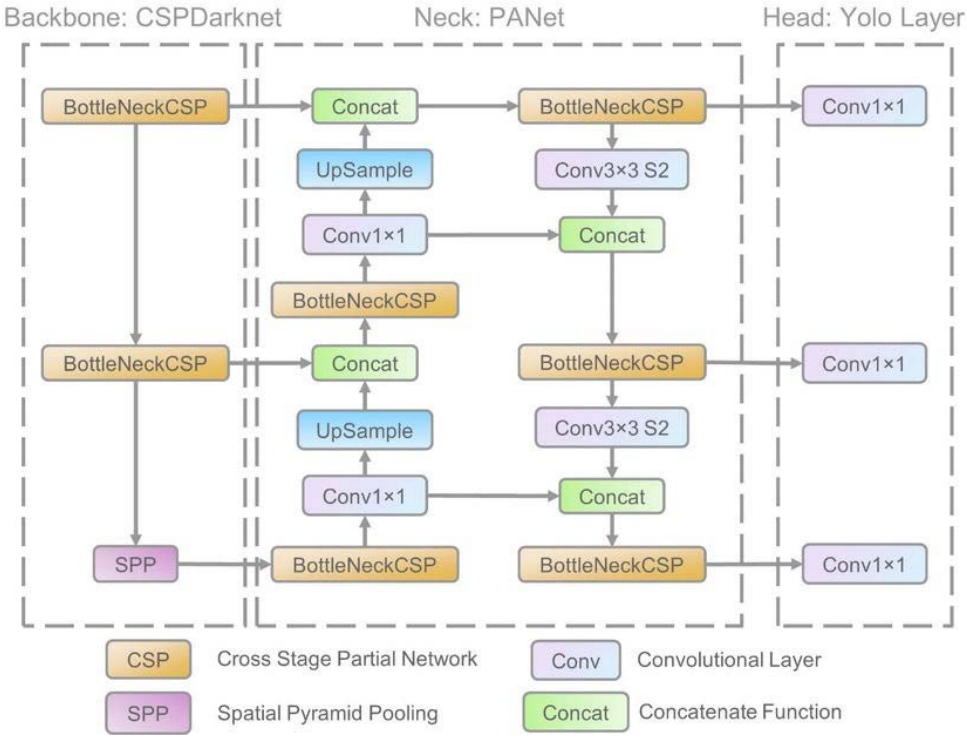


Fig. 3.2.1. YOLOv5 System Architecture

This full network was trained to perform regression task which is nothing but coordinates of a bounding box and the probability of a class.

Once the license plate is detected, it will be cropped and used for license plate recognition using the Tesseract OCR. An Optical Character Recognition Engine (OCR Engine) to automatically recognize text in vehicle registration plates. Py-tesseract is an optical character recognition (OCR) tool for python. That is, it'll recognize and "read" the text embedded in images. Python-tesseract is a wrapper for Google's Tesseract-OCR Engine. It is also used as an individual script, because it

can read all image types like jpeg, png, gif, bmp, tiff, etc. Whereas OpenCV is an open source computer vision library. The library has more than 2500 optimized algorithms. These algorithms are often used to search and recognize faces, identify objects, recognize scenery and generate markers to overlay images using augmented reality, etc. Which will also be used for image processing techniques.

3.3. DATASET

In a vast country like India, most of the plates are not standard, as they have multiple formats and fonts. Also, there is no standard data set available for Indian Plates, so around 1360 images of vehicles with Number plates were manually taken, considering cases like low, non-uniform illumination, distant plates, blurry plates, varied fonts etc. Since all the data were collected in India and most of the Number plates are composed of four letters and six numbers such as AP 31 BA 1432, and four letters and four numbers like, TN 30 BH 45 where the first two characters signify the State of the Vehicle's Registration.

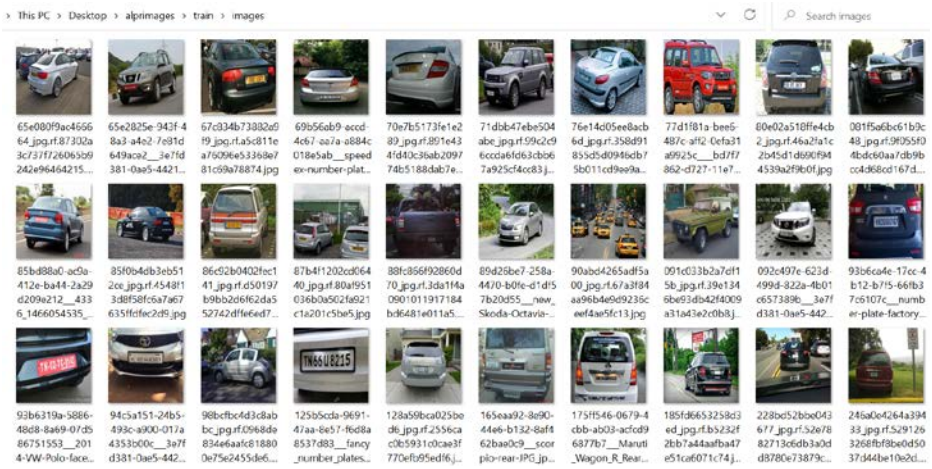


Fig. 3.3.1. Sample Collection of Real Time Images

4. MODULE DESIGN

4.1. DATASET CREATION

Image acquisition has been be done from CCTV footages, parking system cameras. After collecting images we detect the dataset and its format and adds annotations respectively. Here we will be taking ‘vehicle’ and mainly ‘license-plate’ into account. We use Roboflow to create the training images. It is a Computer Vision developer framework for better data collection to preprocessing, and model training techniques.

Here’s one of the image (real time image) from our dataset.

Fig. 4.1.1. Real time image

Now we will be creating bounding boxes for 2 classes – ‘vehicle’, ’license-plate’. The pink box refers to the vehicle while the green one refers to the license plate. Image after annotation,

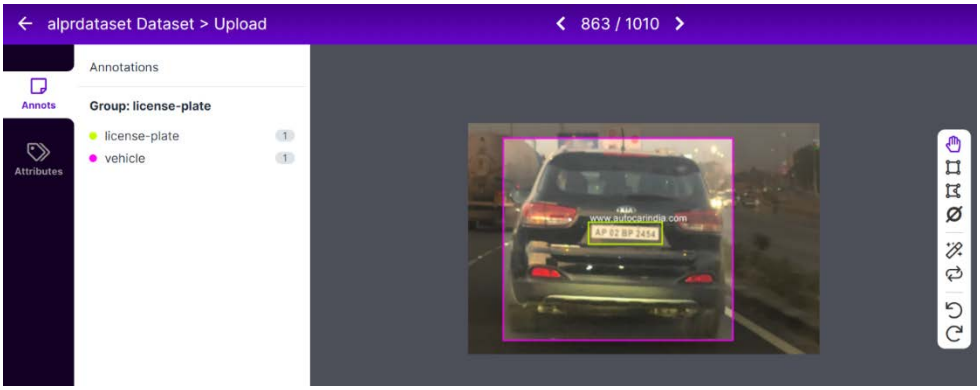


Fig.4.1.2. Annotating the real time image

After annotating all the uploaded images and splitting the images for training(80%), validation(15%) and testing(5%).

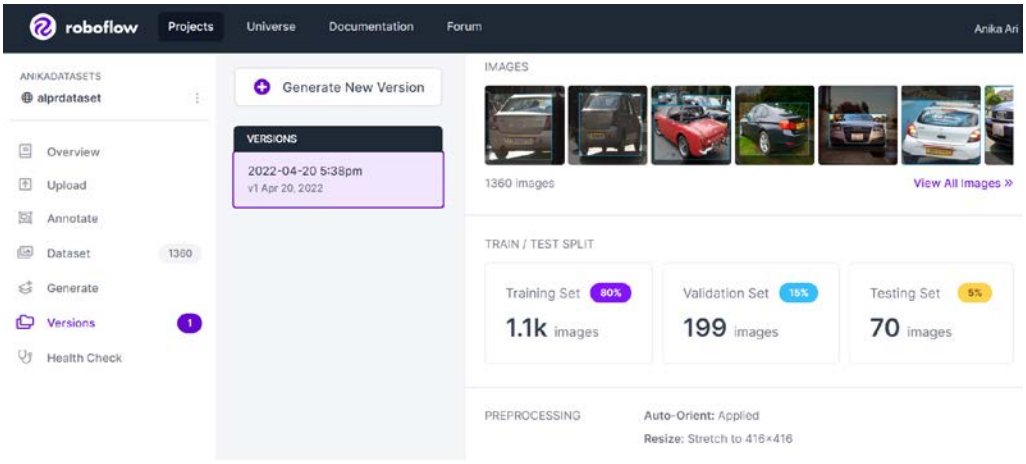


Fig. 4.1.3. Train/Test Split

4.2. MODEL CREATION

(Choose GPU in Runtime if not already selected. Runtime --> Change Runtime Type --> Hardware accelerator --> GPU)

To train our model, first we install YOLOv5 dependencies from the YOLOv5 repository by Ultralytics.

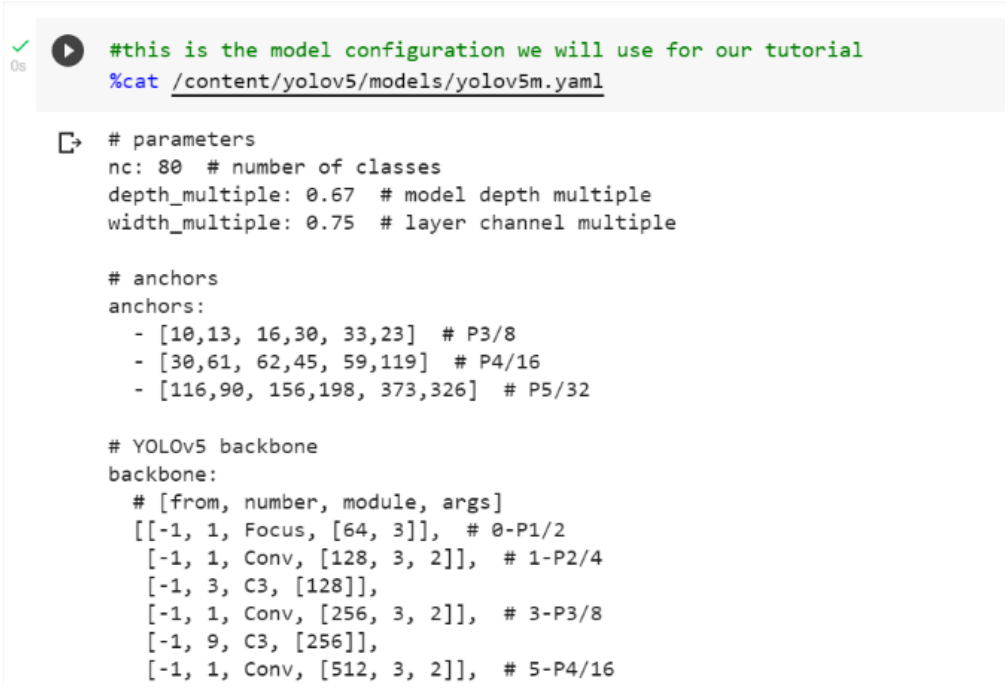
```
[1]  !git clone https://github.com/ultralytics/yolov5  # clone repo

     %cd yolov5
     !git reset --hard 886f1c03d839575afecb059accf74296fad395b6

Cloning into 'yolov5'...
remote: Enumerating objects: 12875, done.
remote: Total 12875 (delta 0), reused 0 (delta 0), pack-reused 12875
Receiving objects: 100% (12875/12875), 11.84 MiB | 31.26 MiB/s, done.
Resolving deltas: 100% (8947/8947), done.
/content/yolov5
HEAD is now at 886f1c0 DDP after autoanchor reorder (#2421)
```

Fig. 4.2.1. Cloning YOLOv5 Ultralytics

After installing other libraries we will install torch and torchvision which are the backbone for training. Python package management system (pip) can be used to install torch and torchvision for CPU.

Fig. 4.2.2. Installing Dependencies

This is the default YOLOv5m model configuration.



```
    [-1, 9, C3, [512]],
    [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
    [-1, 1, SPP, [1024, [5, 9, 13]]],
    [-1, 3, C3, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)


   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Fig. 4.2.3. Default YOLOv5m training configuration

Now we write our custom YOLOv5m training configuration for number plate detection model.

```
[10] #customize iPython writefile so we can write variables
     from IPython.core.magic import register_line_cell_magic

     @register_line_cell_magic
     def writetemplate(line, cell):
         with open(line, 'w') as f:
             f.write(cell.format(**globals()))
```

```
[11] %%writetemplate /content/yolov5/models/custom_yolov5m.yaml

     # parameters
     nc: {num_classes}  # number of classes
     depth_multiple: 0.33  # model depth multiple
     width_multiple: 0.50  # layer channel multiple

     # anchors
     anchors:
       - [10,13, 16,30, 33,23]  # P3/8
       - [30,61, 62,45, 59,119]  # P4/16
       - [116,90, 156,198, 373,326]  # P5/32

     # YOLOv5 backbone
     backbone:
       # [from, number, module, args]
       [[-1, 1, Focus, [64, 3]],  # 0-P1/2
        [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
        [-1, 3, BottleneckCSP, [128]],
        [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
        [-1, 9, BottleneckCSP, [256]],
        [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
        [-1, 9, BottleneckCSP, [512]],
[11]    [-1, 9, BottleneckCSP, [256]],
        [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
        [-1, 9, BottleneckCSP, [512]],
        [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
        [-1, 1, SPP, [1024, [5, 9, 13]]],
        [-1, 3, BottleneckCSP, [1024, False]],  # 9
       ]

     # YOLOv5 head
     head:
       [[-1, 1, Conv, [512, 1, 1]],
        [-1, 1, nn.Upsample, [None, 2, 'nearest']],
        [[-1, 6], 1, Concat, [1]],  # cat backbone P4
        [-1, 3, BottleneckCSP, [512, False]],  # 13

        [-1, 1, Conv, [256, 1, 1]],
        [-1, 1, nn.Upsample, [None, 2, 'nearest']],
        [[-1, 4], 1, Concat, [1]],  # cat backbone P3
        [-1, 3, BottleneckCSP, [256, False]],  # 17 (P3/8-small)

        [-1, 1, Conv, [256, 3, 2]],
        [[-1, 14], 1, Concat, [1]],  # cat head P4
        [-1, 3, BottleneckCSP, [512, False]],  # 20 (P4/16-medium)
[11]    [-1, 1, Conv, [512, 3, 2]],
        [[-1, 10], 1, Concat, [1]],  # cat head P5
        [-1, 3, BottleneckCSP, [1024, False]],  # 23 (P5/32-large)

        [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
       ]
```

Fig. 4.2.4. Custom YOLOv5m Configuration

4.3. MODEL TRAINING

After the setup of our custom YOLOv5m YAML file. We will extract our dataset from roboflow using the “YOLOv5 PyTorch” export format.

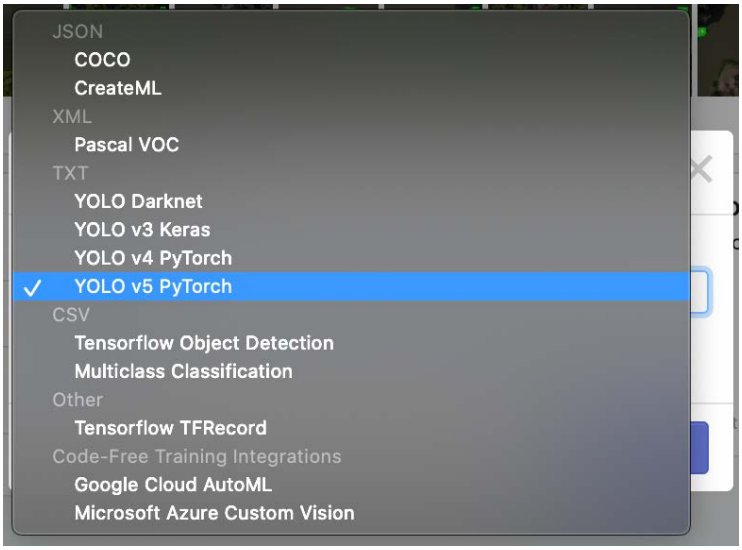


Fig. 4.3.1. Export in YOLOv5 PyTorch format

```
[3]  !pip install -q roboflow
     from roboflow import Roboflow
     rf = Roboflow(model_format="yolov5", notebook="roboflow-yolov5")
```

```
     |████████████████████████████████| 145 kB 5.1 MB/s
     |████████████████████████████████| 178 kB 43.6 MB/s
     |████████████████████████████████| 1.1 MB 52.9 MB/s
     |████████████████████████████████| 67 kB 5.5 MB/s
     |████████████████████████████████| 54 kB 3.3 MB/s
     |████████████████████████████████| 138 kB 60.8 MB/s
     |████████████████████████████████| 63 kB 2.0 MB/s
  Building wheel for roboflow (setup.py) ... done
  Building wheel for wget (setup.py) ... done
```

Fig. 4.3.2. Install Roboflow dependencies

By using the below mentioned API Key we can import our generated dataset from Roboflow.

```
[4]  %cd /content
     !pip install roboflow

     from roboflow import Roboflow
     rf = Roboflow(api_key="IeszVgXe3X8p7GChN8Ar")
     project = rf.workspace("anikadatasets").project("alprdataset")
     dataset = project.version(1).download("yolov5")
```

```
Requirement already satisfied: requests-toolbelt in /usr/local/lib/python3.7/dist-packages (from roboflow) (0.9.1)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.7/dist-packages (from roboflow) (7.1.2)
Requirement already satisfied: chardet==4.0.0 in /usr/local/lib/python3.7/dist-packages (from roboflow) (4.0.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from roboflow) (2.27.1)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.7/dist-packages (from roboflow) (6.0)
Requirement already satisfied: pyparsing==2.4.7 in /usr/local/lib/python3.7/dist-packages (from roboflow) (2.4.7)
Requirement already satisfied: certifi==2021.5.30 in /usr/local/lib/python3.7/dist-packages (from roboflow) (2021.5.30)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from roboflow) (2.8.2)
Requirement already satisfied: urllib3==1.26.6 in /usr/local/lib/python3.7/dist-packages (from roboflow) (1.26.6)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from roboflow) (1.15.0)
Requirement already satisfied: cycler==0.10.0 in /usr/local/lib/python3.7/dist-packages (from roboflow) (0.10.0)
Requirement already satisfied: idna==2.10 in /usr/local/lib/python3.7/dist-packages (from roboflow) (2.10)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.7/dist-packages (from roboflow) (1.21.6)
Requirement already satisfied: kiwisolver==1.3.1 in /usr/local/lib/python3.7/dist-packages (from roboflow) (1.3.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from roboflow) (3.2.2)
Requirement already satisfied: opencv-python>=4.1.2 in /usr/local/lib/python3.7/dist-packages (from roboflow) (4.1.2.30)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.7/dist-packages (from roboflow) (0.20.0)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.7/dist-packages (from roboflow) (4.64.0)
Requirement already satisfied: wget in /usr/local/lib/python3.7/dist-packages (from roboflow) (3.2)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.7/dist-packages (from requests->roboflow) (2.0.12)
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in alprdataset-1 to yolov5pytorch: 100% [37165288 / 37165288] bytes
Extracting Dataset Version Zip to alprdataset-1 in yolov5pytorch:: 100%|██████████| 2732/2732 [00:01<00:00, 1461.24it/s]
```

Fig. 4.3.3. Extracting ZIP Dataset via API key

The url mentioned extracts the data from the zip file.

```
[5]  !curl -L "https://app.roboflow.com/ds/FIgMc9rJep?key=lm2Hcw0cIT" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

```
extracting: valid/labels/video11_4650_jpg.rf.236721602d1d36195c076e5f3b173269.txt
extracting: valid/labels/video11_4800_jpg.rf.74848a56160c9097fe9732f238d35764.txt
extracting: valid/labels/video11_750_jpg.rf.a97f334df4e36be567247463c9ea7165.txt
extracting: valid/labels/video11_860_jpg.rf.c39c24b480615dc82cf4c24927348ce1.txt
extracting: valid/labels/video12_70_jpg.rf.35c140956e089dd44a627f9784ff0d0b.txt
extracting: valid/labels/video2_1590_jpg.rf.85d976acd41c8fb845978804dab03a2d.txt
extracting: valid/labels/video2_1830_jpg.rf.523c80ce1848e32f13d10b68fc7892ba.txt
extracting: valid/labels/video2_2150_jpg.rf.99c77a46da461747a70a55183fba5340.txt
extracting: valid/labels/video2_240_jpg.rf.2e30f3f1ca778ca80ee8f4e40f6a20bf.txt
extracting: valid/labels/video2_3370_jpg.rf.30bec9a7bf61bda9f9182c5fe85cfda4.txt
extracting: valid/labels/video2_3730_jpg.rf.0499f7824f67c761deede54047968aa8.txt
extracting: valid/labels/video2_4080_jpg.rf.920ccee35fbff3c0c970a68be405b855.txt
extracting: valid/labels/video2_820_jpg.rf.ceae1a881659a28fce689289a4a9f00d.txt
extracting: valid/labels/video3_1070_jpg.rf.f1954251bc6045e3db7129858f4cb994.txt
extracting: valid/labels/video3_1100_jpg.rf.120f67b0976c59a1cc767979e9f4f2ed.txt
extracting: valid/labels/video3_1750_jpg.rf.4ac0fbbd51b17c43f9c9bee4982f3672.txt
extracting: valid/labels/video3_1770_jpg.rf.8d3d0b9e4390ff8af1d7a2d182592513.txt
extracting: valid/labels/video3_1780_jpg.rf.26802030771810 91d6db1f5041a000d0.txt
extracting: valid/labels/video3_1850_jpg.rf.eee3cd8ae22df5e49aeab2843e7e04a7.txt
extracting: valid/labels/video3_1890_jpg.rf.07a284db7157a3fe19aafd96c9da29dd.txt
extracting: valid/labels/video3_1920_jpg.rf.c59cce8e8eed8dc9dca47009a327ced3.txt
extracting: valid/labels/video3_1990_jpg.rf.13961a4d70f7f951d705985bdaf87a5b.txt
```

Fig. 4.3.4. Extracting Files from the ZIP Dataset

After extracting the yaml file for training we read the classes (annotated labels) present in the dataset.

```
[6]  # this is the YAML file Roboflow wrote for us that we're loading into this notebook with our data
     %cat data.yaml

     train: ../train/images
     val: ../valid/images

     nc: 2
     names: ['license-plate', 'vehicle']
```

Fig. 4.3.5. Number of classes present in the dataset

We now train our detection model, we will pass the below arguments for training

- img: define input image size
- batch: determine batch size
- epochs: define the number of training epochs.
- data: set the path to our yaml file
- cfg: specify our model configuration
- weights: specify a custom path to weights
- name: result names
- nosave: only save the final checkpoint
- cache: cache images for faster training

```
# train yolov5m on custom data for 300 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 32 --epochs 300 --data '../data.yaml' --cfg ./models/custom_yolov5m.yaml --weights '' --name yolov5m_results  --ca

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
   290/299     3.27G   0.02337   0.01258 0.0008615   0.03681         6       416: 100% 35/35 [00:08<00:00,  4.22it/s]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.72it/s]
                  all         199         364       0.866       0.784       0.823       0.524

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
   291/299     3.27G   0.02337   0.01224  0.000911   0.03652        12       416: 100% 35/35 [00:08<00:00,  4.20it/s]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.62it/s]
                  all         199         364       0.885       0.767       0.817       0.521

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
   292/299     3.27G    0.0242   0.01279  0.001016   0.03801         9       416: 100% 35/35 [00:08<00:00,  4.23it/s]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.69it/s]
                  all         199         364       0.887       0.751       0.815       0.519
```

```
      Epoch    gpu_mem       box       obj       cls     total   targets  img_size
    293/299      3.27G   0.02449   0.01259 0.0009134   0.03799        12       416: 100% 35/35 [00:08<00:00,  4.22it/s]
                 Class     Images     Targets         P         R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.75it/s]
                   all        199         364     0.854     0.783      0.818      0.531

      Epoch    gpu_mem       box       obj       cls     total   targets  img_size
    294/299      3.27G   0.02433   0.01195  0.001012   0.03729         8       416: 100% 35/35 [00:08<00:00,  4.19it/s]
                 Class     Images     Targets         P         R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.69it/s]
                   all        199         364     0.853      0.79      0.817      0.537

      Epoch    gpu_mem       box       obj       cls     total   targets  img_size
    295/299      3.27G   0.02382   0.01195 0.0008227   0.03659         7       416: 100% 35/35 [00:08<00:00,  4.21it/s]
                 Class     Images     Targets         P         R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.69it/s]
                   all        199         364     0.835     0.789       0.81      0.534

      Epoch    gpu_mem       box       obj       cls     total   targets  img_size
    296/299      3.27G   0.02453   0.01233  0.000822   0.03768        15       416: 100% 35/35 [00:08<00:00,  4.23it/s]
                 Class     Images     Targets         P         R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.64it/s]
                   all        199         364     0.899     0.737      0.806      0.521

      Epoch    gpu_mem       box       obj       cls     total   targets  img_size
    297/299      3.27G   0.02386   0.01228 0.0008224   0.03696         7       416: 100% 35/35 [00:08<00:00,  4.22it/s]
                 Class     Images     Targets         P         R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.64it/s]
                   all        199         364     0.871     0.777      0.823      0.529

      Epoch    gpu_mem       box       obj       cls     total   targets  img_size
    298/299      3.27G   0.02358   0.01187 0.0008469    0.0363        13       416: 100% 35/35 [00:08<00:00,  4.20it/s]
                 Class     Images     Targets         P         R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:01<00:00,  3.67it/s]
                   all        199         364     0.853     0.799      0.828      0.531

      Epoch    gpu_mem       box       obj       cls     total   targets  img_size
    299/299      3.27G   0.02411   0.01193  0.000912   0.03695         9       416: 100% 35/35 [00:08<00:00,  4.25it/s]
                 Class     Images     Targets         P         R      mAP@.5  mAP@.5:.95: 100% 4/4 [00:02<00:00,  1.83it/s]
                   all        199         364     0.857     0.794      0.827      0.536
         license-plate        199         218     0.942     0.917      0.941      0.561
               vehicle        199         146     0.772     0.671      0.712      0.512
Optimizer stripped from runs/train/yolov5m_results/weights/last.pt, 14.7MB
Optimizer stripped from runs/train/yolov5m_results/weights/best.pt, 14.7MB
300 epochs completed in 0.816 hours.
```

Fig. 4.3.6. Trained Model

Our license plate detection model is now trained with 300 epochs in 0.816 hours with two pytorch weight files,

Fig. 4.3.7. Ground Truth Training Data

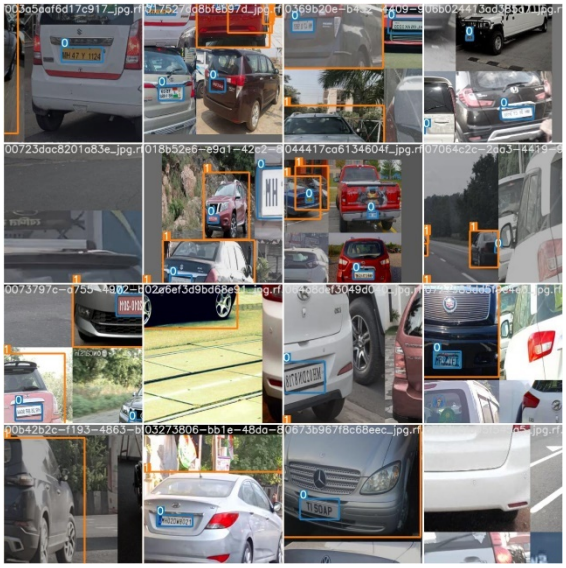


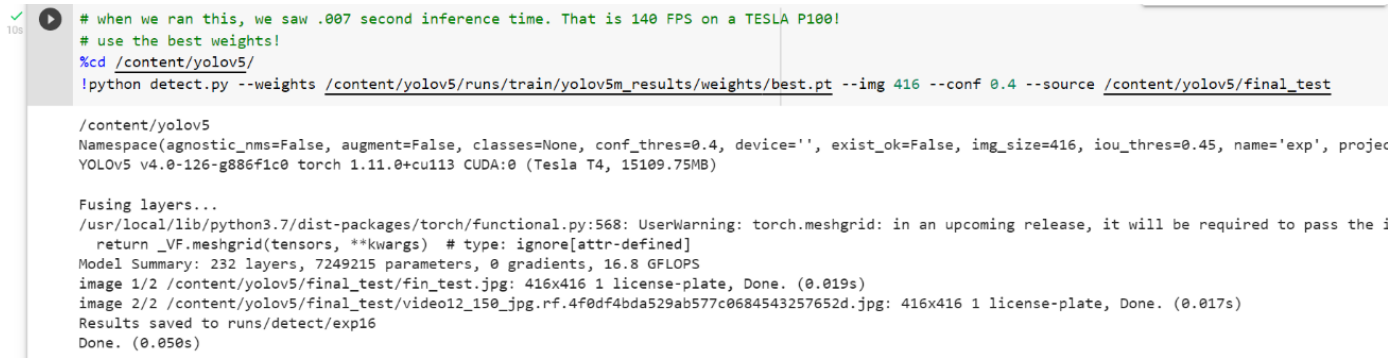
Fig. 4.3.8. Ground Truth Augmented Training Data

Fig. 4.3.9. Predicted Label Data

4.4. LICENSE PLATE DETECTION

Now we test our model on testing data using the using YOLOv5 which uses multiple feature maps with different scales. The size of the candidate box for each feature map is different, which improves the model's ability to recognize small objects. The reason why YOLOv5 is selected for vehicle detection is that it has a fast speed and adaptability to the objects with multi-scales.

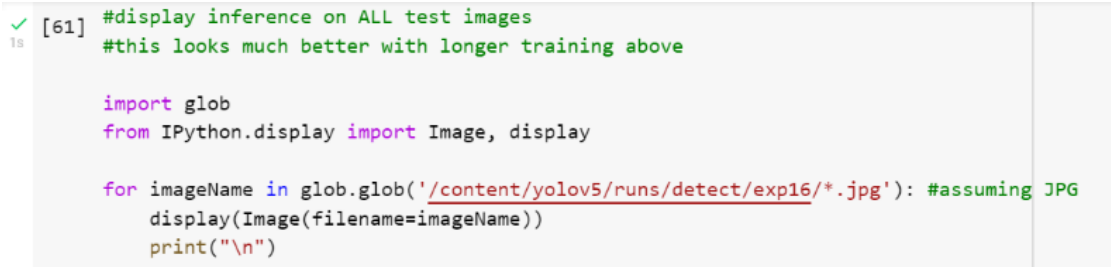
Now we will test our trained model (best.pt) with an unseen image.



```
# when we ran this, we saw .007 second inference time. That is 140 FPS on a TESLA P100!
# use the best weights!
%cd /content/yolov5/
!python detect.py --weights /content/yolov5/runs/train/yolov5m_results/weights/best.pt --img 416 --conf 0.4 --source /content/yolov5/final_test

/content/yolov5
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', exist_ok=False, img_size=416, iou_thres=0.45, name='exp', projec
YOLOv5 v4.0-126-g886f1c0 torch 1.11.0+cu113 CUDA:0 (Tesla T4, 15109.75MB)

Fusing layers...
/usr/local/lib/python3.7/dist-packages/torch/functional.py:568: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the i
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 232 layers, 7249215 parameters, 0 gradients, 16.8 GFLOPS
image 1/2 /content/yolov5/final_test/fin_test.jpg: 416x416 1 license-plate, Done. (0.019s)
image 2/2 /content/yolov5/final_test/video12_150_jpg.rf.4f0df4bda529ab577c0684543257652d.jpg: 416x416 1 license-plate, Done. (0.017s)
Results saved to runs/detect/exp16
Done. (0.050s)
```

Fig. 4.4.1. Testing our trained model with best.pt

The results are now saved in 'runs/detect/exp16', the below snippet of code shall display the results.



```
[61] #display inference on ALL test images
     #this looks much better with longer training above

     import glob
     from IPython.display import Image, display

     for imageName in glob.glob('/content/yolov5/runs/detect/exp16/*.jpg'): #assuming JPG
         display(Image(filename=imageName))
         print("\n")
```

Fig. 4.4.2. Images after license plate detection

To save the bounding box coordinate in text file, we run the detect.py file along with our test folder using –save-txt.

```
[62] # use --save txt
     %cd /content/yolov5/
     !python detect.py --weights /content/yolov5/runs/train/yolov5m_results/weights/best.pt --img 416 --conf 0.4 --source /content/yolov5/final_test --save

/content/yolov5
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', exist_ok=False, img_size=416, iou_thres=0.45, name='exp', project
YOLOv5 v4.0-126-g886f1c0 torch 1.11.0+cu113 CUDA:0 (Tesla T4, 15109.75MB)

Fusing layers...
/usr/local/lib/python3.7/dist-packages/torch/functional.py:568: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the in
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 232 layers, 7249215 parameters, 0 gradients, 16.8 GFLOPS
image 1/2 /content/yolov5/final_test/fin_test.jpg: 416x416 1 license-plate, Done. (0.016s)
image 2/2 /content/yolov5/final_test/video12_150_jpg.rf.4f0df4bda529ab577c0684543257652d.jpg: 416x416 1 license-plate, Done. (0.016s)
Results saved to runs/detect/exp17
2 labels saved to runs/detect/exp17/labels
Done. (0.047s)
```

Fig. 4.4.3. Save txt

The results are now saved into ‘runs/detect/exp17’ folder.

To crop the license plate in the bounding box we have added a new detect.py file with some extra lines of code to crop the bounding box and save the results in the created folder ‘test_images/results’.

```
%cd /content/yolov5/
!python detect1.py --weights /content/yolov5/runs/train/yolov5m_results/weights/best.pt --img 416 --conf 0.4 --source /content/yolov5/final_test/fin_t

/content/yolov5
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', exist_ok=False, img_size=416, iou_thres=0.45, name='exp', projec
YOLOv5 v4.0-126-g886f1c0 torch 1.11.0+cu113 CUDA:0 (Tesla T4, 15109.75MB)

Fusing layers...
/usr/local/lib/python3.7/dist-packages/torch/functional.py:568: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the i
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 232 layers, 7249215 parameters, 0 gradients, 16.8 GFLOPS
image 1/1 /content/yolov5/final_test/fin_test.jpg: test_images/results/fin_test.jpg
There is no detected object
416x416 1 license-plate, Done. (0.017s)
Results saved to runs/detect/exp20
Done. (0.026s)
```

Fig. 4.4.4. Cropping detected license plate

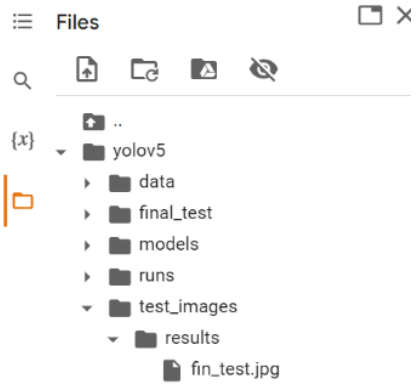


Fig. 4.4.5. Cropped image saved in files

To display the cropped image from the bounding box,

```
#display inference on ALL test images
#this looks much better with longer training above

import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/test_images/results/*.jpg'): #assuming JPG
    display(Image(filename=imageName))
    print("\n")
```

The cropped license plate image obtained.

Fig. 4.4.6. Cropped license plate

4.5. LICENSE PLATE RECOGNITON

First requirement to apply OCR is to crop individual character images from license plates. Most of the time the license plate cut out is not suitable for direct character segmentation. Thus, unwanted symbols were to be removed from the license plate. Image processing steps described earlier will be repeated to remove noise in the plate image. After LP Recognition we will be provided the recognised license plate number for each identified vehicle.

Now we will be installing pytesseract and tesseract-ocr for applying optical character recognition(OCR) on our cropped license plate image.

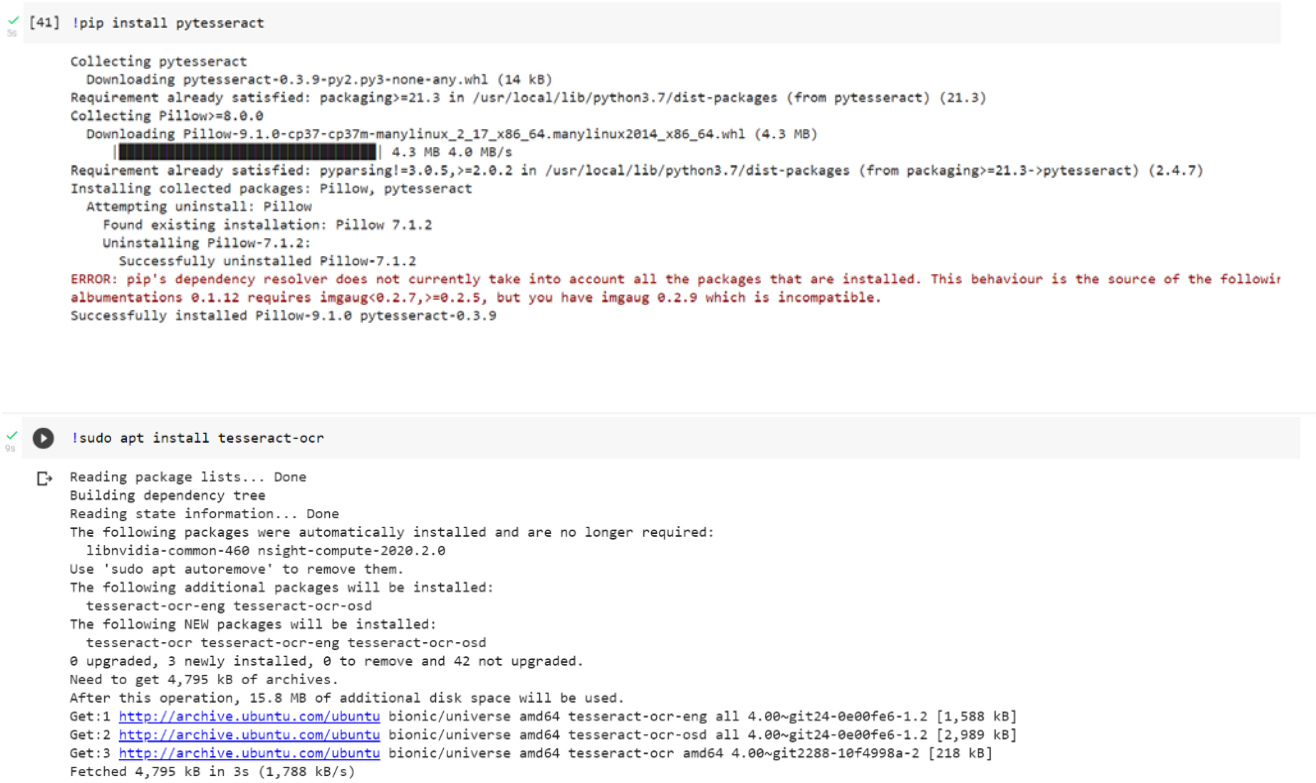


Fig. 4.5.1. Install pytesseract and tesseract OCR

After importing cv2 to convert the image to string using pytesseract, we can see that few of the predicted license plate are inaccurate due to angles, noise and font formats.

```
[43] import pytesseract
     import cv2
```

```
from IPython.display import Image, display
display(Image('/content/yolov5/test_images/results/GJ15CD0564.jpg'))
text1 = pytesseract.image_to_string("/content/yolov5/test_images/results/GJ15CD0564.jpg")
print(text1)
display(Image('/content/yolov5/test_images/results/KL55R2473.jpg'))
text2 = pytesseract.image_to_string("/content/yolov5/test_images/results/KL55R2473.jpg")
print(text2)
display(Image('/content/yolov5/test_images/results/MH04BQ1677.jpg'))
text4 = pytesseract.image_to_string("/content/yolov5/test_images/results/MH04BQ1677.jpg")
print(text4)
display(Image('/content/yolov5/test_images/results/TN30BH45.jpg'))
text5 = pytesseract.image_to_string("/content/yolov5/test_images/results/TN30BH45.jpg")
print(text5)
display(Image('/content/yolov5/test_images/results/TN70T3789.jpg'))
text6 = pytesseract.image_to_string("/content/yolov5/test_images/results/TN70T3789.jpg")
print(text6)
display(Image('/content/yolov5/test_images/results/TS08FM8888.jpg'))
text7 = pytesseract.image_to_string("/content/yolov5/test_images/results/TS08FM8888.jpg")
print(text7)
```

Fig. 4.5.2. Predicted license plate numbers using Easy OCR

Now we will be using OpenCV which is an open source computer vision library to recognize the license plate. The library has more than 2500 optimized algorithms. These algorithms are often used to search and recognize faces, identify objects, recognize scenery and generate markers to overlay images using augmented reality, etc.

```
[ ] !pip install opencv-python

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
    Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.21.6)

[4] # Loading the required python modules
    import pytesseract # this is tesseract module
    import matplotlib.pyplot as plt
    import cv2 # this is opencv module
    import glob
    import os
```

Fig. 4.5.3. Installing required dependencies

After installing the required dependencies, we are creating a list with the actual license plate numbers for future comparison.

```
[37] # specify path to the license plate images folder as shown below
     #path_for_license_plates = os.getcwd() + "/test_images/results/*.jpg"  #/license-plates/**/*.jpg
     path_for_license_plates = '/content/yolov5/test_images/results/*.jpg'
     actual_lp = []
     predicted_lp = []

     for path_to_license_plate in glob.glob(path_for_license_plates, recursive = True):

       license_plate_file = path_to_license_plate.split("/")[-1]
       license_plate, _ = os.path.splitext(license_plate_file)
       actual_lp.append(license_plate)

     print(actual_lp)
     print(predicted_lp)

     ['TS08FM8888', 'GJ15CD0564', 'MH04BQ1677', 'TN70T3789', 'KL55R2473', 'TN30BH45']
     []

[39] from google.colab.patches import cv2_imshow
```

Fig. 4.5.4. List containing actual license plate numbers

For the license plates, the Tesseract OCR Engine predicted incorrectly and others we will apply image processing techniques on those license plate files and pass them to the Tesseract OCR again. Applying the image processing techniques.

Image resizing:

Resize the image file by a factor of 2x in both the horizontal and vertical directions using cv2.resize

Converting to Gray-scale:

Next, we convert our resized image file to gray scale to optimize the detection and reduce the amount of colors present in image drastically which will help in the detection of license plates easily.

Denoising the Image :

Gaussian Blur is a technique for denoising images. It makes the edges more clearer and smoother which in-turn makes the characters more readable.

```
[40] path='/content/yolov5/test_images/results/TS08FM8888.jpg'
     img = cv2.imread(path)
     cv2_imshow(img)
     grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
     #cv2_imshow(grey_image)
     blur = cv2.medianBlur(grey_image, 3)
     #cv2_imshow(blur)
     resized = cv2.resize(blur, None, fx=2, fy=2, interpolation=cv2.INTER_NEAREST)
     #cv2_imshow(resized)
     thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
     #cv2_imshow(thresh)
     predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01
     filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
     new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
     print('License Plate Number   :   ' +new_res)
     predicted_lp.append(new_res)
```


```
License Plate Number   :   TS08FM8888
```

```
[41] path='/content/yolov5/test_images/results/GJ15CD0564.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=6, fy=5, interpolation=cv2.INTER_CUBIC)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
predicted_lp.append(new_res)
```


```
License Plate Number   :   GJ15CD0564
```

```
[42] path='/content/yolov5/test_images/results/MH04BQ1677.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=3, fy=4, interpolation=cv2.INTER_AREA)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
predicted_lp.append(new_res)
```


```
License Plate Number   :   MH04BQ1677
```

```
path='/content/yolov5/test_images/results/TN70T3789.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=6, fy=5, interpolation=cv2.INTER_CUBIC)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
#predicted_lp.append(filter_predicted_result)
predicted_lp.append(new_res)
```


```
License Plate Number   :   3789
```

```
path='/content/yolov5/test_images/results/KL55R2473.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=2, fy=1, interpolation=cv2.INTER_LINEAR)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
#predicted_lp.append(filter_predicted_result)
predicted_lp.append(new_res)
```


```
License Plate Number   :   KL55R2473
```

```
path='/content/yolov5/test_images/results/TN30BH45.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=3, fy=1, interpolation=cv2.INTER_LINEAR)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
predicted_lp.append(new_res)
```


```
License Plate Number   :   TN30BH45
```

Fig.4.5.5. Predicting license plate number using pytesseract

```
[46] print(actual_lp)
     print(predicted_lp)
```

```
['TS08FM8888', 'GJ15CD0564', 'MH04BQ1677', 'TN70T3789', 'KL55R2473', 'TN30BH45']
['TS08FM8888', 'GJ15CD0564', 'MHO4BQ1677', '3789', 'KL55R2473', 'TN30BH45']
```

```
import pandas as pd
df = pd.read_csv('/content/LPDets.csv')
print(df)
```

	A	B	C	D	E	F
1	LicensePlate	VehicleModel	OwnerName	OwnerRegNo	OwnerPhoneNo	StateofRegistration
2	TS08FM8888	Honda	Akash Raj	201893459	9987659032	Telangana
3	GJ15CD0564	Audi	Dev Nand	201845799	8757499302	Gujarat
4	MH04BQ1677	MercedesBenz	Ashlyn Ella	201914790	5749959392	Maharashtra
5	TN70T3789	KIA	Chris Bang	202056789	5750204802	TamilNadu
6	KL55R2473	Renault	Elle Lively	202197884	8753948022	Kerala
7	TN30BH45	BMW	Felix Lee	202210159	8248022042	TamilNadu
8						

Fig. 4.5.6. LPDets.csv

As we can see that almost all of the license plate numbers have been predicted almost correct.

Now we will be taking user's input (LP Number) and will be printing its vehicle and owner details which is stored in the dataset (LPDets.csv).

```
[58] import csv
     import sys
     number = input('Enter LP Number : \n')
     csv_file = csv.reader(open('/content/LPDets.csv', "r"), delimiter=",")
     list_of_column_names = list(df.columns)
     for row in csv_file:
         if number == row[0]:
              print(list_of_column_names)
              print (row)
```

```
Enter LP Number :
TN30BH45
['LicensePlate', 'VehicleModel', 'OwnerName', 'OwnerRegNo', 'OwnerPhoneNo', 'StateofRegistration']
['TN30BH45', 'BMW', 'Felix Lee', '202210159', '8248022042', 'TamilNadu']
```

Fig. 4.5.7. Identified vehicle details

Here the entered input is TH30BH45 and the output displays the owner and vehicle details.

5. TEST CASES AND VALIDATION

Case 1: Frontal view of the vehicle with license plate

Fig. 5.1.1. Input Image

Fig. 5.1.2. Image after license plate detection

Fig. 5.1.3. Cropped License plate image

```
display(Image('/content/yolov5/test_images/results/TS08FM8888.jpg'))
text7 = pytesseract.image_to_string("/content/yolov5/test_images/results/TS08FM8888.jpg")
print(text7)
```

TS 08 FM 8888

Fig. 5.1.4. License Plate Recognition using Easy OCR

```
path='/content/yolov5/test_images/results/TS08FM8888.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=2, fy=2, interpolation=cv2.INTER_NEAREST)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
predicted_lp.append(new_res)
```


```
License Plate Number   :   TS08FM8888
```

Fig. 5.1.5. License Plate Recognition using OpenCV and Pytesseract.

Case 2: Back view of the vehicle with license plate

Fig. 5.2.1. Input Image

Fig. 5.2.2. Image after license plate detection

Fig. 5.2.3. Cropped License plate image

```
display(Image('/content/yolov5/test_images/results/KL55R2473.jpg'))
text2 = pytesseract.image_to_string("/content/yolov5/test_images/results/KL55R2473.jpg")
print(text2)
```


```
KLS55R 2473
```

Fig. 5.2.4. License Plate Recognition using Easy OCR

```
path='/content/yolov5/test_images/results/KL55R2473.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=2, fy=1, interpolation=cv2.INTER_LINEAR)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ012345
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
#predicted_lp.append(filter_predicted_result)
predicted_lp.append(new_res)
```


```
License Plate Number   :   KL55R2473
```

Fig. 5.2.5. License Plate Recognition using OpenCV and Pytesseract.

Case 3: Cross angled license plate

Fig. 5.3.1. Input Image

Fig. 5.3.2. Image after license plate detection

Fig. 5.3.3. Cropped License plate image

```
display(Image('/content/yolov5/test_images/results/GJ15CD0564.jpg'))
text1 = pytesseract.image_to_string("/content/yolov5/test_images/results/GJ15CD0564.jpg")
print(text1)
```

Fig. 5.3.4. License Plate Recognition using Easy OCR

```
path='/content/yolov5/test_images/results/GJ15CD0564.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=6, fy=5, interpolation=cv2.INTER_CUBIC)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
predicted_lp.append(new_res)
```


```
License Plate Number   :   GJ15CD0564
```

Fig. 5.3.5. License Plate Recognition using OpenCV and Pytesseract.

Case 4: License plate with different font

Fig. 5.4.1. Input Image

Fig. 5.4.2. Image after license plate detection

Fig. 5.4.3. Cropped License plate image

```
display(Image('/content/yolov5/test_images/results/MH04BQ1677.jpg'))
text4 = pytesseract.image_to_string("/content/yolov5/test_images/results/MH04BQ1677.jpg")
print(text4)
```


```
MHOA BQ i677
```

Fig. 5.4.4. License Plate Recognition using Easy OCR

```
path='/content/yolov5/test_images/results/MH04BQ1677.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=3, fy=4, interpolation=cv2.INTER_AREA)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
predicted_lp.append(new_res)
```

License Plate Number : MH04BQ1677

Fig. 5.4.5. License Plate Recognition using OpenCV and Pytesseract.

Case 5: Fancy license plate

Fig. 5.5.1. Input Image

Fig. 5.5.2. Image after license plate detection

Fig. 5.5.3. Cropped License plate image

```
display(Image('/content/yolov5/test_images/results/TN30BH45.jpg'))
text5 = pytesseract.image_to_string("/content/yolov5/test_images/results/TN30BH45.jpg")
print(text5)
```

Fig. 5.5.4. License Plate Recognition using Easy OCR

```
path='/content/yolov5/test_images/results/TN30BH45.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=3, fy=1, interpolation=cv2.INTER_LINEAR)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number   :   ' +new_res)
predicted_lp.append(new_res)
```

License Plate Number : TN30BH45

Fig. 5.5.5. License Plate Recognition using OpenCV and Pytesseract.

Case 6: License plate with different format

Fig. 5.6.1. Input Image

Fig. 5.6.2. Image after license plate detection

Fig. 5.6.3. Cropped License plate image

```
display(Image('/content/yolov5/test_images/results/TN70T3789.jpg'))
text6 = pytesseract.image_to_string("/content/yolov5/test_images/results/TN70T3789.jpg")
print(text6)
```


```
"TN70T*

_ 3789
```

Fig. 5.6.4. License Plate Recognition using Easy OCR

```
path='/content/yolov5/test_images/results/TN70T3789.jpg'
img = cv2.imread(path)
cv2_imshow(img)
grey_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#cv2_imshow(grey_image)
blur = cv2.medianBlur(grey_image, 3)
#cv2_imshow(blur)
resized = cv2.resize(blur, None, fx=6, fy=5, interpolation=cv2.INTER_CUBIC)
#cv2_imshow(resized)
thresh = cv2.threshold(resized, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
#cv2_imshow(thresh)
predicted_result = pytesseract.image_to_string(thresh, lang='eng',config='--oem 3  --psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ01234
filter_predicted_result = "".join(predicted_result.split()).replace(":", "").replace("-", "")
new_res =  "".join(str for str in filter_predicted_result.strip() if str.isalnum())
print('License Plate Number    :    ' +new_res)
#predicted_lp.append(filter_predicted_result)
predicted_lp.append(new_res)
```


```
License Plate Number    :    3789
```

Fig. 5.6.5. License Plate Recognition using OpenCV and Pytesseract.

6. EVALUATION METRICS

6.1. INTEREST OVER UNION (IoU)

Interest over Union measures the area of overlap between two rectangles (prediction and ground truth). Calculating IoU is clearly given below

$$Intersection\ over\ Union\ (IoU) = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

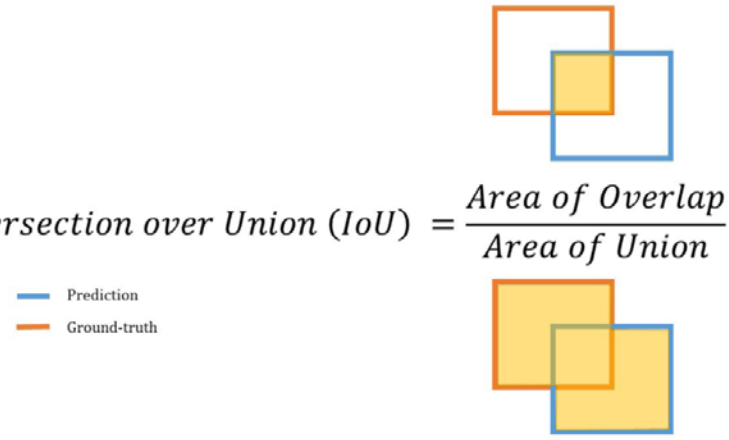


Fig. 6.1.1. Computing Intersect over Union between these bounding boxes

The ideal case of IoU is 1 because in this case, the overlap of ground truth bounding box and predicted bounding box is a perfect match. The worst case is 0. Normally, in the object detection model, the common threshold value is 0.5 to consider the prediction is correct. Otherwise, the prediction is incorrect.

If IoU is greater than or equal to 0.5, then the prediction is correct otherwise it's incorrect.

6.2. MEAN AVERAGE PRECISION (mAP)

In most object detection cases, we need to detect multiple objects in an image. So we need to calculate average precision for all the classes. We calculate the mean for all average precision. This is called mean average precision. This is the metric that is often used in object detection.

The mAP at 0.5 and the mAP at 0.5:0.95 are plotted for 300 epochs. mAP@[0.5:0.95] means average mAP over different IoU thresholds, from 0.5 to 0.95, step 0.05 (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95).

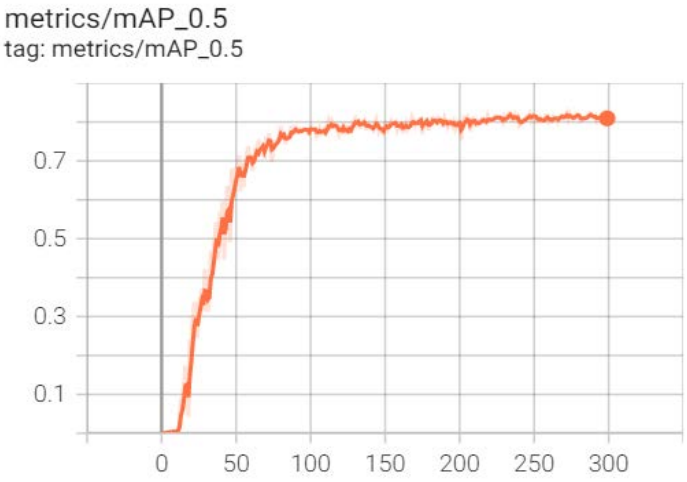


Fig. 6.2.1. mAP @ 0.5

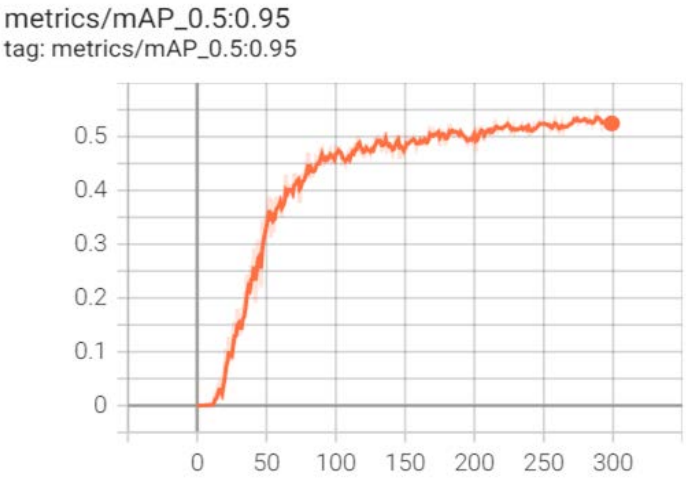


Fig. 6.2.2. mAP @ 0.5:0.95

6.3. CONFUSION MATRIX

A confusion matrix is a technique for summarizing the performance of a classification algorithm. It is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The dataset which we have used for training has two class labels i.e detecting vehicle and license plate.

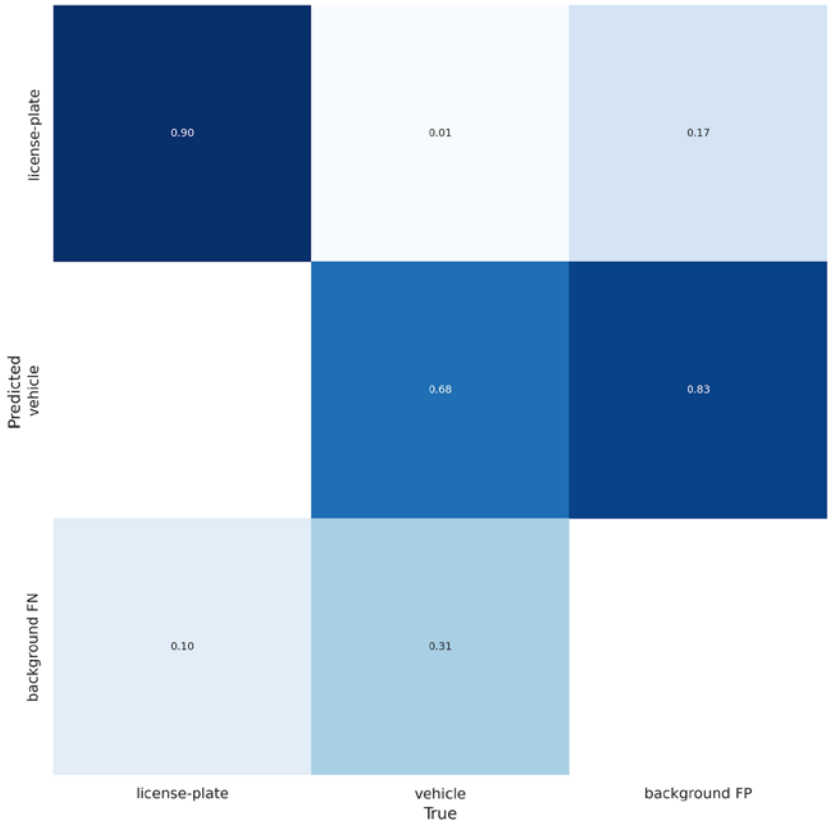


Fig. 6.3.1. Confusion Matrix of our model

Our trained model detects license plate 90% correctly as license plate.

7. INNOVATIVE PLAN

The proposed model revolves around the idea of ALPR - Automatic License Plate Recognition, but implementing it in an advanced high performing recognition manner and with an utilizable potential. ALPR is used to automatically recognise number plates by performing an advanced real-time object detection system YOLOv5 which divides all the given input images into the SxS grid system. For sophisticated character recognition on images, we

have croppend the region of interest i.e, license plate and it aims at exploiting image processing and pattern recognition techniques to extract and recognise the characters on license plates.

Despite various predominant ALPR techniques and their source codes (making use of YOLOv3 etc) readily available to us, certain tuning is necessary for the project to be implemented on a larger scale in India. In this project, we tackle the constraints and make it appropriate for the Indian roads. There is a bright scope for this project due to its application in speed cameras, toll cameras and parking cameras which ultimately is beneficial to the Road and Transport Authority (RTA) and the Indian public. This project's base idea opens up a wide prospect for the Indian masses to get creative and implement/utilize it in an effective manner.

8. CONCLUSION AND FUTURE WORKS

Technologies towards smart vehicles, smart cities, and intelligent transportation systems continue to transform many facets of human life. As a consequence, technologies such as automatic license plate recognition (ALPR) have become part of our everyday activities. Moreover, the concept of ALPR is promising to contribute towards various use cases while eliminating human intervention. ALPR offers numerous advantages that are the basis for real-world applications where the detection, identification, or localization of vehicles is important. Most benefits of ALPR come with automating manual tasks, highly efficient space management, governance, and increasing the customer experience. In conclusion ALPR is already an effective tool for law enforcement, and like all technology, it continues to improve rapidly with time.

Further this project can be integrated with the vahan portal for all registered vehicles in India for monitoring the record of fines/instances of reckless driving by the vehicle, this includes over speeding, cutting the red signal, driving on lanes other than lanes dedicated to the vehicles and such. This may also help traffic police monitor traffic violators/violations and the fines will be computerized eliminating bribes collected by cops and direct payment on the vahan portal means the government could make use of these funds for better infrastructure.

REFERENCES

1. Ali, F., Rathor, H., & Akram, W. (2021). License Plate Recognition System. 2021 International Conference on Advance Computing and Innovative Technologies in Engineering, ICACITE 2021, 1053–1055. https://doi.org/10.1109/ICACITE51222.2021.9404706
2. Hendry, & Chen, R. C. (2019). Automatic License Plate Recognition via sliding-window darknet-YOLO deep learning. Image and Vision Computing, 87, 47–56. https://doi.org/10.1016/j.imavis.2019.04.007
3. Khinchi, M., & Agarwal, C. (2019). A review on automatic number plate recognition technology and methods. Proceedings of the International Conference on Intelligent Sustainable Systems, ICISS 2019, Iciss, 363–366. https://doi.org/10.1109/ISS1.2019.8908014
4. Lin, C. H., Lin, Y. S., & Liu, W. C. (2018). An efficient license plate recognition system using convolution neural networks. Proceedings of 4th

IEEE International Conference on Applied System Innovation 2018, ICASI 2018, 224–227. https://doi.org/10.1109/ICASI.2018.8394573

5.Naren Babu, R., Sowmya, V., & Soman, K. P. (2019). Indian Car Number Plate Recognition using Deep Learning. 2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies, ICICICT 2019, 1269–1272. https://doi.org/10.1109/ICICICT46008.2019.8993238

6. Silva, S. M., & Jung, C. R. (2021). A Flexible Approach for Automatic License Plate Recognition in Unconstrained Scenarios. IEEE Transactions on Intelligent Transportation Systems, 1–11. https://doi.org/10.1109/TITS.2021.3055946

7. Sp, V. V. R., Sathyasri, B., Balaji, A., Vanaja, S., Krishnan, R., & Deepika, Y. (2021). Automatic Number Plate Recognition System for Entry and Exit Management. Proceedings of the 6th International Conference on Communication and Electronics Systems, ICCES 2021. https://doi.org/10.1109/ICCES51350.2021.9489018

8. Varma, P. R. K., Ganta, S., Hari Krishna, B., & Svsrk, P. (2020). A Novel Method for Indian Vehicle Registration Number Plate Detection and Recognition using Image Processing Techniques. Procedia Computer Science, 167(2019), 2623–2633. https://doi.org/10.1016/j.procs.2020.03.324

9. Xu, C., Zhang, H., Wang, W., & Qiu, J. (2020). License Plate Recognition System Based on Deep Learning. Proceedings of 2020 IEEE International Conference on Artificial Intelligence and Computer Applications, ICAICA 2020, 1300–1303. https://doi.org/10.1109/ICAICA50127.2020.9182382