

HELMET WEAR DETECTION USING MACHINE LEARNING

Submitted in partial fulfillment of the requirements

For the award of

Bachelor of Engineering degree in Computer Science and Engineering by

DARBHA SOMA SEKHAR SARMA

(Reg. No. 37110171)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC JEPPIAAR NAGAR,
RAJIV GANDHI SALAI, CHENNAI – 600 119**

MARCH - 2021



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with "A" grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119
www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **DARBHA SOMA SEKHAR SARMA** (Reg. No. 37110171) who carried out the project entitled "**Helmet Detection and Number Plate Extraction using Machine learning**" under our supervision from November, 2020 to April 2021.

Internal Guide

Dr. A. Mary Posonia, M.E., Ph.D.,

Head of the Department

Dr. S. VIGNESHWARI, M.E., Ph.D.,

Dr.L.LAKSHMANAN, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, **Darbha Soma Sekhar Sarma** hereby declare that the Project Report entitled **“Helmet Wear Detection using Machine Learning”** is done by us under the guidance of Associate Professor **Dr. A. Mary Posonia, M.E., Ph.D.**, of the Department of CSE at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATES

1.D. SOMA SEKHAR SARMA

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D.**, Dean, School of Computing, **Dr. S. Vigneshwari, M.E., Ph.D.**, and **Dr. L. Lakshmanan, M.E., Ph.D.**, Heads of the Department, Dept. of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. A. Mary Posonia, M.E., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved the way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department who were helpful in many ways for the completion of the project.

ABSTRACT

The continual motorization of traffic has led to an increase in road accidents across the globe. The government is trying hard to make people follow driving rules to reduce accidents and fatal deaths, especially in India. In our country, the motorcycle is the most used vehicle. As a recommendation, a system for automatically detecting helmet wear is mandatory for road safety, where a significant number of people resulted in the death of not wearing a helmet. This project's main aim is to automatically detect bike-riders without helmets using CCTV videos. As a result, a machine learning-based algorithm is used to generate a unique object detection model that can specify motorcycle riders. The first step is to use the background subtraction technique to identify the bike riders. Then it detects whether the rider is wearing a helmet or not. If the person who is riding the motor-cycle without the helmet is detected then the number plate is extracted from the video frame. Then with the help of an Optical character recognizer, the number plate is recognized.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	V
	LIST OF FIGURES	Viii
	LIST OF TABLES	Ix
1.	INTRODUCTION	1
	1.1 AOBJECTIVE	2
	1.2 LITERATURE AND LITERATURE SURVEY	2
2.	AIM AND SCOPE OF PRESENT INVESTIGATION	4
	2.1 PROPOSED SYSTEM	4
	2.2 WORKING METHODOLOGY	4
	2.2.1 BACKGROUND	5
	MODELING	
	CLASSIFICATION OF MOTORCYCLE	5
	RECOGNITION OF MOTORCYCLE WITHOUT HELMET	5
	EXTRACTION OF NUMBER PLATE	5
3.	IMPLEMENTATION	6
	IMPLEMENTATION	6
	DATA SETS	6
	YOLO	7
	HOW TO ENCODE BOUNDARY BOXES	9
	TESTING	13
	IMPLEMENTATION OF YOLO IN PYTHON	14
4.	RESULTS AND DISCUSSIONS	24

	RESULT	24
5.	CONCLUSION AND FUTURE WORK	25
	CONCLUSION	25
	FUTURE WORK	25
	REFERENCES	26
	APPENDIX	27
	A. SOURCE CODE	
	B. SCREEN SHOTS	

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
1	ROAD ACCIDENTS	1
2.1	SYSTEM ARCHITECTURE	4
3.1.2	TRAINING PROCESS	9
3.1.3	YOLO FINAL DETECTION	11
3.1.5	DETECTION OF OBJECTS	23
4.1	RESULT	24
5.2	FUTURE IMPLEMENTATION	25

CHAPTER 1

INTRODUCTION

According to the statistics given by the transport ministry about 4 people die every hour in India because they do not wear a helmet. The number of two-wheeler users who died in road accidents was 48,746. Incidentally, 73.8 percent of them did not wear a helmet.

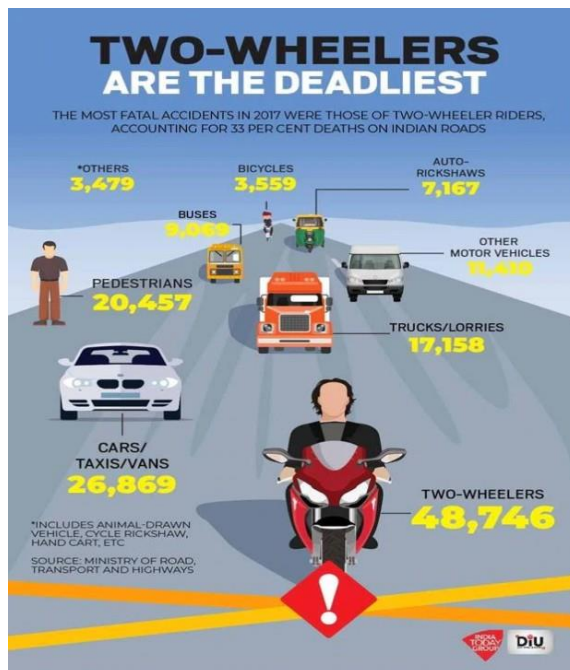


Fig 1.Road Accidents

This project tries to mitigate the aforementioned problems by giving a feasible solution using the detection of continuous traffic video feed. Every vehicle on the street gets evaluated and a database of offenders will be generated real time. Hence every rider who is not wearing a helmet will be prosecuted and this generates a lot of awareness in the public which results in wearing a helmet while riding bikes, this method results in a decrease of bike accidents.

Object Detection is a common Computer Vision problem which deals with identifying and locating objects of certain classes in the image. Interpreting the object localisation can be done in various ways, including creating a bounding box around the object or marking every pixel in the image which

contains the object (called segmentation).

Object detection was studied even before the breakout popularity of CNNs in Computer Vision. While CNNs are capable of automatically extracting more complex and better features, taking a glance at the conventional methods can at worst be a small detour and at best an inspiration. Object detection before Deep Learning was a several step process, starting with edge detection and feature extraction using techniques like SIFT, HOG etc. These images were then compared with existing object templates, usually at multi scale levels, to detect and localize objects present in the image.

Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all of these problems are referred to as object recognition.

Objective

The following are the objectives of this project:

- The objective of this project is to develop an application for enforcing helmet wearing using CCTV cameras.
- The developed application aims to help law enforcement by police, and eventually resulting in changing risk behaviours and consequently reducing the number of accidents and its severity

Background and Literature Survey

According to the Research paper in 2016 titled Automatic Detection of Bike-riders without Helmet using Surveillance Videos in Real-time. The Author from IIT Hyderabad developed the project using the algorithms like object classification, feature extraction and SVM classifier for the result.

Chiverton proposed the use of a circular arc to identify helmets in a video feed, it has very low accuracy. On the other hand, given the number of vehicles on the speed at a given instant, the

computation that is required is very heavy and consumes lots of resources. This method will determine any circular object around the bike rider as a helmet.

Based on the positioning of these corner center peaks in the heatmaps, resulting bounding boxes are predicted. Since a different heatmap can be created for every class, this method also combines detection and classification. While heatmap-based object detection is currently leading new research, it is still not as fast as conventional one-shot object detection algorithms. This is due to the fact that these algorithms require more complex backbone architectures (CNNs) to get respectable accuracy.

Heatmap-based object detection can be, in some sense, considered an extension of one-shot based Object Detection. While one-shot based object detection algorithms try to directly regress the bounding box coordinates (or offsets), heatmap-based object detection provides probability distribution of bounding box corners/center.

Silva proposed a system in which he tracks the vehicles using Kalman filter. An important advantage of this Kalman tracking system is the ability to continue to track objects even if they are lightly occluded but when there were more than two or three motorcyclists appear in a same frame, Kalman filter fails because Kalman filter mostly works well for linear state transitions (i.e tracking single objects/one object at a time). But to track multiple objects, we need non-linear functions to track them.

CHAPTER 2

AIM AND SCOPE OF PRESENT INVESTIGATION

PROPOSED SYSTEM

The following block diagram (figure 2.1) shows the system architecture of this project.

ARCHITECTURE:

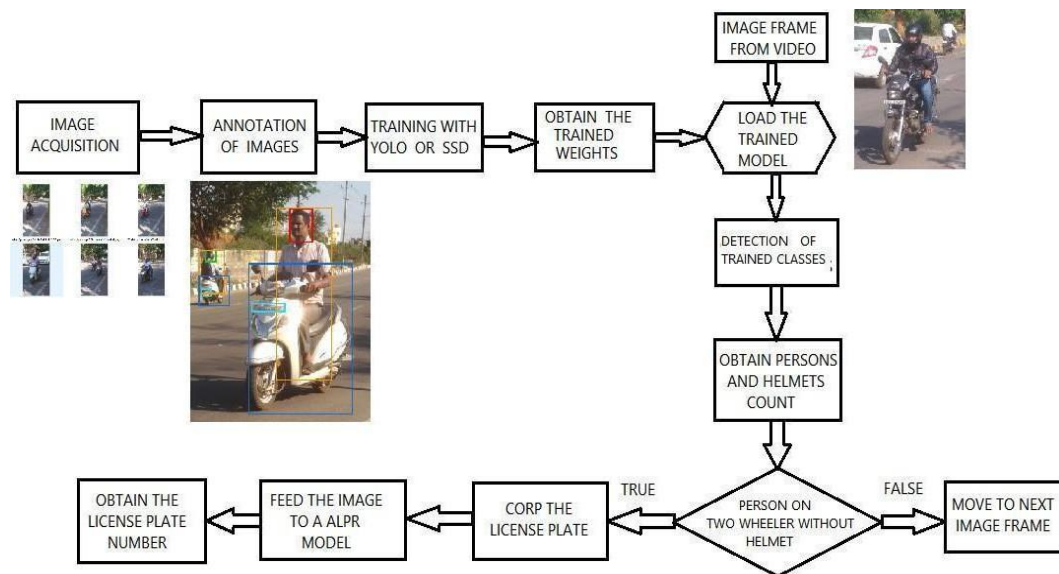


Fig 2.1 System Architecture

Working Methodology

For real-time helmet detection, there is a need for accuracy and speed. Hence a DNN based model known as YOLO version3 was chosen. YOLO means You Only Look Once. YOLO V3 is extremely

fast and accurate and is a great improvement over the previous versions of YOLO. It is based on single network evaluation unlike systems like R-CNN. Here, there are 5 objects to be detected and classified i.e., Helmet, No Helmet, Bike, Person on bike, and license plate. So for the object detection the collection of images containing each object is used for training the custom model. Once the model has been trained, it can be used to detect these five objects. We need to collect a huge data set of bike riders with and without helmets. After collecting them we need to annotate every image as required. After completion of annotation the images will be fed into the Yolo v3 model, which is used to train for custom classes. The model is loaded with the weights obtained after training. After that an image or video input is provided. In that all five trained classes will be detected by the system. We can quickly extract the rider's main class information if the person is not wearing a helmet. The number plate can be pulled easily. Once the related person class is detected and the helmetless motorcyclist is noticed. This is accomplished by determining whether the no helmet class's coordinates are contained within the person class or not. Likewise, the methods for detecting the related motorcycle and license plate are the same. Here the Optical Character Recognition algorithm receives the trimmed image of the number plate as the input. The OCR identifies content in the given input and generates the output in machine-encoded text form of strings. The OCR module would generate a list of anticipated license plate numbers as well as a level of confidence. The confidence level expresses how confident the system is in detecting the given license plate. The license plate with the strongest confidence level is then saved as a textual file for future reference.

Background Modeling

The first step of the methodology is background modeling and moving object detection. Here in this background modeling the moving objects in the surveillance video is detected and other unwanted objects will not be considered.

Classification of Motorcyclists

To find bounding boxes of different objects, we used Gaussian background subtraction which uses a method to model each background pixel. The probable background colors are the ones which stay longer and are more static. On these varying pixels, we draw a rectangular bounding box. After obtaining all the objects of motorcyclists and non-motorcyclists, a CNN model is built using these images to separate the motorcyclists from other moving objects and able to distinguish between a motorcyclist and other objects.

Recognition of Motorcyclists without helmet

Here in this step, from all images of motorcycles we will crop the top 1/4th region of the biker as that is the portion where the biker wears a helmet. Then we check whether the helmet is there or not using the object detection and trained model. Then we separate the helmet motorcyclists and non-helmet motorcyclists.

Extraction of Number Plate

After the classification of helmet and non-helmet, number from the number plate gets extracted if the motorcyclist doesn't wear a helmet through a license plate recognizer API which performs accurately. Then using the reference of license number the rule violator gets fine.

CHAPTER 3 IMPLEMENTATION

IMPLEMENTATION

To implement helmet detection in real-life we should have precision and speed. Hence in-order to full-fill the requirements the DNN based model known as the YOLO version3 was selected. YOLO is defined as You Only Look Once. The V3 version of YOLO is very rapid and specific. The new version is highly advanced when compared to the previous versions of YOLO. It is generally based on single network evaluation. It is not like the other systems named R-CNN. Here, there are 5 objects to be detected and classified i.e., Helmet, Bike, No Helmet, Person on the bike, and then the license plate. So for object detection, we need to collect a lot of images containing the required object and then we can use them for training the custom model. Once the system is trained successfully, it is easily able to detect those five objects. Collection of five required objects that are bike, helmet, no helmet, biker, and license plate form different custom datasets and used for training.

This method uses a single neural network that is developed from start to finish, snapping a picture as input and precisely calculating bounding boxes and complexity labels within every bounding box. This process has a lower predictive performance. ORB is an integration of the rapid key point detector and the easy descriptor, with additional developments. It starts by looking for key points,

then uses Harris corner to estimate and to figure the top N points among them. It also uses a pyramid to produce multi scale-features and it draws the bounding boxes around the identified object.

Dataset

Collection of five required objects that are bike, helmet, no helmet, biker, and license plate forms different custom datasets and used for training.

YOLO

The approach involves a single neural network trained end to end that takes a photograph as input and predicts bounding boxes and class labels for each bounding box directly. The technique offers lower predictive accuracy (e.g. more localization errors), although operates at 45 frames per second and up to 155 frames per second for a speed-optimized version of the model. Conceptually, YOLO divides the input image into an $S \times S$ grid, and if the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Then, each grid cell predicts B bounding boxes. Each bounding box has associated 5 predictions plus C conditional class probabilities:

Object detection is a computer vision task that involves both localizing one or more objects within an image and classifying each object in the image.

The R-CNN family of techniques we saw in Part 1 primarily use regions to localize the objects within the image. The network does not look at the entire image, only at the parts of the images which have a higher chance of containing an object.

The YOLO framework (You Only Look Once) on the other hand, deals with object detection in a different way. It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes.

The biggest advantage of using YOLO is its superb speed – it's incredibly fast and can process 45 frames per second. YOLO also understands generalized object representation.

This is one of the best algorithms for object detection and has shown a comparatively similar performance to the R-CNN algorithms. In the upcoming sections, we will learn about different techniques used in YOLO algorithm. The following explanations are inspired by [Andrew NG's course on Object Detection](#) which helped me a lot in understanding the working of YOLO.

It is a challenging computer vision task that requires successful object localization in order to locate and to draw a bounding box around each object in an image, and object classification to predict the correct class of object that was localized.

The “*You Only Look Once*,” or YOLO, family of models are a series of end-to-end deep learning models designed for fast object detection, developed by [Joseph Redmon](#), et al. and first described in the 2015 paper titled “[You Only Look Once: Unified, Real-Time Object Detection](#).”

The approach involves a single deep convolutional neural network (originally a version of GoogLeNet, later updated and called DarkNet based on VGG) that splits the input into a grid of cells and each cell directly predicts a bounding box and object classification. The result is a large number of candidate bounding boxes that are consolidated into a final prediction by a post-processing step. Now that we have grasp on why YOLO is such a useful framework, let's jump into how it actually works. In this section, I have mentioned the steps followed by YOLO for detecting objects in a given image.

Pretty straightforward, isn't it? Let's break down each step to get a more granular understanding of what we just learned. Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects (if any are found, of course).

We need to pass the labelled data to the model in order to train it. Suppose we have divided the image into a grid of size 3×3 and there are a total of 3 classes which we want the objects to be classified into. Let's say the classes are Pedestrian, Car, and Motorcycle respectively. I feel incredibly lucky to be part of our machine learning community where even the top tech behemoths embrace open source technology. Of course it's important to understand and grasp concepts before implementing them, but it's always helpful when the groundwork has been laid for you by top industry data scientists and researchers.

So, for each grid cell, the label y will be an eight dimensional vector.

- x, y : Centered relative to the grid cell ($x \in [0..1]$)
- w, h : Width and height
- IOU: Intersection over union
- OBJconf: Confidence score (Objectness)
- $C_1, C_2, \dots, C_n$: Conditional class probabilities

The confidence score for each box can be defined as:

$$\text{OBJconf} = \text{Pr}(\text{Object}) * \text{IOU}_{\text{truth, prediction}}$$

And the conditional class probability as:

$$C_i = \text{Pr}(\text{Class}_i | \text{Object})$$

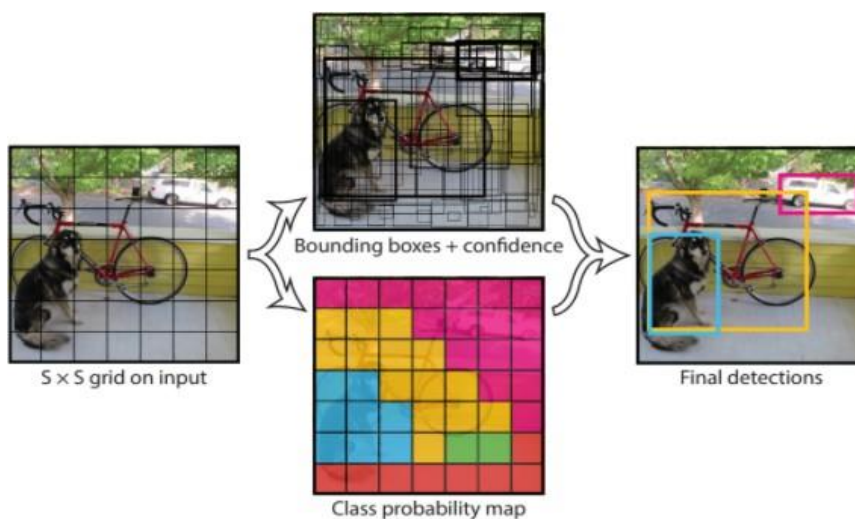


Fig 3.1.2 Training Process

HOW TO ENCODE BOUNDING BOXES

As I mentioned earlier, b_x , b_y , b_h , and b_w are calculated relative to the grid cell we are dealing with. Let's understand this concept with an example. Consider the center-right grid which contains a car:

So, b_x , b_y , b_h , and b_w will be calculated relative to this grid only. The y label for this grid will be $p_c = 1$ since there is an object in this grid and since it is a car, $c_2 = 1$. Now, let's see how to decide b_x , b_y , b_h , and b_w . In YOLO, the coordinates assigned to all the grids are b_x , b_y are the x and y coordinates of the midpoint of the object with respect to this grid.

In this case, it will be (around) $b_x = 0.4$ and $b_y = 0.3$ b_h is the ratio of the height of the bounding box (red box in the above example) to the height of the corresponding grid cell, which in our case is around 0.9. So, $b_h = 0.9$. b_w is the ratio of the width of the bounding box to the width of the grid cell. So, $b_w = 0.5$ (approximately). The y label for this grid will be

Notice here that b_x and b_y will always range between 0 and 1 as the midpoint will always lie within the grid. Whereas b_h and b_w can be more than 1 in case the dimensions of the bounding box are more than the dimension of the grid.

In the next section, we will look at more ideas that can potentially help us in making this algorithm's performance even better.

Here's some food for thought – how can we decide whether the predicted bounding box is giving us a good outcome (or a bad one)? This is where Intersection over Union comes into the picture. It calculates the intersection over union of the actual bounding box and the predicted bounding box. Consider the actual and predicted bounding boxes for a car as shown below:

Here, the red box is the actual bounding box and the blue box is the predicted one. How can we decide whether it is a good prediction or not? IoU, or Intersection over Union, will calculate the area of the intersection over union of these two boxes. That area will be:

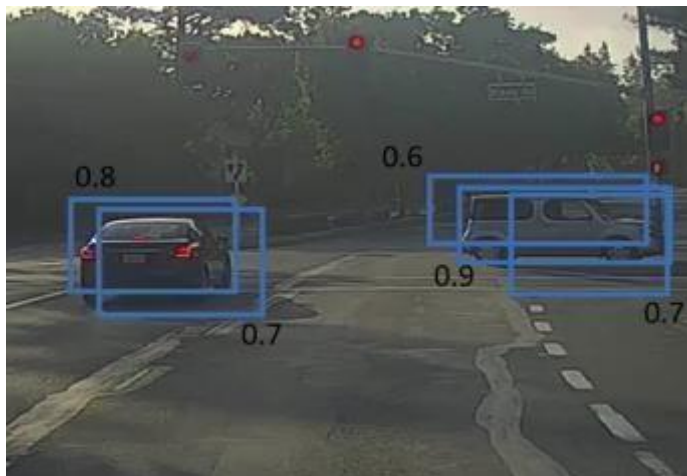
$$\text{IoU} = \text{Area of the intersection} / \text{Area of the union, i.e. IoU}$$

$$= \text{Area of yellow box} / \text{Area of green box}$$

If IoU is greater than 0.5, we can say that the prediction is good enough. 0.5 is an arbitrary threshold we have taken here, but it can be changed according to your specific problem. Intuitively, the more you increase the threshold, the better the predictions become.

There is one more technique that can improve the output of YOLO significantly – Non-Max Suppression.

One of the most common problems with object detection algorithms is that rather than detecting an object just once, they might detect it multiple times. Consider the below image:



Yolo Final Detection

Here, the cars are identified more than once. The Non-Max Suppression technique cleans up this up so that we get only a single detection per object. Let's see how this approach works.

1. It first looks at the probabilities associated with each detection and takes the largest one. In the above image, 0.9 is the highest probability, so the box with 0.9 probability will be selected first:
2. Now, it looks at all the other boxes in the image. The boxes which have high IoU with the

current box are suppressed. So, the boxes with 0.6 and 0.7 probabilities will be suppressed in our example:

3. It first looks at the probabilities associated with each detection and takes the largest one. In the above image, 0.9 is the highest probability, so the box with 0.9 probability will be selected first:

4. Now, it looks at all the other boxes in the image. The boxes which have high IoU with the current box are suppressed. So, the boxes with 0.6 and 0.7 probabilities will be suppressed in our example:

5. After the boxes have been suppressed, it selects the next box from all the boxes with the highest probability, which is 0.8 in our case:

This is what Non-Max Suppression is all about. We are taking the boxes with maximum probability and suppressing the close-by boxes with non-max probabilities. Let's quickly summarize the points which we've seen in this section about the Non-Max suppression algorithm:

1. Discard all the boxes having probabilities less than or equal to a predefined threshold (say, 0.5)
2. For the remaining boxes:
3. Repeat step 2 until all the boxes are either taken as the output prediction or discarded

TESTING

The new image will be divided into the same number of grids which we have chosen during the training period. For each grid, the model will predict an output of shape 3 X 3 X 16 (assuming this is the shape of the target during training time). The 16 values in this prediction will be in the same format as that of the training label. The first 8 values will correspond to anchor box 1, where the first value will be the probability of an object in that grid. Values 2-5 will be the bounding box coordinates for that object, and the last three values will tell us which class the object belongs to. The next 8 values will be for anchor box 2 and in the same format, i.e., first the probability, then the bounding box coordinates, and finally the classes.

Finally, the Non-Max Suppression technique will be applied on the predicted boxes to obtain a single prediction per object.

That brings us to the end of the theoretical aspect of understanding how the YOLO algorithm works, starting from training the model and then generating prediction boxes for the objects. Below are the exact dimensions and steps that the YOLO algorithm follows:

- Takes an input image of shape (608, 608, 3)
- Passes this image to a convolutional neural network (CNN), which returns a (19, 19, 5, 85) dimensional output
- The last two dimensions of the above output are flattened to get an output volume of (19, 19, 425):
 - Here, each cell of a 19 X 19 grid returns 425 numbers
 - $425 = 5 * 85$, where 5 is the number of anchor boxes per grid
 - $85 = 5 + 80$, where 5 is (pc, bx, by, bh, bw) and 80 is the number of classes we want to detect
- Finally, we do the IoU and Non-Max Suppression to avoid selecting overlapping boxes

IMPLEMENTATION OF YOLO IN PYTHON

Time to fire up our Jupyter notebooks (or your preferred IDE) and finally implement our learning in the form of code! This is what we have been building up to so far, so let's get the ball rolling.

The code we'll see in this section for implementing YOLO has been taken from [Andrew NG's GitHub repository](#) on Deep Learning. You will also need to download the [pretrained weights](#) required to run this code.

This is especially true for deep learning domains like computer vision. Not everyone has the computational resources to build a DL model from scratch. That's where predefined frameworks and pretrained models come in handy. And in this article, we will look at one such framework for object detection – YOLO. It's a supremely fast and accurate framework, as we'll see soon.

Let's first define the functions that will help us choose the boxes above a certain threshold, find the IoU, and apply Non-Max Suppression on them. Before everything else however, we'll first import the required libraries:

```
import os
```

```
import matplotlib.pyplot as plt
```



```
from matplotlib.pyplot import imshow
```

```
import scipy.io
```

```
import scipy.misc
```

```
import numpy as np
```

```
import pandas as pd
```

```
import PIL
```

```
import tensorflow as tf
```

```
from skimage.transform import resize
```

```
from keras import backend as K
```

```
from keras.layers import Input, Lambda, Conv2D
```

```
from keras.models import load_model, Model
```

```
from yolo_utils import read_classes, read_anchors, generate_colors, preprocess_image, draw  
_boxes, scale_boxes
```

```
from yad2k.models.keras_yolo import yolo_head, yolo_boxes_to_corners, preprocess_true_boxes,  
yolo_loss, yolo_body
```

```
%matplotlib inline
```

Now, let's create a function for filtering the boxes based on their probabilities and threshold

```
def yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = .6):  
  
    box_scores = box_confidence*box_class_probs  
  
    box_classes = K.argmax(box_scores,-1)  
  
    box_class_scores = K.max(box_scores,-1)  
  
    filtering_mask = box_class_scores>threshold  
  
    scores =  
  
    tf.boolean_mask(box_class_scores,filtering_mask) boxes  
  
    = tf.boolean_mask(boxes,filtering_mask)  
  
    classes = tf.boolean_mask(box_classes,filtering_mask)  
  
  
    return scores, boxes, classes
```

Next, we will define a function to calculate the IoU between two boxes:

```
def iou(box1, box2):  
  
    xi1 = max(box1[0],box2[0])  
  
    yi1 = max(box1[1],box2[1])
```

```
    xi2 = min(box1[2],box2[2])
```

```
    yi2 = min(box1[3],box2[3])
```

Let's define a function for Non-Max Suppression:

```
def yolo_non_max_suppression(scores, boxes, classes, max_boxes = 10,
iou_threshold = 0.5)
:

max_boxes_tensor = K.variable(max_boxes, dtype='int32')

K.get_session().run(tf.variables_initializer([max_boxes_tensor]))

nms_indices =

tf.image.non_max_suppression(boxes,scores,max_boxes,iou_threshold) scores =

K.gather(scores,nms_indices)

return scores, boxes, classes
```

boxes =

We now have the functions that will calculate the IoU and perform Non-Max Suppression. We get the output from the CNN of shape (19,19,5,85). So, we will create a random volume of shape (19,19,5,85) and then predict the bounding boxes:

```
yolo_outputs = (tf.random_normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1),
tf.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1),
tf.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1),
tf.random_normal([19, 19, 5, 80], mean=1, stddev=4, seed = 1))
```

Finally, we will define a function which will take the outputs of a CNN as input and return the suppressed boxes:

```
def yolo_eval(yolo_outputs, image_shape = (720., 1280.), max_boxes=10,
score_threshold=.6
, iou_threshold=.5):

box_confidence, box_xy, box_wh, box_class_probs = yolo_outputs

boxes = yolo_boxes_to_corners(box_xy, box_wh)

scores, boxes, classes = yolo_filter_boxes(box_confidence, boxes, box_class_probs,
thresh old = score_threshold)

boxes = scale_boxes(boxes, image_shape)

scores, boxes, classes = yolo_non_max_suppression(scores, boxes, classes, max_boxes,
iou_threshold)

return scores, boxes, classes
```

Let's see how we can use the *yolo_eval* function to make predictions for a random volume which we created above:

```
scores, boxes, classes = yolo_eval(yolo_outputs)
```

How does the outlook look?

```

with tf.Session() as test_b:

    print("scores[2] = " + str(scores[2].eval()))

    print("boxes[2] = " + str(boxes[2].eval()))

    print("classes[2] = " + str(classes[2].eval()))

```

```

scores[2] = 138.79124
boxes[2] = [1292.3297  -278.52167  3876.9893  -835.56494]
classes[2] = 54

```

‘scores’ represents how likely the object will be present in the volume. ‘boxes’ returns the (x1, y1, x2, y2) coordinates for the detected objects. ‘classes’ is the class of the identified object.

Now, let’s use a pretrained YOLO algorithm on new images and see how it works:

```

yolo_model = load_model("model_data/yolo.h5")

read_classes("model_data/coco_classes.txt") anchors
= read_anchors("model_data/yolo_anchors.txt")

```

After loading the classes and the pretrained model, let’s use the functions defined above to get the *yolo_outputs*.

```
yolo_outputs = yolo_head(yolo_model.output, anchors, len(class_names))
```

Now, we will define a function to predict the bounding boxes and save the images with these bounding boxes included:

```
def predict(sess, image_file):  
  
    image, image_data = preprocess_image("images/" + image_file, model_image_size = (608,  
608))  
  
    out_scores, out_boxes, out_classes = sess.run([scores, boxes, classes], feed_dict={yolo_m  
odel.input: image_data, K.learning_phase(): 0})  
  
    print('Found {} boxes for {}'.format(len(out_boxes), image_file))  
  
    # Generate colors for drawing bounding  
  
    boxes.colors =  
  
    generate_colors(class_names)
```

```
# Draw bounding boxes on the image file
```

```
draw_boxes(image, out_scores, out_boxes, out_classes, class_names, colors)
```

```
# Save the predicted bounding box on the image
```

```
image.save(os.path.join("out", image_file),
```

```
quality=90)
```

```
# Display the results in the notebook
```

```
output_image = scipy.misc.imread(os.path.join("out", image_file))
```

```
plt.figure(figsize=(12,
```

```
12))
```

```
imshow(output_imag
```

```
e)
```

Next, we will read an image and make predictions using the *predict*

function: Finally, let's plot the predictions:

```
out_scores, out_boxes, out_classes = predict(sess, "img.jpg")
```

Found 10 boxes for img.jpg
person 0.69 (1027, 270) (1091, 343)
car 0.70 (587, 352) (817, 539)
car 0.74 (390, 82) (509, 194)
car 0.74 (507, 193) (653, 327)
car 0.75 (259, 455) (490, 621)
car 0.77 (243, 265) (431, 432)
truck 0.78 (634, 30) (915, 269)
car 0.80 (139, 58) (255, 160)
car 0.83 (216, 146) (369, 267)
car 0.83 (771, 216) (945, 351)

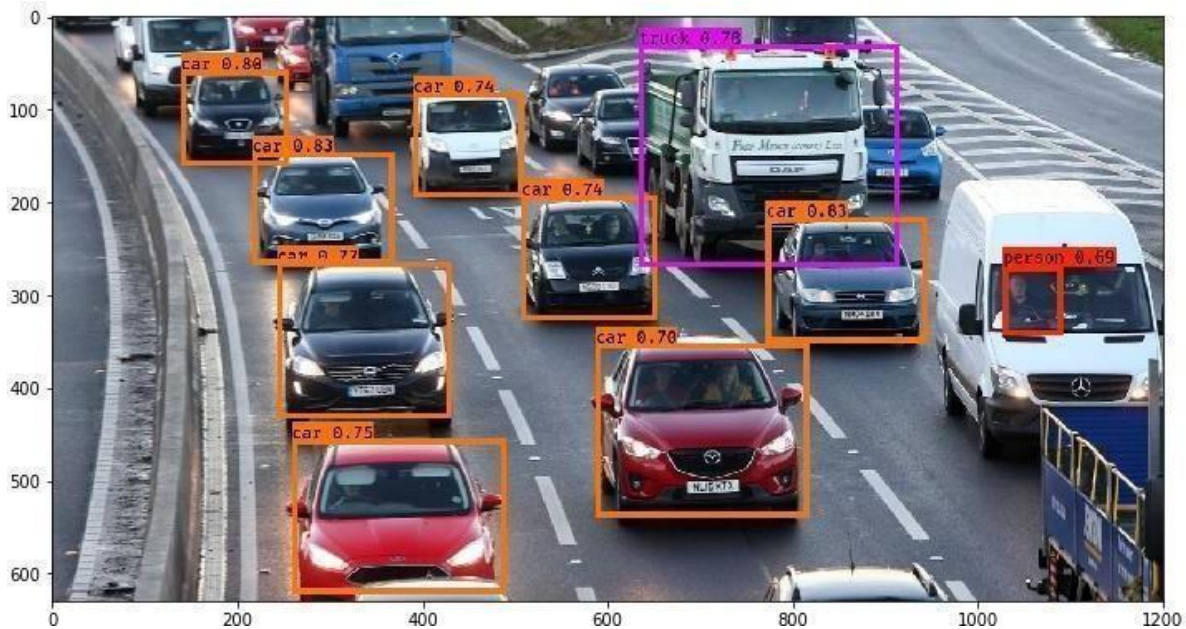


Fig 3.1.5 Detection of cars

CHAPTER 4

RESULT

4.1 RESULT

Given in the below figure we can observe that the license number from the number plate got extracted when the bike rider is not wearing a helmet by this way the government is able to catch the violators and fine them, making them not to repeat this type of violations without any manual support. This project saves a lot of manual effort and time. As the above lines speaks about the working and the methodology of the project. We can also plan to implement it so that it can help the people and the society. From the below figure we can observe the whole process of detecting the license number from the number plate that is extracted. When the bike rider is not wearing a helmet this way we can reduce road accidents. The road safety authority can get a huge advantage by implementing this process in their current system. It can help to catch the violators and fine them, making them not repeat this type of violation.

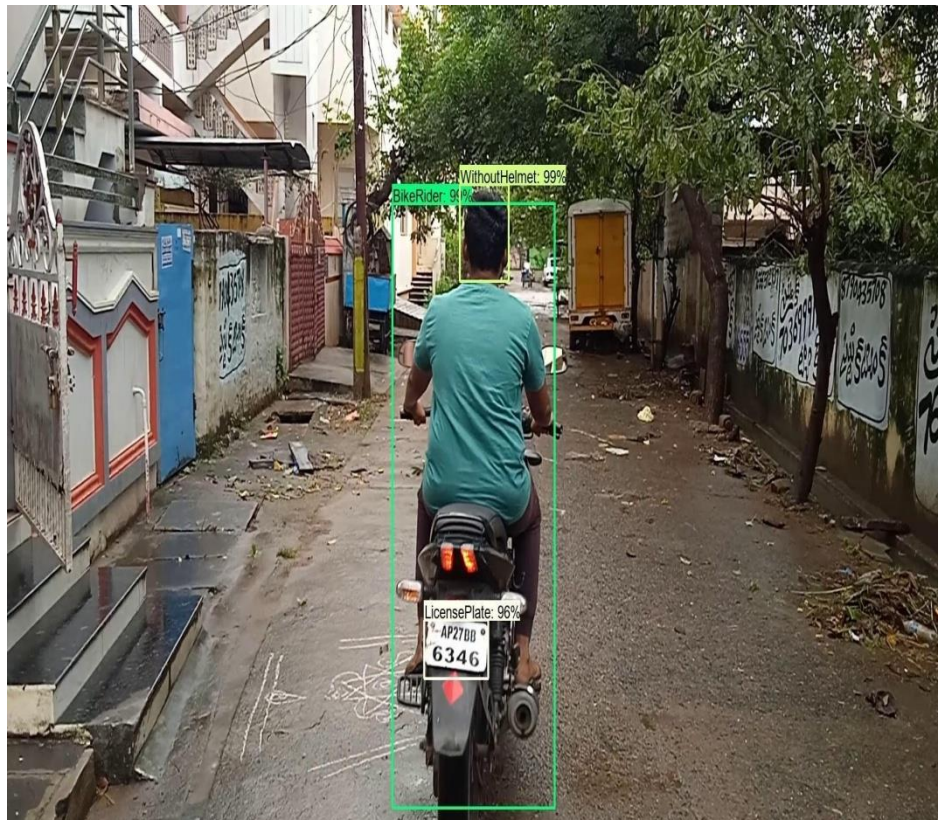


Fig 4.1 Final Output in JPG Format

CHAPTER 5

CONCLUSION AND FUTURE WORK

CONCLUSION

Thus the project speaks about the whole process of utilizing the above developed system in the day to day life. It concludes the advantages of using machine learning in order to detect the number plate of a motor-bike rider who is not wearing a helmet. This platform is comparatively less costly due to the use of open and free tools such as tensor flow, OpenCV, and YoloV3.

A lot can be done in this area. There is a large scope that we could be ventured and new designs or systems could be made to improve the conditions and efficiency of the detection and by using AI. There are so many advantages of using this technology in our system. We can reduce the count of road accidents that are happening due to not wearing the helmet. The future work of the project mainly focuses on implementing it in daily traffic. Where we can find a large number of bike riders. For those who don't wear the helmet when riding. It will be a big advantage to the road traffic authority to identify and punish them.

Future Work

The future work of the project mainly focuses on implementing it in daily traffic. Where we can find a large number of bike riders. For those who don't wear the helmet when riding. It will be a big advantage to the road traffic authority to identify and punish them.

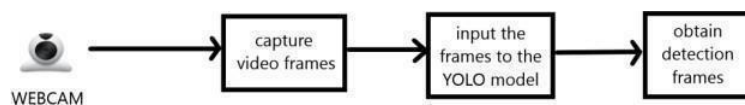


Fig 5.2 Future Implementation

REFERENCES

- [1] Pathasu Doungmala, Katanyoo Klubsuwan, "Half and Full Helmet Wearing Detection in Thailand using Haar Like Feature and Circle Hough Transform on Image Processing", IEEE International Conference on Computer and Information Technology.IEEE, 2016.
- [2] Jie Li, Huanming Liu, Tianzheng Wang ,Min Jiang and Kang Li, "Safety Helmet Wearing Detection Based on Image Processing and Machine Learning", Ninth International Conference on Advanced Computational Intelligence(ICACI), pp.109-119, 2017.
- [3] Romuere Silva, Kelson Aires, Thiago Santos, Kalyf Abdala, Rodrigo Veras, Andr e Soares, "Automatic Detection Of Motorcyclists without Helmet", 2013 XXXIX Latin America Computing Conference (CLEI).IEEE,2013.
- [4] Romuere Silva, "Helmet Detection on Motorcyclists Using Image Descriptors and Classifiers", 27th SIBGRAPI Conference on Graphics, Patterns and Images.IEEE, 2014.
- [5] Shoeb Ahmed Shabbeer, Merin Meleet(2017), "Smart Helmet for Accident Detection and Notification", 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), 2017.
- [6] Kang Li, Xiaoguang Zhao, Jiang Bian, Min Tan, "Automatic Safety Helmet Wearing Detection", 2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER).IEEE, 2017.
- [7] Kunal Dahiya, Dinesh Singh, C. Krishna Mohan, "Automatic Detection of Bike-riders without Helmet using Surveillance Videos in Real-time", International joint conference on neural network(IJCNN). IEEE, 2016.
- [8] XINHUA JIANG "A Study of Low-resolution Safety Helmet Image Recognition Combining Statistical Features with Artificial Neural.
- [9] C.Y.Wen, S.H.Chiu, J.J.Liaw, and C.P. Lu, "*The safety helmet* detection for ATM's surveillance system via the modified Hough *transform*" Proceedings of IEEE 37th Annual International Carnahan Conference on Security Technology, pp. - 364-369, 2003.

- [10] J. Chiverton, *"Helmet presence classification with motorcycle detection and Tracking"*, IET Intelligence Transport Systems (ITS), Volume 6, Issue no. 3, pp. 259-269, 2012.
- [11] "Vimal Kumar, S., Vasudevan, S., Mary Posonia, A." *Urban mode of dispatching students from hostel". ARPN Journal of Engineering and Applied Sciences 12(13), pp. 4089-4092.*
- [12] Vigneshwari, S., and M. Aramudhan. "Web information extraction on multiple ontologies based on concept relationships upon training the user profiles." In *Artificial Intelligence and Evolutionary Algorithms in Engineering Systems*, pp. 1-8. Springer, New Delhi, 2015.
- [13] B. Bharathi; P. Bindhu Shamily, *A review on iris recognition system for person identification, International Journal of Computational Biology and Drug Design, 2020 Vol.13 No.3, pp.316 – 331.*
- [14] Mary, S.P., Lakshmi, S.V., Anuhya, S.," *Color detection and sorting using internet of things machine"*, *Journal of Computational and Theoretical Nanoscience 16(8), pp. 3276- 3280, 2019.*
- [15] Usha Nandini, D., Sathyabama Krishna, R., Nithya, M. and Pavithra, R., 2019. *A resourceful information collecting system using smart black box. Journal of Computational and Theoretical Nanoscience, 16(8), pp.3346-3350.*

APPENDIX

A. SOURCE CODE

```
import os
import cv2
import numpy as np
import tensorflow as tf
import sys
from imutils.video import FPS
import imutils
import time
from decimal import Decimal, ROUND_HALF_UP

sys.path.append(r"D:/helmet detection")
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

MODEL_NAME = 'inference_graph'
VIDEO_NAME = 'accurate.mp4'
CWD_PATH = os.getcwd()
PATH_TO_CKPT = os.path.join(CWD_PATH, MODEL_NAME,
'frozen_inference_graph.pb')
PATH_TO_LABELS = os.path.join(CWD_PATH, 'labelmap.pbtxt')
PATH_TO_VIDEO = os.path.join(CWD_PATH, VIDEO_NAME)
NUM_CLASSES = 4
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES,
```

```

        use_display_name=True)
category_index = label_map_util.create_category_index(categories)
detection_graph = tf.compat.v1.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.compat.v2.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')
    sess = tf.compat.v1.Session(graph=detection_graph)
fps = FPS().start()
video = cv2.VideoCapture(PATH_TO_VIDEO)
import json
from collections import OrderedDict
from glob import glob
import cv2
import requests

def platemain():
    regions = ['in']
    result = []
    path = 'frame.jpg'
    with open(path, 'rb') as fp:
        response = requests.post(
            'https://api.platerecognizer.com/v1/plate-reader/',
            files=dict(upload=fp),
            data=dict(regions=regions),
            headers={'Authorization': 'Token ' +
'fe801a314498e5fd43a6069099e65b7bc5ff9c3d'})
        print("respond.status_code", response.status_code)
        result.append(response.json(object_pairs_hook=OrderedDict))

```

```

print(result)
time.sleep(1)
im = cv2.imread(path)
resp_dict = json.loads(json.dumps(result, indent=2))
if resp_dict[0]['results']:
    num = resp_dict[0]['results'][0]['plate']
    boxs = resp_dict[0]['results'][0]['box']
    xmins, ymins, ymaxs, xmaxs = boxs['xmin'], boxs['ymin'], boxs['ymax'], boxs['xmax']
    # cv2.imshow("image",im)
    # cv2.waitKey(0)
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    # cv2.imshow("Gray Image",img)
    # cv2.waitKey(0)
    edges = cv2.Canny(img, 100, 200)
    # cv2.imshow("Edge Image",edges)
    # cv2.waitKey(0)
    cv2.rectangle(im, (xmins, ymins), (xmaxs, ymaxs), (255, 0, 0), 2)
    cv2.rectangle(edges, (xmins, ymins), (xmaxs, ymaxs), (255, 0, 0), 2)
    # cv2.imshow("Box Edges",edges)
    # cv2.waitKey(0)
    # cv2.imshow("Box On Original",im)
    # cv2.waitKey(0)
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(im, num, (xmins, ymins - 10), font, 1, (255, 0, 0), 2, cv2.LINE_AA)
    # cv2.imshow("Number",im)
    # cv2.waitKey(0)
    cv2.destroyAllWindows()
    print("the bike number is {}".format(str(num).upper()))
    return str(num).upper()

```

```

def resize(w, h, w_box, h_box, pil_image):

```

```

f1 = 1.0 * w_box / w # 1.0 forces float division in Python2
f2 = 1.0 * h_box / h
factor = min([f1, f2])
width = int(w * factor)
height = int(h * factor)
return pil_image.resize((width, height))

```

```

j = 0
s = []
with detection_graph.as_default():
    with tf.compat.v1.Session(graph=detection_graph) as sess:
        nmp = []
        st = []
        while True:
            start_time = time.time()
            j += 1
            ret, image_np = video.read()
            if ret == True:
                image_np_expanded = np.expand_dims(image_np, axis=0)
                image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
                boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
                scores = detection_graph.get_tensor_by_name('detection_scores:0')
                classes = detection_graph.get_tensor_by_name('detection_classes:0')
                num_detections = detection_graph.get_tensor_by_name('num_detections:0')
                (boxes, scores, classes, num_detections) = sess.run(
                    [boxes, scores, classes, num_detections],
                    feed_dict={image_tensor: image_np_expanded})
                # scores=(scores[0]*10).astype(int)
                cl = (classes[0].astype(int))[:7]
                sc = (scores[0][:7]) * 100
                sc = list(map(int, sc))

```

```

# print(cl,sc)
d = {1: 0, 2: 0, 3: 0, 4: 0}
for i in range(len(cl)):
    if (sc[i] > d[cl[i]]):
        d[cl[i]] = sc[i]
# Visualization of the results of a detection.
vis_util.visualize_boxes_and_labels_on_image_array(
    image_np,
    np.squeeze(bboxes),
    np.squeeze(classes).astype(np.int32),
    np.squeeze(scores),
    category_index,
    use_normalized_coordinates=True,
    line_thickness=4,
    min_score_thresh=0.85)
data = {}
if (d[3] > 90):
    if (d[4] < 90):
        print("Without Helmet: No Number plate")
if (d[4] > 90):
    name = 'frame.jpg'
    print('Creating...' + name)
    cv2.imwrite(name, image_np)
    number = platemain()
    # if (d[4] > 80):
    # name = 'frame.jpg'
    # print('Creating...' + name)
    # cv2.imwrite(name, image_np)
    # number = platemain()
    if len(number) not in [5, 6, 7]:
        continue
    print("pass to number plate", number)

```

```

    if (d[3] > 90):
        print("Without Helmet Number plate ")
        if number not in nmp:
            nmp.append(number)
            st.append('Without Helmet')
    elif (d[2] > 90):
        print("With Helmet Number plate")
        if number not in nmp:
            nmp.append(number)
            st.append('With Helmet')
    elif (d[1] < 90):
        print("Vehicle with number plate")
        if number not in nmp:
            nmp.append(number)
            st.append('Other Vehicle')

# print(nmp, st)
print('Iteration %d: %.3f sec' % (j, time.time() - start_time))
cv2.imshow('object detection', cv2.resize(image_np, (800, 600)))
# cv2.waitKey()
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

for m in range(len(nmp)):
    data = {'Status': st[m],
           'Number_Plate': nmp[m]
          }
    print(data)

m = json.dumps(data)
with open("D:\helmet detection\data1.json", "w") as f:
    f.write(m)

cv2.destroyAllWindows()

```

B.SCREEN SHOTS



Fig: INPUT FRAME

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help Helmet detection - new.py
Helmet detection - new.py
Project
  Helmet detection
    inference_graph
    training
    bike1.mp4
    frame.jpg
    new.py
  External Libraries
  Scratches and Consoles
new.py
11 from decimal import Decimal, ROUND_HALF_UP
12 sys.path.append("G:\Helmet detection/")
13 from object_detection.utils import label_map_util
14 from object_detection.utils import visualization_utils as vis_util
15 MODEL_NAME = 'inference_graph'
16 VIDEO_NAME = 'bike1.mp4'
17 CWD_PATH = os.getcwd()
18 PATH_TO_CKPT = os.path.join(CWD_PATH, MODEL_NAME, 'frozen_inference_graph.pb')
19 PATH_TO_LABELS = os.path.join(CWD_PATH, 'training', 'labelmap.pbtxt')
20 PATH_TO_VIDEO = os.path.join(CWD_PATH, VIDEO_NAME)
21 NUM_CLASSES = 4
22 label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
23 categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
24 category_index = label_map_util.create_category_index(categories)
25 detection_graph = tf.compat.v1.Graph()
26 with detection_graph.as_default():
27     od_graph_def = tf.compat.v1.GraphDef()
28     with tf.compat.v2.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
29         serialized_graph = fid.read()
30         od_graph_def.ParseFromString(serialized_graph)
31         tf.import_graph_def(od_graph_def, name='')
32     sess = tf.compat.v1.Session(graph=detection_graph)
33     fns = FPS().start()
Run
[OrderedDict([('processing_time', 68.564), ('results', [OrderedDict([('box', OrderedDict([('xmin', 860), ('ymin', 1404), ('xmax', 1059), ('ymax', 1532)])), ('plate', 'ap27bb6346')]),
the bike number is AP27BB6346
pass to number plate AP27BB6346
Without Helmet Number plate
Iteration 22: 10.397 sec
Creating...frame.jpg
Process finished with exit code -1
Python versions compatibility
Your source code contains _future_ imports
which will be removed in future releases.
Go to Settings to activate Windows
12:41 CRLF UTF-8 4 spaces Python 3.8
```

Fig: OUTPUT

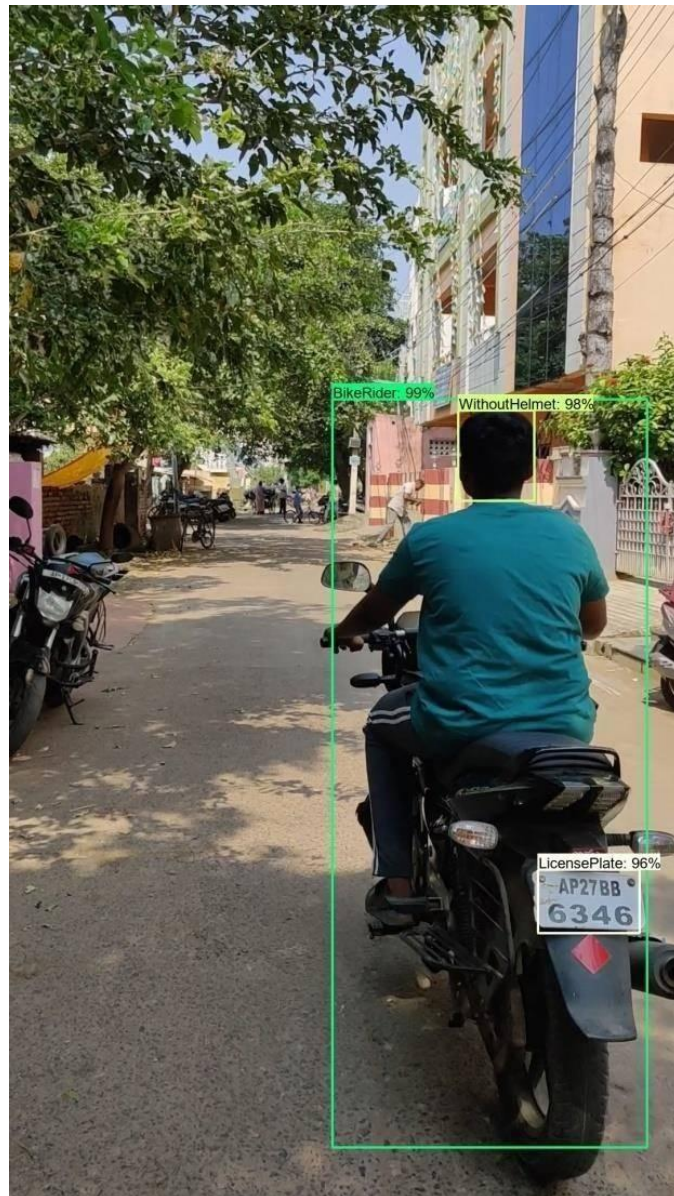


Fig: OUTPUT FRAME

