INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

Speed Estimation of a moving vehicle from video stream using Machine Learning Techniques

by

Ankan Halder (510818008)
Aneek Ghosh (510818039)
Pranjal Sharma(510818089)

Under the guidance of
Dr. Arindam Biswas

A report submitted in completion of the requirements for the major project of Bachelor of Technology in Information Technology
(8th Semester)

Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur
P.O. Botanic Garden, Howrah 711103, WB, India
May 2022

Contents

Acknowledgement

We would like to extend our sincere thanks to our project guide, **Dr. Arindam Biswas**, Professor, Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, for his guidance and constant supervision without which it would not have been possible to complete this project within the due time.

We are fortunate enough to get constant encouragement, support, and guidance from **Dr. Sukanta Das**, Associate Professor and Head of the department, Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, as well as other faculty members and staff members who helped us in successfully completing our project work.

Finally, we cordially thank our fellow classmates of Information Technology, of the Indian Institute of Engineering Science and Technology, our parents, and all others who have been attached to this phase of the development of the project directly or otherwise.

—---------- —----------

Signature of the Supervisor Signature of the H.O.D

—----------

Signature of the Student

Introduction

The aim of our project is **speed estimation** of moving vehicles from a video stream using **Machine Learning techniques**.

We have divided our project into three parts

1. Vehicle Detection

2. Vehicle Tracking

3. Speed Estimation

It aims to detect one or more than one vehicle(s) (eg:- car, truck, bike) in an image or a video stream. Vehicle Detection provides information assisting vehicle counting, vehicle speed measurement, identification of traffic accidents and traffic flow prediction.

We have used the **OpenCV library** for preprocessing the input stream and loading our model into the system. The Machine Learning model we have used in this project is **YOLOv3** which is a faster and lighter model used for object detection.

Motivation & Objective

Rash Driving and accidents are increasing day by day, so to decrease accidents, **speed of vehicles** should be controlled. Detection of speed of a vehicle is a necessity for this reason. We are detecting the speed of vehicles using machine learning techniques.

We are tracking the vehicle in the video frame and then applying speed estimation techniques to find the speed of a vehicle.

One of the applications of our project is in traffic monitoring and management systems. Nowadays the most common way to measure speed is by using the radar equipment, therefore it is very important to propose any other concepts like measuring vehicle speed from video streams. Instead of hardware dependency that is a problem with radar systems we can use image processing for **speed estimation**, which is mainly based on software implementation.

Tools and Technologies

The machine learning model that we have chosen for this project is **YOLOv3**. It is a fast and light-weight model used for object detection. The **YOLOv3** is a pre-trained model (trained on **coco** dataset). The required configuration and weights files for the YOLOv3 has been taken from the official yolo website. The model is then loaded into a deep neural network using the **OpenCV** library. The **OpenCV** library is also used to preprocess the input stream and draw the bounding boxes along with the confidence score on the stream once it has the predicted values. We have used **python** as the programming language for our project.

- **python -** Programming Language
- **OpenCV -** for Computer Vision
- **YOLOv3 -** Machine Learning model

About YOLOv3

YOLOv3 (**You Only Look Once**, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. YOLO uses features learned by a **deep convolutional neural network** to detect an object. Versions 1-3 of YOLO were created by **Joseph Redmon and Ali Farhadi.**

- ***History of YOLO***

The first version of YOLO was created in 2016, and version 3 was made two years later in 2018. **YOLOv3** is an **improved version** of YOLO and YOLOv2. YOLO is implemented using the OpenCV deep learning libraries.

- **You Only Look Once**

As typical for object detectors, the features learned by the convolutional layers are passed onto a classifier which makes the detection prediction. In YOLO, the prediction is based on a convolutional layer that uses **1×1 convolutions.**

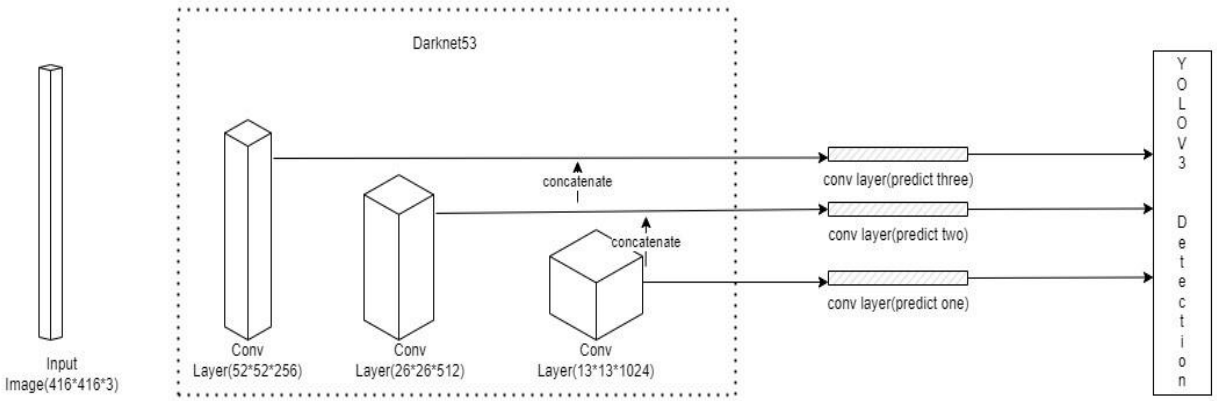
YOLO is named **"you only look once**" because its prediction uses 1×1 convolutions. The size of the prediction map is exactly the size of the feature map before it.

- ***Working principle and architecture***

YOLO is a **Convolutional Neural Network (CNN)** for performing object detection in real-time. CNNs are classifier-based systems that can process input images as structured arrays of data and identify patterns between them. YOLO has the advantage of being much **faster** than other networks and still maintains **accuracy**.

It allows the model to look at the whole image at test time, so its predictions are informed by the global context in the image. YOLO and other convolutional neural network algorithms "**score**" regions based on their similarities to predefined classes.

YOLOv3 uses a variant of **Darknet**, a framework to train neural networks, which originally has 53 layers. Another 53 layers are stacked onto it, accumulating to a total of a **106-layer** fully convolutional architecture. The layers include **Residual blocks**, **Upsampling layer**, **Detection layer**. In the convolutional layer, 1 x 1 kernel is applied to the input. The algorithm makes predictions at three different scales.



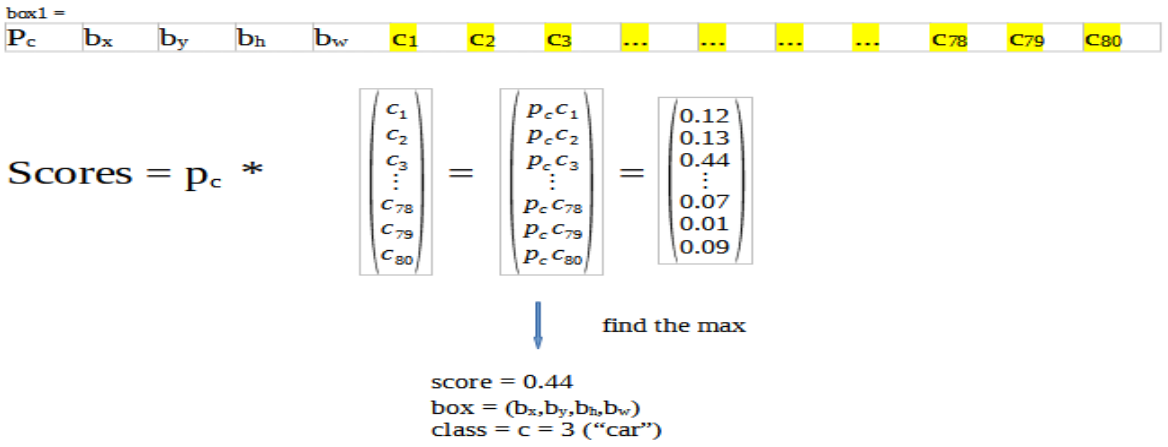
- ***Algorithm of YOLOV3***

The YOLOv3 algorithm first separates an image into a **grid**. Each grid cell predicts some number of **boundary boxes** (sometimes referred to as anchor boxes) around objects that score highly with the aforementioned predefined classes.

Each boundary box has a respective confidence score of how accurate it assumes that prediction should be and detects only one object per bounding box. The boundary boxes are generated by **clustering** the dimensions of the **ground truth boxes** from the original dataset to find the most common shapes and sizes.

Other comparable algorithms that can carry out the same objective are **R-CNN** (Region-based Convolutional Neural Networks made in 2015) and **Fast R-CNN** (R-CNN improvement developed in 2017), and **Mask R-CNN**.

However, unlike systems like R-CNN and Fast R-CNN, YOLO is trained to do classification and bounding box regression at the same time.



Box (b_x,b_y,b_h,b_w) has detected c= 3 ("car") with probability score: 0.44

The class with the highest score will be the predicted result.

Vehicle Detection

- ***Why YOLOv3?***

There are major differences between YOLOv3 and older versions in terms of speed, precision, and specificity of classes. YOLOv2 and YOLOv3 are worlds apart in terms of accuracy, speed, and architecture. YOLOv2 came out in 2016, two years before YOLO v3. YOLO v3 is **faster** and more **accurate**.

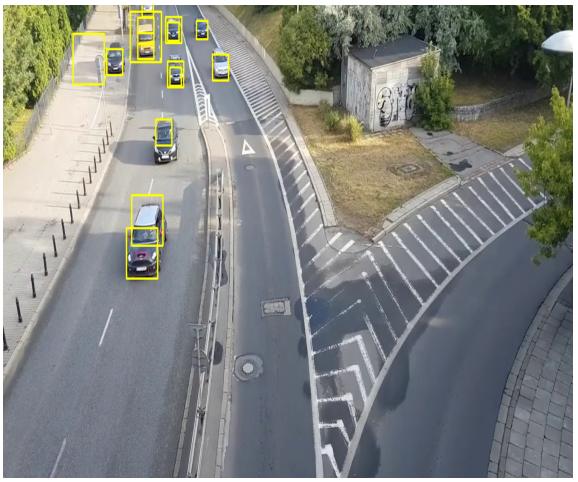
- ***Comparison with other Models***

Model	mAP-50	FPS
SSD321	45.4	16
DSSD321	46.1	12
R-FCN	51.9	12
SSD513	50.4	8
DSSD513	53.3	6
Retinanet-50	50.9	14
YOLOv3-416	55.3	35

- ***Use in our project***

The video stream is read and processed by cv2 library. We have loaded the YOLOv3 model using the **dnn library** of cv2 . We store the confidence score and coordinates of bounding boxes of each of the vehicles that is detected. Next, for each of the detection we have drawn the **bounding boxes** and **confidence scores** on the stream using cv2.

Input ***Output***



Vehicle Tracking

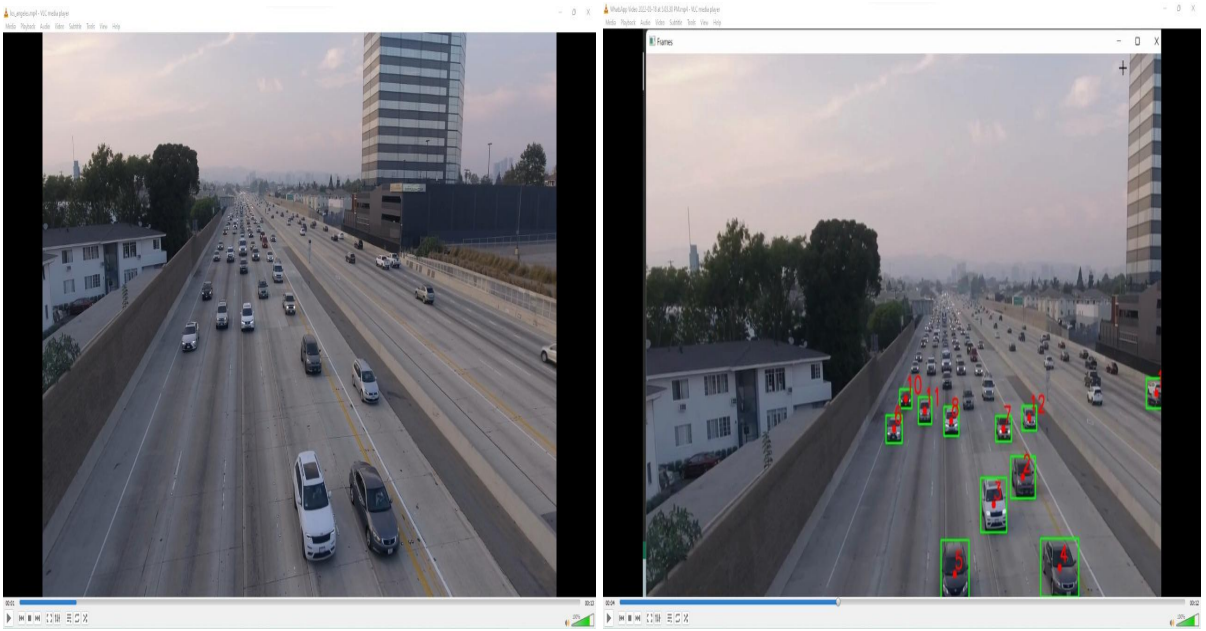
Object tracking is the process of **locating** a moving object in a video.

Computers only understand numbers. It doesn't understand what an image is but the pixel values associated with the image. Two images that appear to be exactly the same to the human eye may not be the same case for the computer, as even a slight change in a pixel will result in a difference. Because of that, object tracking is considered one of the most **complex tasks** in computer vision.

A video is basically a combination of many frames or a combination of many still images. We are tracking the **centroid** of every vehicle in each frame. We are using **openCV** library to do these operations frame by frame in the video stream.

In vehicle tracking we keep track of the centroid of each vehicle in an array. For each frame change we calculate the change in the position of the centroids of the current frame and previous frame. We assign or retain a tracking id to each vehicle if the change in position is less than our assumed **threshold** value. If the change in position is greater than our assumed threshold value then the tracking id is **destroyed**. In that case a new tracking id is assigned to the vehicle if it has not left the video stream.

Input **Output**



- ***Difficulty in Object Tracking***

Tracking algorithms are expected to track the object in a video in a fraction of a second and with high accuracy. This can be significantly tampered due to the variety of **background distractions** in many scenarios. An object can be present in an image (or a video) in various sizes and orientations. Another problem with object tracking is when **multiple objects** come so close together that they appear to be merged. This can confuse the computer into thinking the merged object is a single object or simply wrongly identifying the object.

Speed Estimation

For the speed estimation part, we have used an approximated mathematical formula.

We assumed the width of the vehicles to be 1m and from the detection output we calculated the pixels in horizontal distance. This gives us the value of **pixels per meter (ppm)**.

For each frame change we find the distance covered by the centroid of each vehicle in **pixels (d_pixels)**.

We divide the d_pixels by ppm to get the **distance in meters (d_meters)**.

Then we multiply d_meters with fps of the video which gives us the **distance covered in one second** by each centroid.

So, we have the speed of each vehicle in meters/second and multiplying it by **3.6** we get the **speed in kilometers/hour**.

Estimated speed(Km/hr) = Distance (meters) * FPS * 3.6

Validation of output

We have taken a test case where the vehicle was moving at a speed of around **15 to 20 km/hr** and our output is approximately **12 to 14 km/hr** which is quite close to the actual speed.

- ***Estimated Speed detection video***

Conclusion

This module consists of YOLOv3 as the **backbone**. It uses features learned by a deep convolutional neural network(YOLOv3) to detect vehicles. We have used openCV library for object tracking and speed estimation purposes. At last we have validated the output against a real life test case.

- **Limitations and Assumptions of the project**

1) We have used the YOLOv3 model whose **accuracy** is quite high but there are still some limitations of YOLOv3 .

2) Our assumption was that the vehicle **width** is **1 meter** but is not always true for all the vehicles.

3) We have assumed a certain **threshold value** for distance of pixels to track a vehicle.

4) If the resolution of the video is **poor**, then there is difficulty in vehicle detection.

- **Improvements and Future scope**

1) The detection of the vehicles can be improved by using **future versions** of the YOLO model or any other machine learning algorithms.

2) The tracking of the vehicles can be improved by more **efficient** tracking algorithms.

References

- https://pjreddie.com/media/files/papers/YOLOv3.pdf
- https://opencv.org
- https://github.com/opencv
- https://towardsdatascience.com/yolo-v3-explained-ff5b850390f

Thank You.