Chapter 1

INTRODUCTION

The topic “Vehicle Tally Using Open CV” is an application project which keeps a track on simple detection, count on number of vehicles and speed detection of vehicles that are being moved on the roads. Vehicle detection that is proposed has been developed using tensorflow object detection API and Open CV library tools. Tensorflow API is widely used in computer vision tasks like image processing, object detection and related computer vision tasks like face detection , face recognition and video object video co-segmentation. In this project we extensively use tensorflow’s video object co-segmentation technique.

Vehicle detection and frames has been developed using Open CV software. Open CV(Open Source Computer Vision Library) is a open source computer vision and machine learning library. It was built to provide a common infrastructure of computer vision applications and to accelerate the use of machine perception in the commercial products. Here vehicle detection and tracking has been developed using Open CV via Image Pixel Manipulation and Calculation. The entire project works on the above two mentioned application software.

Here we provide a sample video of a road with moving vehicles, the program is developed which can count(track), detect and speed can also be determined, vehicles can be of any type like car, bike, heavy load vehicles. This project can be extensively used in traffic signals and highway tolls where detection and speed of vehicles plays a major role.

The purpose of this project was to detect and count vehicles from a CCTV feed, and to ultimately give an idea of what the real-time on street situation is across the road network.

Object Detection: In order to count vehicles we first need to be able to detect them in an image. This is pretty simple for a human to pick out but harder to implement in the machine world. However, if we consider that an image is just an array of numbers (one value per pixel), we may be able to use this to determine what a vehicle looks like and what we'd expect to see when there isn't a vehicle there. We can use OpenCV to look at how the value of certain pixels changes for these two conditions. To do this, we must first translate our image from RGB channels (Red, Green Blue) to HSV (Hue, Saturation, Value) and inspect each channel to see if it can tell us something. Now that we have a

background image, or an array of default/background values, we can use OpenCV to detect when these values go above a certain value (or 'threshold value'). We assume that this occurs when there is a vehicle within that pixel, and so use OpenCV to set the pixels that meet the threshold criteria to maximum brightness (this will make detecting shapes/vehicles easier later on).

Counting Vehicles: The vehicle counter is split into two class objects, one named Vehicle which is used to define each vehicle object, and the other Vehicle Counter which determines which 'vehicles' are valid before counting them (or not). Vehicle is relatively simple and offers information about each detected object such as a tracked position in each frame, how many frames it has appeared in (and how many it has not been seen for if we temporarily loose track of it), whether we have counted the vehicle yet and what direction we believe the vehicle to be travelling in. We can also obtain the last position and the position before that in order to calculate a few values within our Vehicle Counter algorithm.

Vehicle Counter is more complex and serves several purposes. We can use it to determine the vector movement of each tracked vehicle from frame to frame, giving an indicator of what movements are true and which are false matches. We do this to make sure we're not incorrectly matching vehicles and therefore getting the most accurate count possible. In this case, we only expect vehicles travelling from the top of the image to the bottom right hand corner, or the reverse. This means we only have a certain range of allowable vector movements based on the angle that the vehicle has moved.

Expressways, highways and roads are getting overcrowded due to increase in number of vehicles. Vehicle detection, tracking, classification and counting is very important for military, civilian and government applications, such as highway monitoring, traffic planning, toll collection and traffic flow. For the traffic management, vehicles detection is the critical step. Computer Vision based techniques are more suitable because these systems do not disturb traffic while installation and they are easy to modify. In this project I present inexpensive, portable and Computer Vision based system for moving vehicle detection and counting. Image from video sequence are taken to detect moving vehicles, so that background is extracted from the images. This highway traffic counting process has been developed by background subtraction, image filtering, image binary and segmentation methods are used. This system is capable of counting moving vehicles from pre-recorded videos. Traffic counts, speed and vehicle classification are fundamental data

for a variety of transportation projects ranging from transportation planning to modern intelligent transportation systems. Still 'Traffic Monitoring' and 'Information Systems' related to classification of vehicles rely on sensors for estimating traffic parameters. Currently, magnetic loop detectors are often used to count vehicles passing over them. Vision-based video monitoring systems offer a number of advantages over earlier methods. In addition to vehicle counts, a much larger set of traffic parameters such as vehicle classifications, lane changes, parking areas etc., can be measured in such type of systems. In large metropolitan areas, there is a need for data about vehicle classes that use a particular highway or a street. A classification and counting system like the one proposed here can provide important data for a particular decision making agency. This system uses a single camera mounted on a pole or other tall structure, looking down on the traffic scene. It can be used for detecting and classifying vehicles in multiple lanes and for any direction of traffic flow.

Vehicles detection and counting are done a non-intrusives sensor/manual method/video infrared, magnetic, radar, ultrasonic, acoustic, and video imaging sensors and intrusive sensor sensors include pneumatic road tube, piezo-electric sensor, magnetic sensor, and inductive loop . Non inrusive technology has advantages over intrusive technology which requires closing of traffic lanes and put construction workers in harm's way, stop traffic or a lane closure and non-intrusive sensors are above the roadway surface and don't typically require a stop in traffic or lane closure. Both types of sensors have advantages and disadvantages . But the accuracy from video or digital counting manually is very high as compared to other technology. However the image processing is time consuming and requires some automation to save the time for image count and classification. In current era of python type programming language has much addition of image processing and time saving for vehicle detection, counting and classification. This project states about way to image process, type of filter used and proposed technique are able to detect, count and classify the image accurately.

1.1 Motivation of the project.

- Expressways, highways and roads are getting overcrowded due to increase in number of vehicles. Vehicle detection, tracking, classification and counting is very important for military, civilian and government applications, such as highway monitoring, traffic planning, toll collection and traffic flow.
- Computer Vision based techniques are more suitable because these systems do not disturb traffic while installation and they are easy to modify.
- "Vehicles Tally Using Open CV" has been developed for easy detection and tracking the speed of the vehicles, thus we can determine the traffic and any over speed moving vehicles on the roads.
- The primary motivation of this current work is to keep the vehicles moving because moving vehicles can only be detected.
- Usage of Open CV in conjunction with the tensorflow API makes easy detection of vehicles.
- This type of methods are easier and faster when compared to hardware methods of detection which requires proportional framework.

1.2 Problem Statement:

"As the number of vehicles on road are increasing day by day, it is very necessary to keep the count of the number of vehicles moving on the road."

Chapter 2

SYSTEM REQUIREMENTS

2.1 HARDWARE AND SOFTWARE USED

Hardware System Configuration:

Processor	- Intel Core i5
Speed	- 1.8 GHz
RAM	- 256 MB (min)
Hard Disk	- 10 GB

Software System Configuration:

Operating System	- Ubuntu
Programming Language	- Python
Compiler	- Python Interpretor

Why Python?

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until July 2018.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python features a comprehensive standard library, and is referred to as "batteries included".

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open-source software and has a community-based

development model. Python and CPython are managed by the non-profit Python Software Foundation.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter, map and reduce functions; list comprehensions, dictionaries, sets and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document *The Zen of Python* (*PEP 20*), which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is *not* considered a compliment in the

Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.[51]When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonists*, *Pythonistas*, and *Pythoneers*.

Why OpenCV?

OpenCV (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe.

Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of

optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

On May 2016, Intel signed an agreement to acquire Itseez a leading developer of OpenCV.

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Perl, Ch, Haskell, and Ruby have been developed to encourage adoption by a wider audience.

Since version 3.4, **OpenCV.js** is a JavaScript binding for selected subset of OpenCV functions for the web platform.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface.

OpenCV runs on the following desktop operating systems: Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD. OpenCV runs on the following mobile operating systems: Android, iOS, Maemo, BlackBerry 10. The user can get official releases from SourceForge or take the latest sources from GitHub. OpenCV uses CMake.

Chapter 3

SYSTEM DESIGN

3.1 MODULES:

- Creating a frame:

  We give the all possible coordinates of all type of vehicles so as to make sure that all the vehicles in a particular video is traced with ease.

- Accessing the input file:

  The video footage of any of the road is taken and is saved in the same folder where the program code is saved and the video file is accessed from the code.

- Recognizing the vehicles:

  The vehicles passing on either side of the road is recognized based on the frame that is created.

- Counting the vehicles:

  The number of vehicles passing on either side of the road is calculated separately for analyzing the traffic.

3.2 ARCHITECTURE:

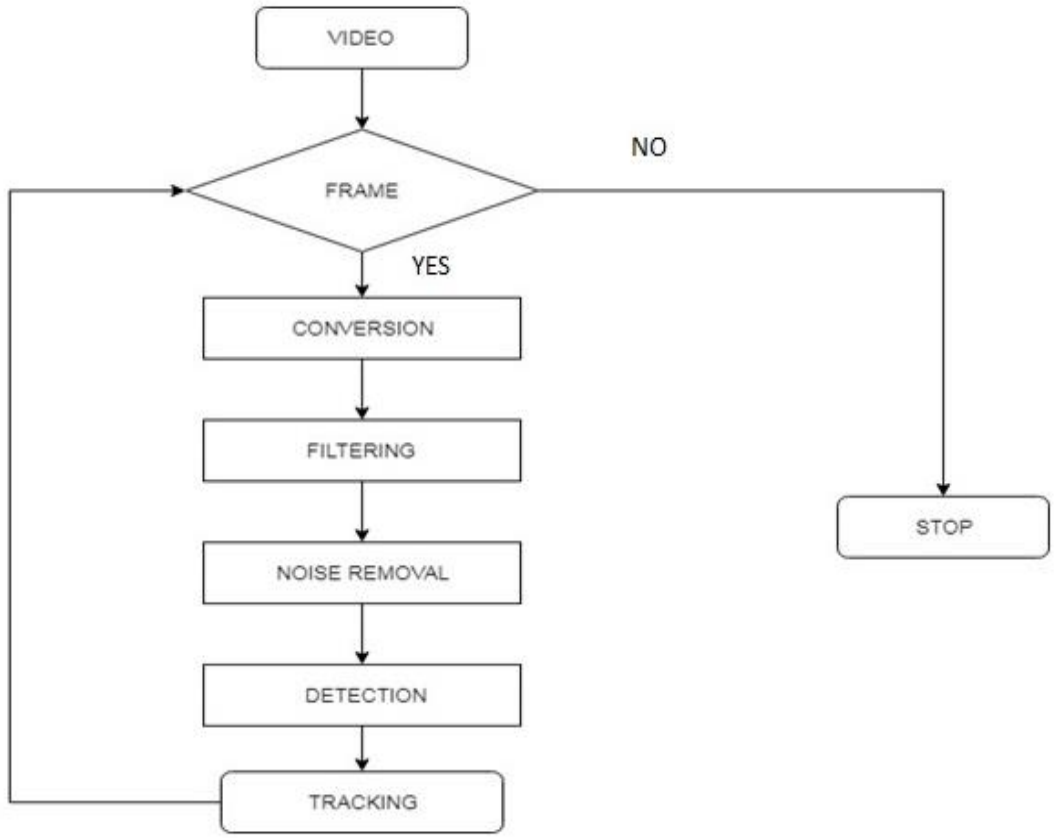

Fig3.2.1 Architecture of the proposed system

3.3 ALGORITHM:

- Step 1- Start.
- Step 2- Take the video sample as input.
- Step 3- convert the video to frames using OpenCV.
- Step 4- Filter the given frames into black and white.
- Step 5- Perform noise cancelling using OpenCV.
- Step 6-Identify the object(i.e, determining the object as a car ,bike or others).
- Step 7- After detecting the object, count for each vehicle being moved is assigned on the counter and speed is assigned to the respective vehicle.

- Step 8- Stop.

3.4 CODE:

```
import cv2

import numpy as np

import vehicles

import time

cnt_up=0

cnt_down=0

x = str(input("enter the video name:"))

cap=cv2.VideoCapture(x)

w=cap.get(3)

h=cap.get(4)

frameArea=h*w

areaTH=frameArea/400

line_up=int(2*(h/5))

line_down=int(3*(h/5))

up_limit=int(1*(h/5))

down_limit=int(4*(h/5))

print("Red line y:",str(line_down))

print("Blue line y:",str(line_up))

line_down_color=(255,0,0)
```

```
line_up_color=(255,0,255)

pt1 = [0, line_down]

pt2 = [w, line_down]

pts_L1 = np.array([pt1,pt2], np.int32)

pts_L1 = pts_L1.reshape((-1,1,2))

pt3 = [0, line_up]

pt4 = [w, line_up]

pts_L2 = np.array([pt3,pt4], np.int32)

pts_L2 = pts_L2.reshape((-1,1,2))

pt5 = [0, up_limit]

pt6 = [w, up_limit]

pts_L3 = np.array([pt5,pt6], np.int32)

pts_L3 = pts_L3.reshape((-1,1,2))

pt7 = [0, down_limit]

pt8 = [w, down_limit]

pts_L4 = np.array([pt7,pt8], np.int32)

pts_L4 = pts_L4.reshape((-1,1,2))

fgbg=cv2.createBackgroundSubtractorMOG2(detectShadows=True)

kernalOp = np.ones((3,3),np.uint8)

kernalOp2 = np.ones((5,5),np.uint8)

kernalCl = np.ones((11,11),np.uint)

font = cv2.FONT_HERSHEY_SIMPLEX
```

```
cars = []

max_p_age = 5

pid = 1

while(cap.isOpened()):

ret,frame=cap.read()

for i in cars:

i.age_one()

fgmask=fgbg.apply(frame)

fgmask2=fgbg.apply(frame)


if ret==True:

ret,imBin=cv2.threshold(fgmask,200,255,cv2.THRESH_BINARY)

ret,imBin2=cv2.threshold(fgmask2,200,255,cv2.THRESH_BINARY)

#OPeningi.e First Erode the dilate

mask=cv2.morphologyEx(imBin,cv2.MORPH_OPEN,kernalOp)

mask2=cv2.morphologyEx(imBin2,cv2.MORPH_CLOSE,kernalOp)

mask=cv2.morphologyEx(mask,cv2.MORPH_CLOSE,kernalCl)

mask2=cv2.morphologyEx(mask2,cv2.MORPH_CLOSE,kernalCl)

countours0,hierarchy=cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_AP
PROX_NONE)

for cnt in countours0:

area=cv2.contourArea(cnt)

print(area)
```

```
if area>areaTH:

m=cv2.moments(cnt)

cx=int(m['m10']/m['m00'])

cy=int(m['m01']/m['m00'])

x,y,w,h=cv2.boundingRect(cnt)

new=True

if cy in range(up_limit,down_limit):

for i in cars:

if abs(x - i.getX()) <= w and abs(y - i.getY()) <= h:

new = False

i.updateCoords(cx, cy)


if i.going_UP(line_down,line_up)==True:

cnt_up+=1

print("ID:",i.getId(),'crossed going up at', time.strftime("%c"))

elifi.going_DOWN(line_down,line_up)==True:

cnt_down+=1

print("ID:", i.getId(), 'crossed going up at', time.strftime("%c"))

break

if i.getState()=='1':

if i.getDir()=='down'andi.getY()>down_limit:

i.setDone()
```

```
elifi.getDir()=='up'andi.getY()<up_limit:

i.setDone()

if i.timedOut():

index=cars.index(i)

cars.pop(index)

del i


if new==True: #If nothing is detected,create new

p=vehicles.Car(pid,cx,cy,max_p_age)

cars.append(p)

pid+=1


cv2.circle(frame,(cx,cy),5,(0,0,255),-1)

img=cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)


for i in cars:

cv2.putText(frame, str(i.getId()), (i.getX(), i.getY()), font, 0.3, i.getRGB(), 1, cv2.LINE_AA)



str_up='UP: '+str(cnt_up)

str_down='DOWN: '+str(cnt_down)
```

```
frame=cv2.polylines(frame,[pts_L1],False,line_down_color,thickness=2)

frame=cv2.polylines(frame,[pts_L2],False,line_up_color,thickness=2)

frame=cv2.polylines(frame,[pts_L3],False,(255,255,255),thickness=1)

frame=cv2.polylines(frame,[pts_L4],False,(255,255,255),thickness=1)

cv2.putText(frame, str_up, (10, 40), font, 0.5, (255, 255, 255), 2, cv2.LINE_AA)

cv2.putText(frame, str_up, (10, 40), font, 0.5, (0, 0, 255), 1, cv2.LINE_AA)

cv2.putText(frame, str_down, (10, 90), font, 0.5, (255, 255, 255), 2, cv2.LINE_AA)

cv2.putText(frame, str_down, (10, 90), font, 0.5, (255, 0, 0), 1, cv2.LINE_AA)

cv2.imshow('Frame',frame)


if cv2.waitKey(1)&0xff==ord('q'):

break


else:

break


cap.release()

cv2.destroyAllWindows()
```

Chapter 4

RESULTS AND DISCUSSIONS

4.1 SIGNIFICANCE OF THE RESULTS OBTAINED:

- The goal of this project is to keep a track on the vehicles that are being flowed on the road and a counter is found which updates the count parallely.
- Normally after the sample video is processed into frames and performs noise cancellation, the sample video now gets updated with a counter which counts it efficiently.
- Here we find both UP and DOWN counters which track the count of both up and down moving vehicles on the road and speed of the vehicles is assigned to the vehicle itself.
- This application helps to determine the traffic in a specific area and highway tolls for better management where tracking of vehicles is needed.

4.2 OUTPUT SNAPSHOT:

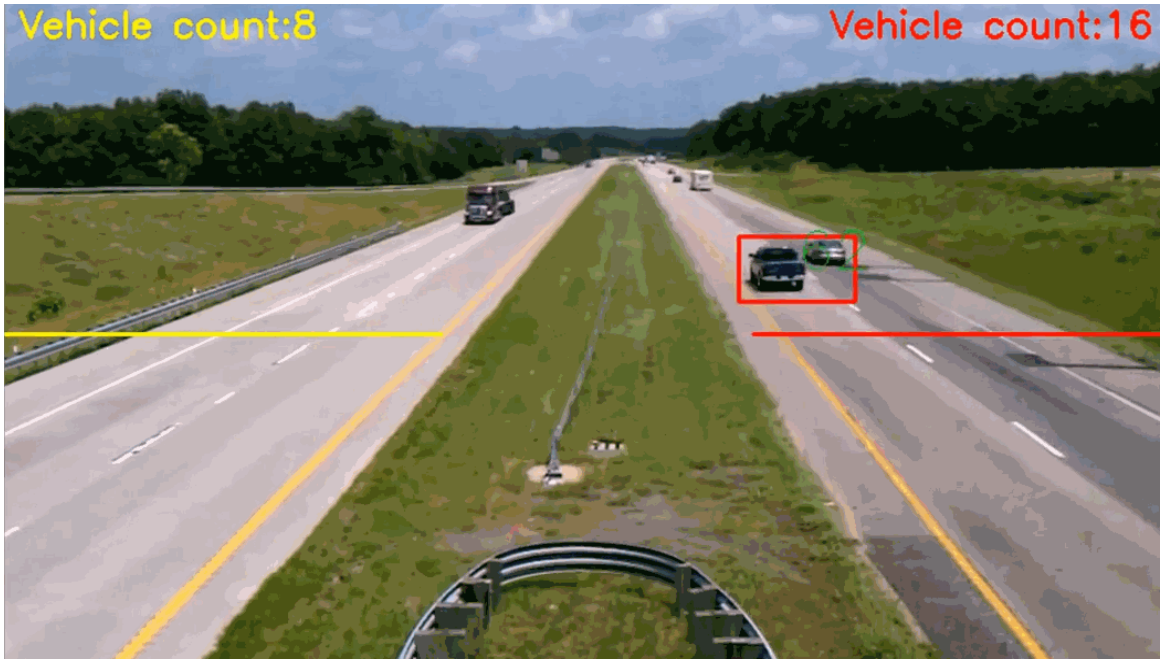

Fig 4.2.1 Counting the number vehicles passing on either side of the road.

CONCLUSION

The proposed system is an attempt the hardware and manual methods of counting and tracking the vehicles that are flown on the roads. This application project helps to avoid traffic jams in well populated roads thus taking measures to reduce it. This project is an automated system of displaying the count of vehicles by using Open CV and tensorflow libraries. The number of vehicles that are passing through are counted and thus we can determine the traffic rate of a particular area or a road. Speed can also be determined which also helps in tracking fast moving vehicles which are moving off the speed limit in restricted roads.

REFERENCES

- Real time traffic density measurement-Brac University
- Automatic car detection-IIT Kanpur
- Stack Overflow website
- Think Python - by Allen B.Downey
- www.opencv.org
- www.tenserflow.org