# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Belagavi – 590018, Karnataka State, India

PROJECT ENTITLED

**"To Develop an Efficient "Critical Vehicle Seamless Movement" Technique using AI and ML Methods"**

Submitted in partial fulfilment of the requirements for the award of degree of

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**For the academic year 2021-2022**

**Submitted by:**


**SUJITH N E (1MV18CS113)**

**YALAMANCHILI SAI GOKUL (1MV18CS127)**


Project report Carried out at

**Sir M. Visvesvaraya Institute of Technology**

**Bengaluru - 562157**

Under Guidance of

**Dr. SOUMYA PATIL**

Associate Professor


**SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**HUNASAMARANAHALLI BENGALURU – 562157**

# SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY

Krishnadevaraya Nagar, International Airport Road,
Hunasmaranahalli, Bengaluru – 562157

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# CERTIFICATE

It is certified that the project entitled " **TO DEVELOP AN "EFFICIENT CRITICAL VEHICLE SEAMLESS MOVEMENT TECHNIQUE" USING AI AND ML METHODS** " is carried out by **SUJITH N E (1MV18CS113), YALAMANCHILI SAI GOKUL (1MV18CS127)** bonafide students of **Sir M Visvesvaraya Institute of Technology** in partial fulfilment for the award of the Degree of Bachelor of Engineering in Computer Science and Engineering of the **Visvesvaraya Technological University, Belagavi** during the year **2021- 2022**. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the course of Bachelor of Engineering.

| Name & Signature of Guide | Name & Signature of HOD | Name & Signature of Principal |
|---|---|---|
| **Dr. Soumya Patil**<br>Asso. Prof & Internal Guide<br>Dept. Of CSE, Sir MVIT<br>Bengaluru - 562157 | **Dr. G. C. Bhanu Prakash**<br>HOD, Dept. Of CSE,<br>Sir MVIT<br>Bengaluru - 562157 | **Dr. V.R. Manjunath**<br>Principal,<br>Sir MVIT<br>Bengaluru - 562157 |

External Examination:

| Name of Examiner | Signature with Date |
|---|---|
| 1) | |
| 2) | |

# DECLARATION

We hereby declare that the project report embodied in this dissertation has been carried out by us and no part has been submitted for any degree or diploma of any institution previously.

Place: Bengaluru

Date:

Signature of Student:

| | |
|---|---|
| _____ | _____ |
| SUJITH N E | YALAMANCHILI SAI GOKUL |
| (1MV18CS113) | (1MV18CS127) |

# ACKNOWLEDGMENT

It gives us immense pleasure to express our sincere gratitude to the management of **Sir M. Visvesvaraya Institute of Technology,** Bengaluru for providing the opportunity and the resources to accomplish our project work in their premises.

On the path of learning, the presence of an experienced guide is indispensable, and we would like to thank our guide **Dr. Soumya Patil,** Associate Professor, Dept. of CSE, for her invaluable help and guidance.

Heartfelt and sincere thanks to **Dr. G. C. Bhanu Prakash,** HOD, Dept. of CSE, for his suggestions, constant support, and encouragement.

We would also like to convey our regards to **Dr. V.R. Manjunath,** Principal, Sir. MVIT for providing us with the infrastructure and facilities needed to develop our project.

We would also like to thank the staff of Department of Computer Science and Engineering and lab-in-charges for their co-operation and suggestions. Finally, we would like to thank all our friends for their help and suggestions without which completing this project would not have been possible.

- SUJITH N E 1MV18CS113
- YALAMANCHILI SAI GOKUL 1MV18CS127

# ABSTRACT

Traffic congestion is becoming one of the critical issues by posing significant hurdles to critical vehicles, to nature by posing significant pollution. Traffic congestion can not only hinder the movement of vital vehicles, but also causing additional delays and stress for drivers, as well as increasing fuel consumption and pollution levels. Although it seems to pervade everywhere, megacities are the ones most affected by traffic jams. And its ever-increasing nature makes it necessary to calculate the road traffic density in real-time for better signal control and effective traffic management. One of the most important variables affecting traffic flow is the traffic controller. As a result, there is a need to improve traffic control in order to better meet this growing demand. The goal of our proposed system is to use live images from traffic junction cameras to calculate traffic density using image processing and AI. It also focuses on the algorithm for switching traffic lights depending on the movement of critical vehicles and vehicle density to alleviate congestion, allowing critical vehicles movement and people to go faster while simultaneously lowering pollution.

# TABLE OF CONTENTS

# LIST OF FIGURES AND TABLES

| Table No. | Description | Page No. |
|---|---|---|

# CHAPTER 1

INTRODUCTION

Every city is quickly expanding in the modern day. As a result, the number of automobiles is rapidly expanding. With the growing number of automobiles in cities, many road networks are experiencing traffic congestion, and contemporary technology offers various methods to alleviate traffic congestion.

Traffic congestion is one of the numerous issues that the world is facing as a result of growing population and rapid expansion in the number of vehicles. In countries like India, the rate of road expansion is only one- third that of vehicular growth.

According to statistics, the present annual growth rate of automobiles is over 11%, while the annual road extension rate is just about 4%. The consequences of increased traffic congestion are numerous. Congestion stifles economic progress by delaying services, wasting fuel, and harming the environment.

According to studies, traffic congestion wastes 2.5 lakh litres of non-renewable fuel in a single day. Not only would this cause delays in vehicle transit, but it might also impede the movement of emergency vehicles. It is difficult for an emergency vehicle to cross through a signal due to traditional traffic signals. A traffic control system that continuously monitors vehicle density on the busy road can alleviate traffic congestion problems by adjusting the length of traffic signals based on vehicle density on the busy road.

The benefits of this sort of intelligent transportation system include lower capital and maintenance costs. Traffic data may be saved and successfully used for transportation planning in the future. Dynamic signal control is a complicated multi-criteria decision-making process that includes factors such as lowering latency, queuing spill backs, boosting productivity, reacting to events, and so on. Algorithms like yolov4 , the AI concept are said to offer potential capabilities for dealing with such decision-making processes. The signal control characteristics impacting decision making (such as flows, intensities, queues, and so on) are difficult to quantify and fluctuate across time and space. AI theory creates numerous types of knowledge in order to create adaptive engineering systems that can cope with complicated, nonlinear, and unpredictable stochastic interactions effectively and efficiently.

Emergency vehicles play an important role in saving lives as every second is as important as a life. Our project focuses on the serious impact that traffic congestion has on the emergency vehicle transportation system. Critical vehicles have a tough time navigating through traffic in places like India, where the road width and length make it hard to build a dedicated lane for emergency vehicles. Traffic management systems are essential for managing traffic congestion on the road, particularly during peak hours and peak seasons. One of the most difficult issues is controlling traffic when there are emergency situations at traffic signal intersections, especially during peak hours.

This could affect the route for emergency vehicles such as ambulance, fire brigade and police car to reach their destination. there is a need for further improvement needed for traffic control techniques.

# CHAPTER 2

## LITERATURE SURVEY

## 2.1 Arduino-UNO based traffic control system

Vogel, I. Oremović, R. Šimić and E. Ivanjko proposes an Arduino-UNO based system that aims to reduce traffic congestion and waiting time. This system acquires images through the camera and then processes the image in MATLAB, where the image is converted to a threshold image by removing saturation and hues, and traffic density is calculated. Arduino and MATLAB are connected using USB and simulation packages, which are preinstalled.

Depending on traffic count and traffic density, the Arduino sets the duration of green light for each lane. But this method has several flaws. The cars often overlap with each other and it is difficult to get a proper count of how many vehicles are on the road. Moreover, different objects interfered with the detection as they too were converted to black and white and there was no way of making a distinction between regular objects like billboards, poles, and trees with vehicles.

Vehicle Detection using Image Processing. Kanungo, A. Sharma and C. Singla, makes use of a support vector machine algorithm along with image processing techniques. From live video. images in small frames are captured and the algorithm is applied. Image processing is done using OpenCV and the images are converted to grayscale images before SVM is applied. This system not only detects traffic density but also detects red light violations.

## 2.2 Comparing Various existing methods

Ms. Saili Shinde, Prof. Sheetal Jagtap, reviews various techniques used for traffic light management system. This paper observes that each technique has a common architecture: choose input data, acquire traffic parameters from input data, process it, determine density, and update parameters.

In the first method, VANETS are used to get information and location of every vehicle, which in turn is passed on to the nearest Intelligent Traffic light with the help of installed GPS Further, these ITLs will update the statistics and sent it to nearby vehicles. In case of accidents, the information would be sent to drivers to choose an alternate route to avoid congestion. However, this technique is not feasible as its deployment is quite expensive.

In the second method, infrared sensor-based microcontrollers are used, which capture the unique ID of every car using transmitter and receiver. In case of an emergency situation, vehicle's radio frequency tags can be used to identify them and let other vehicles move. This method detects red light violations. However, this technique is not flexible due to the fact that infrared sensors need to be in sight.

In the third method, fuzzy logic technique is used in which two fuzzy logic controllers are used – one is to optimize the signal and the other controller is used to extend the green phase of a road in an intersection. The sensors used to collect input data are video cameras that are placed

at incoming and outgoing lines. The controller then utilizes the information collected through these sensors to make optimal decisions and minimize the goal function.

In the fourth method, fuzzy logic is used, and the system takes in the number of vehicles and the average speed of traffic flow in each direction as the input parameters. The number of vehicles and the average speed of traffic flow can be determined using sensors placed on the road.

In the fifth method, photoelectric sensors are used, which are set at some distance apart, that capture data and send it to the traffic cabinet, which calculates the weight of each road and then set the traffic light accordingly. However, the maintenance cost is quite high.

In the sixth method, video imaging is used to capture the data. Dynamic background subtraction and various morphological operations are performed to capture a clear image of the vehicle. Every time a new vehicle enters the area of interest, a new rectangle is drawn and vehicle count is incremented. The algorithm is easy to implement but does not handle occlusion and shadow overlapping.

### 2.3 Smart Traffic system using ANN and fuzzy controller

Renjith Soman proposes a smart traffic light system using ANN and fuzzy controller. This system makes use of images captured from cameras installed at traffic site. The image is first converted to a grayscale image before further normalization. Then, segmentation is performed using sliding window technique to count the cars irrespective of size and ANN is run through the segmented image, the output of which is used in fuzzy controller to set timers for red and green light using crisp output. Results had an average error of 2% with execution time of 1.5 seconds.

### 2.4 Simulation Environment.

A. Maria, explains a simulation is an imitation of a model based on a system in reality. A simulation model can change different parameters, then perform testing for a system that is costly or difficult to be constructed. A simulation model can be used to investigate different states . A simulation environment can be built to show the scenario by using the new traffic light application, comparing with the current FCTL system. Moreover, it can simulate the operations in different traffic conditions by using the new application.

# CHAPTER 3

# SYSTEM REQUIREMENT AND SPECIFICATION

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

## 3.1 Development Requirements

### 3.1.1 Hardware Requirements

PROCESSOR : i5

RAM : 4GB

HARD DISK : 16GB

### 3.1.2 Software Requirements

OPERATING SYSTEM : Linux/Windows/Mac

BACK-END : Python 3

OTHER BACKEND LIBRARIES : Pygame, darknet

FRAMEWORKS : yolov4

OTHER REQUIREMENTS : Google Colaboratory, Google Drive, labelimg

# CHAPTER 4

## SYSTEM ANALYSIS

## 4.1 Existing Systems

1) A smart traffic light system using ANN and fuzzy controller. This system makes use of images captured from cameras installed at traffic site. The image is first converted to a grayscale image before further normalization. Then, segmentation is performed using sliding window technique to count the cars irrespective of size and ANN is run through the segmented image, the output of which is used in fuzzy controller to set timers for red and green light using crisp output. Results had an average error of 2% with execution time of 1.5 seconds.

2) Infrared sensor-based microcontrollers are used, which capture the unique ID of every car using transmitter and receiver. In case of an emergency situation, vehicle's radio frequency tags can be used to identify them and let other vehicles move. This method detects red light violations. However, this technique is not flexible due to the fact that infrared sensors need to be in sight.

3) The third method employs the fuzzy logic methodology, which employs two fuzzy logic controllers, one for signal optimization and the other for extending the green phase of a road in an intersection. Video cameras are utilised to collect input data and are placed at the incoming and outgoing lines. The controller then uses the data gathered by these sensors to make the best judgments possible and minimise the objective function

## 4.2 Proposed System

Our proposed system takes images from the CCTV cameras at traffic junctions as input for real-time critical vehicle detection and traffic density calculation using image processing and object detection. This image is initially sent to the vehicle detection algorithm. The number of vehicles in each class, such as ambulance, cars, bicycles, buses, and trucks are calculated. The traffic density is then calculated using the weightage allotted to different classes of vehicles. The signal switching algorithm uses this density, along with a few other factors including the presence of critical vehicles, to determine the green signal time for each lane. The red signal times have been modified to reflect this. In order to prevent lane starvation, the green signal time is limited to a maximum and minimum value. To demonstrate the system's effectiveness, a simulation is created.

This project can be broken down into 3 modules:

1)Vehicle Detection Module.

2) Signal Switching Module.

3)Simulation Module.

**Vehicle Detection Module** : This module is in charge of detecting class of vehicles and number of vehicles in the image using YOLO algorithm. More specifically, each class of vehicle is given a specified weightage based on its importance, size, and the approximate number of passengers that may travel in it, so that a realistic traffic density can be determined. The dataset for training the model will be prepared by scraping images from google and labelling them manually. YOLOV4 is a smart convolutional neural network (CNN) that can conduct real-time object detection. The technique divides the image into areas and predicts bounding boxes for each region using a single neural network applied to the entire image.

**Signal Switching Module** : This algorithm updates the red, green, and yellow timers of all signals. These timers are set bases on the count of vehicles of each class received from the vehicle detection module and several other factors such as the number of lanes, average speed of each class of vehicle, and so on. The findings from the vehicle detection module are used to distinguish critical vehicles from other vehicles. The signal would then be changed based on the critical vehicles' position.

**Simulation Module** : To simulate traffic lights and automobiles moving across a traffic intersection, a simulation is created from scratch using the Pygame module. Pygame is a set of cross- platform Python tools for creating video games. It offers sound and graphics libraries that can be utilised with the Python programming language.

The simulation contains a four-way intersection with four traffic lights. On top of each signal is a timer that displays the amount of time until the signal changes from green to red or red to green . Cars, bicycles, buses, trucks, and rickshaws arrive from all directions. Some of the vehicles in the rightmost lane turn to cross the intersection to make the simulation more realistic.

# CHAPTER 5

## SYSTEM DESIGN

## 5.1 System Architecture

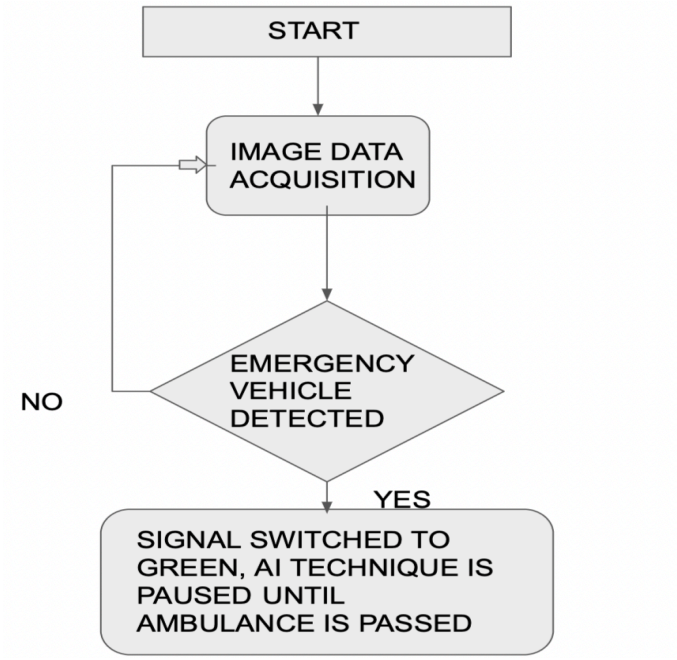



Fig 5.1 : System Architecture

A system architecture is the conceptual model that defines the structure and behaviour of the system.

Fig 5.1 shows the general architecture of the system. The system mainly consists of three steps:

**Image Acquisition** – here the images are taken from the cctv and are processed using yolov4 object detection algorithm, this algorithm specifies the vehicles type, and sends this data to next step.

**Emergency Vehicle Detected** – if any vehicles class is ambulance or firetruck then it sends the information to the next step signal switching, if the vehicle type is not ambulance or firetruck then the process goes back to image acquisition.

**Signal Switching** – here the algorithm instantly converts the signal into green and pauses the adaptive timer until emergency vehicle passes the signal then the process is repeated normally.

# CHAPTER 6

IMPLEMENTATION

## 6.1 Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

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, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

## 6.2 YOLOV4

YOLO is short for You Only Look Once. It is a real-time object recognition system that can recognize multiple objects in a single frame. YOLO recognizes objects more precisely and faster than other recognition systems. It can predict up to 9000 classes and even unseen classes. The real-time recognition system will recognize multiple objects from an image and also make a boundary box around the object. It can be easily trained and deployed in a production system.

YOLO is based on a single Convolutional Neural Network (CNN). The CNN divides an image into regions and then it predicts the boundary boxes and probabilities for each region. It simultaneously predicts multiple bounding boxes and probabilities for those classes. YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.

YOLO is developed by Joseph Redmon. The introduction of the YOLO real-time object recognition system in 2016 is the cornerstone of object recognition research. This led to better and faster Computer Vision algorithms.

YOLO v4 is developed by three developers Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao.

YOLOv4 is twice as fast as EfficientDet (competitive recognition model) with comparable performance. In addition, AP (Average Precision) and FPS (Frames Per Second) increased by 10% and 12% compared to YOLOv3.

YOLO is a futuristic recognizer that has faster FPS and is more accurate than available detectors. The detector can be trained and used on a conventional GPU which enables widespread adoption. New features in YOLOv4 improve accuracy of the classifier and detector and may be used for other research projects.

### 6.3 DARKNET

Darknet is a high performance open source framework for the implementation of neural networks. Written in C and CUDA, it can be integrated with CPUs and GPUs. Advanced implementations of deep neural networks can be done using Darknet. These implementations include You Only Look Once (YOLO) for real-time object detection, ImageNet classification, recurrent neural networks (RNNs), and many others.

### 6.4 GOOGLE COLAB

Google Colab was developed by Google to provide free access to GPU's and TPU's to anyone who needs them to build a machine learning or deep learning model. Google Colab can be defined as an improved version of Jupyter Notebook.

Nowadays, machine learning and deep learning has become the hottest trend of the Computer Science industry. Many students are trying to learn and build amazing projects with it. We all know that just studying or reading or watching a tutorial is of no use if you didn't try it out on your own. But in order to do that, you need really advanced specifications, for your system to withstand such a workload. And not everyone can afford a laptop with such specifications. So what can they do to learn and practice machine learning?

Google Colab is the answer.

### 6.5 PYGAME

Python is the most popular programming language or nothing wrong to say that it is the next-generation programming language. In every emerging field in computer science, Python makes its presence actively. Python has vast libraries for various fields such as Machine Learning (Numpy, Pandas, Matplotlib), Artificial intelligence (Pytorch, TensorFlow), and Game development (Pygame,Pyglet).

# CHAPTER 7

## SOFTWARE IMPLEMENTATION

A system is developed which detects the different kinds on vehicle from the given input images which are in jpg/jpeg format.

The project is divided in to two parts. The first is the detection of vehicles, and the second is traffic simulation.

To begin, we must label the images for training purposes; the labels for these images are saved in a.txt file. The dataset is then trained using these labelled images. We get weights for the dataset after training, and we use these weights to detect vehicles in the images.

### 7.1 Creating Dataset by labelling each Image

- we need a suitable dataset to train our custom object detection model.
- we will Label the images using LabelImg.
- LabelImg is a graphical image annotation tool. The Annotations are saved as TXT files in YOLO format. The below image shown how the Annotation tool look like:

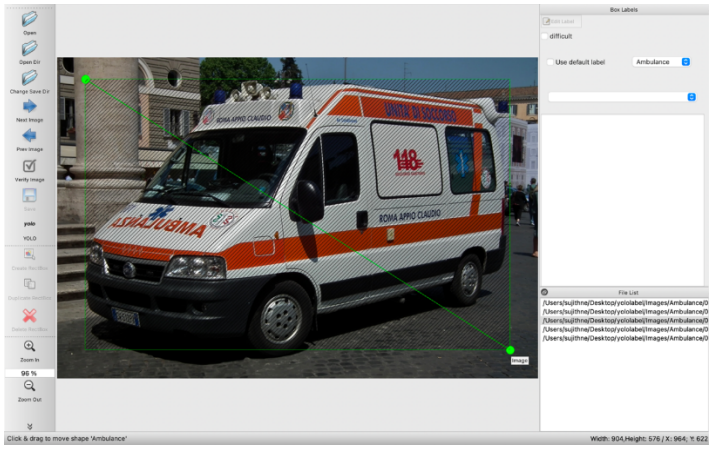


Fig 7.1(a) labelling

Fig 7.1(b) labelling

The output of the following Format is given below:

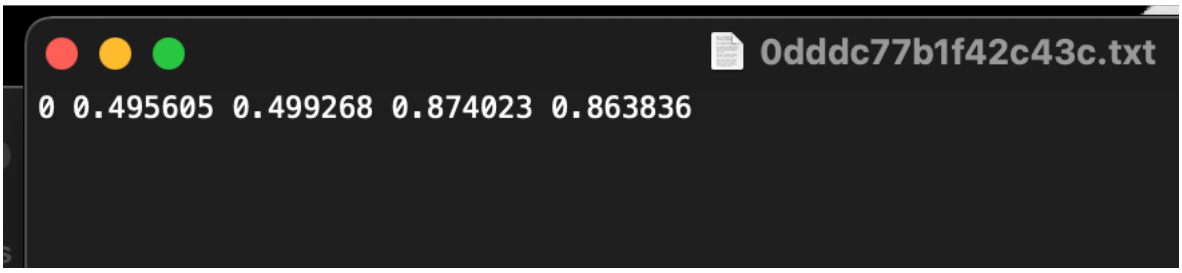



Fig 7.1(c) Text file of labelled Image


Fig 7.2(a) Input Image

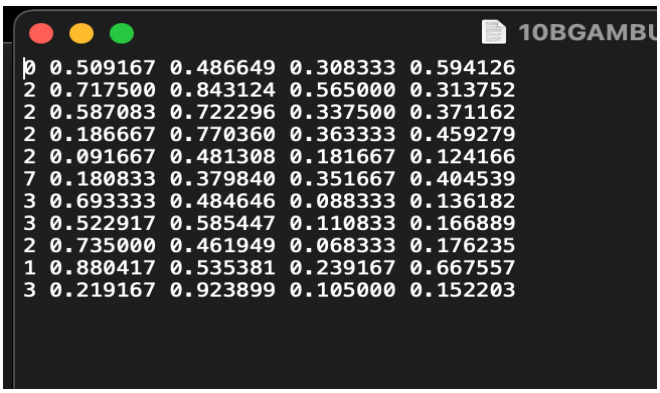



Fig 7.2(b) Text file of labelled Image

### 7.2 Training the Dataset

Our dataset is trained using yolov4.

YOLOv4 is known for its up-gradation in terms of AP and FPS. YOLOv4 prioritizes real-time object detection and training takes place on a single CPU. YOLOv4 has obtained state-of-art results on the COCO dataset with 43.5% speed (AP) at 65 Performance (FPS) on Tesla V100. This achievement is the result of a combination of the features like DropBlock Regularization,

Data Augmentation, Mish-Activation, CrossStage-Partial-connections (CSP), Self-adversarial-training (SAT), Weighted-Residual-Connections (WRC) and many more.

There are two types of models, one and two-staged object detectors. In two-stage detectors works in two parts that are first regions of importance are detected and then regions are classified to see if the object is detected in that particular region. YOLOv4 being a single staged object detector works more accurate and faster than Two staged detectors like R-CNN, Fast R-CNN.

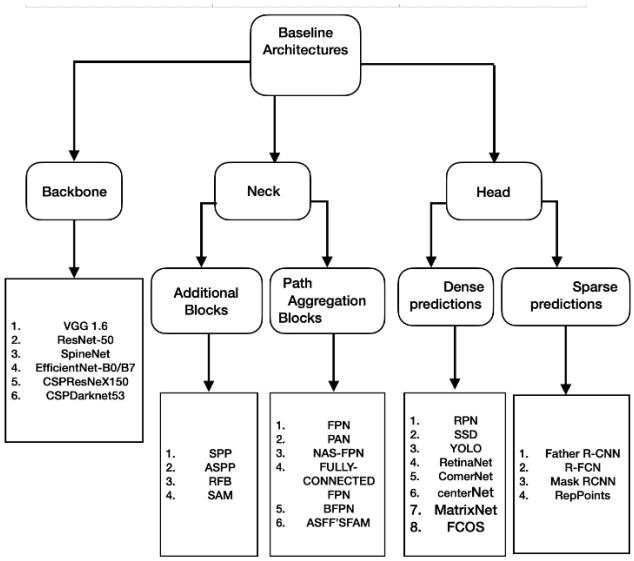



Fig 7.3 Basic Architecture of YOLOv4

## 7.3 Pygame

Pygame is a cross-platform set of Python modules which is used to create video games. It consists of computer graphics and sound libraries designed to be used with the Python programming language. Pygame was officially written by Pete Shinners to replace PySDL. Pygame is suitable to create client-side applications that can be potentially wrapped in a standalone executable.

pygame is a Python wrapper for the SDL library, which stands for Simple DirectMedia Layer. SDL provides cross-platform access to your system's underlying multimedia hardware components, such as sound, video, mouse, keyboard, and joystick. pygame started life as a replacement for the stalled PySDL project. The cross-platform nature of both SDL and pygame means you can write games and rich multimedia Python programs for every platform that supports them!

To install pygame on your platform, use the appropriate pip command:
$ pip install pygame

# CHAPTER 8

SOURCE CODE

## 8.1 Vehicle Detection Model

To clone Darknet

- The following cells will clone darknet from AlexeyAB's repository, adjust the Makefile to enable OPENCV and GPU for darknet and then build darknet.

```
!git clone https://github.com/AlexeyAB/darknet
```

- change makefile to have GPU and OPENCV enabled

```
%cd /content/darknet/
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

- make darknet (builds darknet so that you can then use the darknet executable file to run or train object detectors)

```
!make
```

- YOLOv4 has been trained already on the coco dataset which has 80 classes that it can predict. We will grab these pretrained weights so that we can run YOLOv4 on these pretrained classes and get detections.

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/
yolov4-tiny.conv.29
```

- These three functions are helper functions that will allow you to show the image in your Colab Notebook after running your detections, as well as upload and download images to and from your Cloud VM.

```
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline

  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_
CUBIC)
```

```
  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()

# use this to upload files
def upload():
  from google.colab import files
  uploaded = files.upload()
  for name, data in uploaded.items():
    with open(name, 'wb') as f:
      f.write(data)
      print ('saved file', name)

# use this to download a file
def download(path):
  from google.colab import files
  files.download(path)
```

- Mount Google Drive

```
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')
```

- This creates a symbolic link so that now the path /content/gdrive/My\ Drive/ is equal to /mydrive

```
!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive
```

- Change Directory back into the darknet folde to run detection.

```
%cd darknet
```

- This is where my datasets are stored within my Google Drive (I created a detecting-vehicle folder to store all important files for custom training)

```
!ls /mydrive/detecting-vehicle
```

- Copy over both datasets into the root directory of the Colab VM

```
!cp /mydrive/detecting-vehicle/obj.zip ../
!cp /mydrive/detecting-vehicle/test.zip ../
```

- unzip the datasets and their contents so that they are now in /darknet/data/ folder

```
!unzip ../obj.zip -d data/
!unzip ../test.zip -d data/
```

- download cfg to google drive and change its name
- Changes Required in custom configuration file.
- Change line batch to batch=64
- Change line subdivisions to subdivisions=16
- Change line max_batches to (classes*2000 but not less than number of training images, but not less than number of training images and not less than 6000), i.e. max_batches=6000 if you train for 3 classes
- Change line steps to 80% and 90% of max_batches, f.e. steps=4800,5400 set network size width=416 height=416 or any value multiple of 32
- Change line classes=80 to your number of objects in each of 2 [yolo]-layers
- Change [filters=255] to filters=(classes + 5)x3 in the 2 [convolutional] before each [yolo] layer, keep in mind that it only has to be the last [convolutional] before each of the [yolo] layers. So if classes=1 then it should be filters=18. If classes=2 then write filters=21.
- upload the custom .cfg back to cloud VM from Google Drive

```
!cp /mydrive/detecting-vehicle/yolov4-obj.cfg ./cfg
```

- Create a new file within a code or text editor called obj.names where you will have one class name per line in the same order as your classes.txt from the dataset generation step.

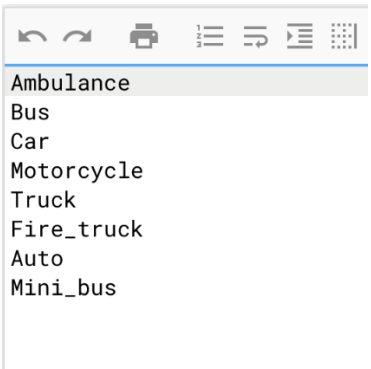



Fig 8.1 object.name

- obj.data

```
classes = 8
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = /mydrive/detecting-vehicle/backup
```

Fig 8.2 object.data

- upload the obj.names and obj.data files to cloud VM from Google Drive

```
!cp /mydrive/detecting-vehicle/obj.names ./data
!cp /mydrive/detecting-vehicle/obj.data  ./data
```

- code in generate_train.py

```
import os

image_files = []
os.chdir(os.path.join("data", "obj"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        image_files.append("data/obj/" + filename)
os.chdir("..")
with open("train.txt", "w") as outfile:
    for image in image_files:
        outfile.write(image)
        outfile.write("\n")
    outfile.close()
os.chdir("..")
```

Fig 8.3 generate_train.py

- generate_test.py

```
import os

image_files = []
os.chdir(os.path.join("data", "test"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        image_files.append("data/test/" + filename)
os.chdir("..")
with open("test.txt", "w") as outfile:
    for image in image_files:
        outfile.write(image)
        outfile.write("\n")
    outfile.close()
os.chdir("..")
```

Fig 8.4 generate_test.py

- upload the generate_train.py and generate_test.py script to cloud VM from Google Drive

```
!cp /mydrive/detecting-vehicle/generate_train.py ./
!cp /mydrive/detecting-vehicle/generate_test.py ./
```

- run both scripts to do the work for you of generating the two txt files.

  !python generate_train.py
  !python generate_test.py

- This step downloads the weights for the convolutional layers of the YOLOv4 network. By using these weights it helps your custom object detector to be way more accurate and not have to train as long.

  !wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137

- train your custom detector (dont_show flag stops chart from popping up since Colab Notebook can't open images on the spot, -map flag overlays mean average precision on chart to see how accuracy of your model is, only add map flag if you have a validation dataset)

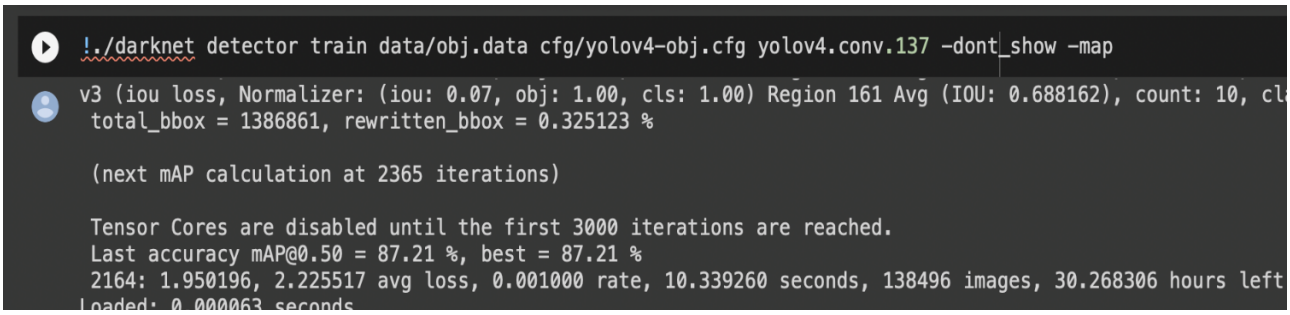



Fig 8.5 vehicle-detection.ipynb

- To restart training from where it last saved

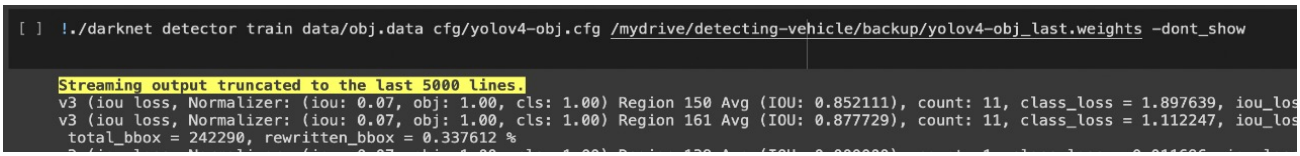



Fig 8.6 vehicle-detection.ipynb

- Checking the Mean Average Precision (mAP) of Your Model

  !./darknet detector map data/obj.data cfg/yolov4-obj.cfg /mydrive/detecting-vehicle/backup/yolov4-obj_last.weights

  detections_count = 427, unique_truth_count = 195
  class_id = 0, name = Ambulance, ap = 99.58% (TP = 28, FP = 2)
  class_id = 1, name = Bus, ap = 100.00% (TP = 20, FP = 1)
  class_id = 2, name = Car, ap = 94.40% (TP = 44, FP = 2)
  class_id = 3, name = Motorcycle, ap = 97.06% (TP = 34, FP = 2)
  class_id = 4, name = Truck, ap = 96.98% (TP = 34, FP = 1)
  class_id = 5, name = Fire_truck, ap = 98.81% (TP = 11, FP = 0)
  class_id = 6, name = Auto, ap = 100.00% (TP = 4, FP = 1)
  class_id = 7, name = Mini_bus, ap = 100.00% (TP = 11, FP = 0)

## 8.2 Simulation Model

- Main function

```
class Main:
    thread4 = threading.Thread(name="simulationTime",target=simulationTime, args=())
    thread4.daemon = True
    thread4.start()

    thread2 = threading.Thread(name="initialization",target=initialize, args=())    # initialization
    thread2.daemon = True
    thread2.start()

    # Colours
    black = (0, 0, 0)
    white = (255, 255, 255)

    # Screensize
    screenWidth = 1400
    screenHeight = 800
    screenSize = (screenWidth, screenHeight)

    # Setting background image i.e. image of intersection
    background = pygame.image.load('images/mod_int.png')

    screen = pygame.display.set_mode(screenSize)
    pygame.display.set_caption("SIMULATION")

    # Loading signal images and font
    redSignal = pygame.image.load('images/signals/red.png')
    yellowSignal = pygame.image.load('images/signals/yellow.png')
    greenSignal = pygame.image.load('images/signals/green.png')
    font = pygame.font.Font(None, 30)

    thread3 = threading.Thread(name="generateVehicles",target=generateVehicles, args=())    # Generating vehicles
    thread3.daemon = True
    thread3.start()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        screen.blit(background,(0,0))   # display background in simulation
        for i in range(0,noOfSignals):  # display signal and set timer according to current status: green, yello, or red
            if(i==currentGreen):
                if(currentYellow==1):
                    if(signals[i].yellow==0):
                        signals[i].signalText = "STOP"
                    else:
                        signals[i].signalText = signals[i].yellow
                    screen.blit(yellowSignal, signalCoods[i])
                else:
                    if(signals[i].green==0):
                        signals[i].signalText = "SLOW"
                    else:
                        signals[i].signalText = signals[i].green
                    screen.blit(greenSignal, signalCoods[i])
            else:
                if(signals[i].red<=10):
                    if(signals[i].red==0):
                        signals[i].signalText = "GO"
                    else:
                        signals[i].signalText = signals[i].red
```

Fig 8.7 Simulation.py

- Generating Vehicles

```
# Generating vehicles in the simulation
def generateVehicles():
    cnt=0
    while(True):
        cnt+=1
        if cnt==115:
            vehicle_type = 5
            direction_number = 2
            lane_number = 1
            will_turn = 0


        else:
            vehicle_type = random.randint(0,4)
            if(vehicle_type==4):
                lane_number = 0
            else:
                lane_number = random.randint(0,1) + 1
            will_turn = 0
            if(lane_number==2):
                temp = random.randint(0,4)
                if(temp<=2):
                    will_turn = 1
                elif(temp>2):
                    will_turn = 0
            temp = random.randint(0,999)
            direction_number = 0
            a = [400,800,900,1000]
            if(temp<a[0]):
                direction_number = 0
            elif(temp<a[1]):
                direction_number = 1
            elif(temp<a[2]):
                direction_number = 2
            elif(temp<a[3]):
                direction_number = 3
        Vehicle(lane_number, vehicleTypes[vehicle_type], direction_number, directionNumbers[direction_number], will_turn)
        time.sleep(0.75)
```

Fig 8.8 Simulation.py

- Set Signal Time

```
# Set time according to formula
def setTime():
    global noOfCars, noOfBikes, noOfBuses, noOfTrucks, noOfRickshaws, noOfambulances, noOffiretrucks, noOfLanes
    global carTime, busTime, truckTime, rickshawTime, bikeTime, ambulanceTime, firetruckTime

    noOfCars, noOfBuses, noOfTrucks, noOfRickshaws, noOfBikes, noOfambulances, noOffiretrucks = 0,0,0,0,0,0,0
    for j in range(len(vehicles[directionNumbers[nextGreen]][0])):
        vehicle = vehicles[directionNumbers[nextGreen]][0][j]
        if(vehicle.crossed==0):
            vclass = vehicle.vehicleClass
            # print(vclass)
            noOfBikes += 1
    for i in range(1,3):
        for j in range(len(vehicles[directionNumbers[nextGreen]][i])):
            vehicle = vehicles[directionNumbers[nextGreen]][i][j]
            if(vehicle.crossed==0):
                vclass = vehicle.vehicleClass
                # print(vclass)
                if(vclass=='car'):
                    noOfCars += 1
                elif(vclass=='bus'):
                    noOfBuses += 1
                elif(vclass=='truck'):
                    noOfTrucks += 1
                elif(vclass=='rickshaw'):
                    noOfRickshaws += 1
                elif(vclass=='ambulance'):
                    noOfambulances += 1
                elif(vclass=='firetruck'):
                    noOffiretrucks += 1
    # print(noOfCars)
    greenTime = math.ceil(((noOfCars*carTime) + (noOfRickshaws*rickshawTime) + (noOfBuses*busTime) + (noOfTrucks*truckTime)+ (noOfBikes*bikeTime) + (noOfambulances*ambulanceTime) +
        (noOffiretrucks/firetruckTime))/(noOfLanes+1))
    # greenTime = math.ceil((noOfVehicles)/noOfLanes)
    print('Green Time: ',greenTime)
    if(greenTime<defaultMinimum):
        greenTime = defaultMinimum
    elif(greenTime>defaultMaximum):
        greenTime = defaultMaximum
    # greenTime = random.randint(15,50)
    signals[(currentGreen+1)%(noOfSignals)].green = greenTime
```

Fig 8.9 Simulation.py

# CHAPTER 9

## SYSTEM TESTING

## 9.1 INTRODUCTION

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

During the development of software, errors can be injected at any stage. However, requirements and design errors are likely to remain undetected. During testing, the program to be tested is executed with a set of test cases and output of program for test cases are evaluated to describe the program performance to expected level. No system design is perfect because of communication problem, programmer‘s negligence or constraints create errors that must be eliminated before the system is ready for use of acceptance. There are various types of test. Each test type addresses a specific testing requirement. Following sections describes the system testing and test cases.

## 9.2 UNIT TESTING

This is the first level of testing, where different components are tested against the requirement specification for the individual components. Unit testing is essential for verification of the code produced during the coding phase and hence the goal is to test internal logic of those components.

## 9.3 SYSTEM TESTING

System testing is performed on the entire system in the context of functional requirements specification and system requirement specifications. System testing tests not only the design, but also the behaviour and the expectations. It is also to test up to and beyond the bounds defined in the software requirement specification. System testing for our platform is accomplished by the following test cases. These test cases show how each and every component interacts with each other as a dynamic system.

# CHAPTER 10

## RESULTS AND SCREENSHOTS

## 10.1 Vehicle detection

- need to set our custom cfg to test mode

```
[ ]  %cd cfg
     !sed -i 's/batch=64/batch=1/' yolov4-obj.cfg
     !sed -i 's/subdivisions=16/subdivisions=1/' yolov4-obj.cfg
     %cd ..

     /content/darknet/cfg
     /content/darknet
```

Fig 10.1 vehicle-detection.ipynb

- run your custom detector with this command (upload an image to your google drive to test, thresh flag sets accuracy that detection must be in order to show it)

- **Input Images**


Fig 10.2 ambulance.ipg

- **Command to run prediction on image**

!./darknet detector test data/obj.data cfg/yolov4-obj.cfg /mydrive/detecting-vehicle/backup/yolov4-obj_last.weights /mydrive/Images/ambulance.jpg -thresh 0.3
imShow('predictions.jpg')

```
[ ] !./darknet detector test data/obj.data cfg/yolov4-obj.cfg /mydrive/detecting-vehicle/backup/yolov4-obj_last.weights /mydrive/Images/ambulance.jpg -thresh 0
    imShow('predictions.jpg')

 CUDA-version: 11010 (11020), cuDNN: 7.6.5, CUDNN_HALF=1, GPU count: 1
 CUDNN_HALF=1
 OpenCV version: 3.2.0
 0 : compute_capability = 750, cudnn_half = 1, GPU: Tesla T4
net.optimized_memory = 0
mini_batch = 1, batch = 1, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     64       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  64 0.354 BF
   3 route  1                                     ->  208 x 208 x  64
   4 conv     64       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  64 0.354 BF
   5 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   6 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   7 Shortcut Layer: 4,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   8 conv     64       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  64 0.354 BF
   9 route  8 2                                   ->  208 x 208 x 128
  10 conv     64       1 x 1/ 1    208 x 208 x 128 ->  208 x 208 x  64 0.709 BF
  11 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
  12 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
```

10.3 vehicle-detection.ipynb

```
Ambulance: 99%
Unable to init server: Could not connect: Connection refused

(predictions:1705): Gtk-WARNING **: 05:01:21.464: cannot open display:
```


Fig 10.4 Predicted Image


Fig 10.5(a) Input image

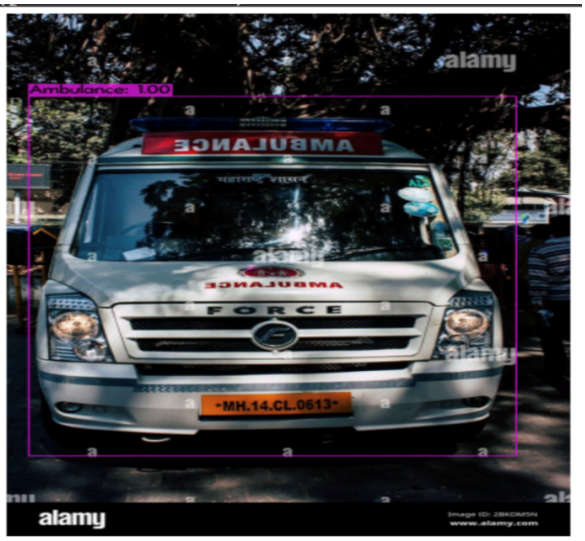



Fig 10.5(b) output Image


Fig 10.6(a) Input image


Fig 10.6(b) output Image

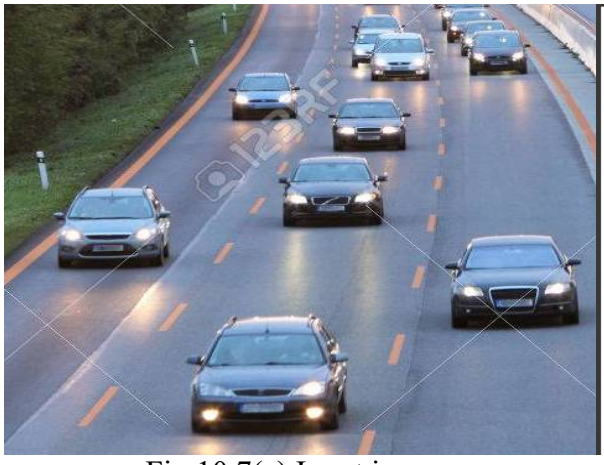


Fig 10.7(a) Input image

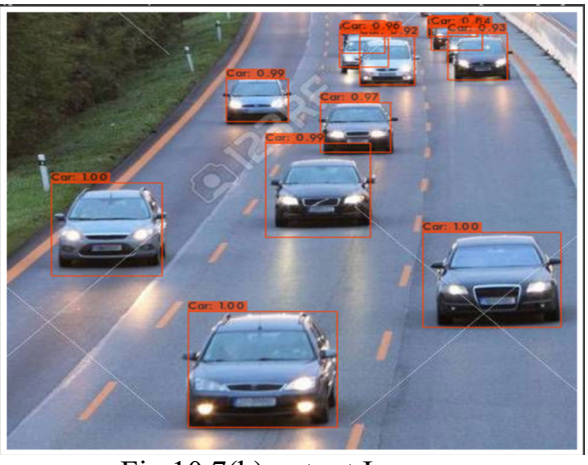



Fig 10.7(b) output Image


Fig 10.8(a) Input image

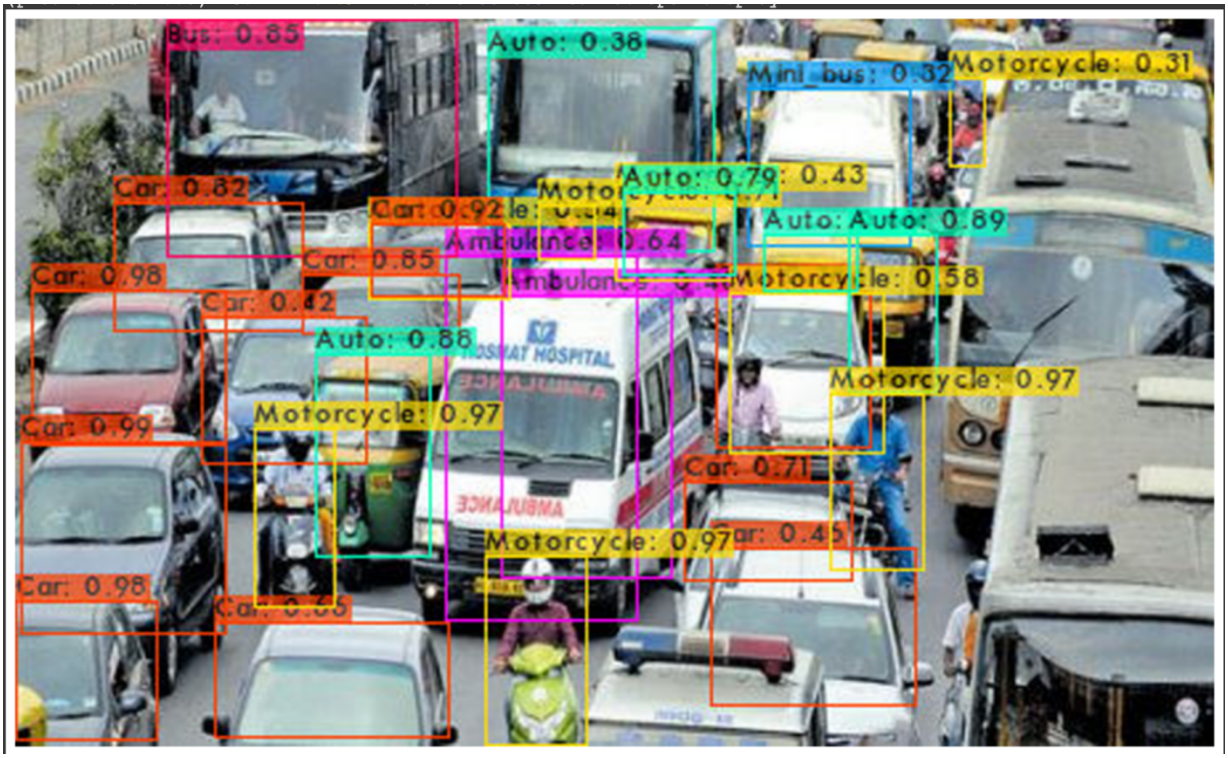



Fig 10.8(b) output Image

## 10.2 Simulation

A simulation was developed from scratch using Pygame to simulate real-life traffic. It assists in visualizing the system and comparing it with the existing static system. It contains a 4-way intersection with 4 traffic signals. Each signal has a timer on top of it, which shows the time remaining for the signal to switch from green to yellow, yellow to red, or red to green. Each signal also has the number of vehicles that have crossed the intersection displayed beside it. Vehicles such as cars, bikes, buses, trucks, rickshaws, ambulances and FireTruck come in from all directions. In order to make the simulation more realistic, some of the vehicles in the rightmost lane turn to cross the intersection. Whether a vehicle will turn or not is also set using random numbers when the vehicle is generated. It also contains a timer that displays the time elapsed since the start of the simulation. Fig. 10.7 shows a snapshot of the final output of the simulation.

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. Pygame adds functionality on top of the excellent SDL library. This allows users to create fully featured games and multimedia programs in the python language. Pygame is highly portable and runs on nearly every platform and operating system. It is free and licensed under LGPL.

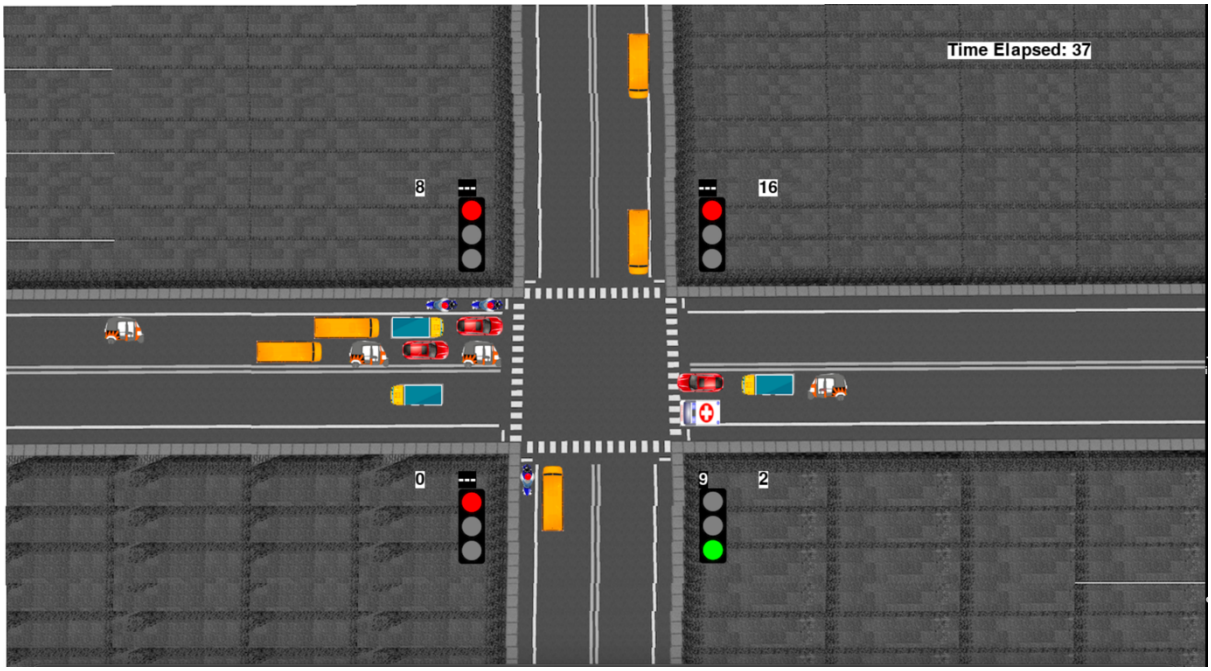



Fig 10.9 Simulation

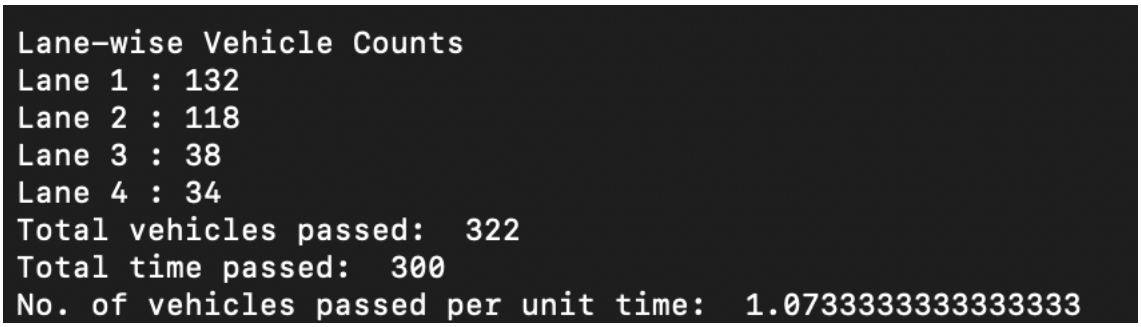



Fig 10.10 Total Vehicles Passed for a Time Period of 300 Sec

- **Evaluation of the proposed adaptive system**

To measure how the proposed adaptive system compares to the existing static system, 15 simulations of both the systems were run for a period of 5 minutes each, with varying traffic distributions across the 4 directions. Performance was measured in terms of the number of vehicles that were able to pass the intersection per unit of time. In other words, the idle time of the signal i.e. the time when the signal is green but no car passes the intersection is compared. This has an impact on the waiting time of vehicles and queue lengths of the other signals.

The distribution [a,b,c,d] means that the probability of a vehicle being in lane 1, lane 2, lane 3, and lane 4 is a/d, (b- a)/d, (c-b)/d, and (d-c)/d, respectively. For example, in simulation 1, the distribution is [300,600,800,1000] which means probabilities of 0.3, 0.3, 0.2, and 0.2. The results obtained were tabulated in the form of number of vehicles passed lane-wise and the total number of vehicles passed.

| No | Distribution | Lane 1 | Lane2 | Lane3 | Lane4 | Total |
|---|---|---|---|---|---|---|
| 1 | [300,600,800,1000] | 72 | 52 | 52 | 66 | 242 |
| 2 | [500,700,900,1000] | 113 | 50 | 48 | 32 | 243 |
| 3 | [250,500,750,1000] | 73 | 51 | 65 | 62 | 251 |
| 4 | [300,500,800,1000] | 76 | 45 | 64 | 71 | 256 |
| 5 | [700,800,900,1000] | 88 | 33 | 25 | 42 | 188 |
| 6 | [500,900,950,1000] | 95 | 71 | 16 | 14 | 196 |
| 7 | [300,600,900,1000] | 73 | 63 | 69 | 24 | 229 |
| 8 | [200,700,750,1000] | 54 | 89 | 10 | 67 | 220 |
| 9 | [940,960,980,1000] | 100 | 10 | 8 | 4 | 122 |
| 10 | [400,500,900,1000] | 81 | 29 | 88 | 37 | 235 |
| 11 | [200,400,600,1000] | 42 | 47 | 54 | 86 | 229 |
| 12 | [250,500,900,1000] | 39 | 52 | 93 | 22 | 206 |
| 13 | [850,900,950,1000] | 74 | 10 | 13 | 17 | 114 |
| 14 | [350,500,850,1000] | 49 | 46 | 69 | 50 | 214 |
| 15 | [350,700,850,1000] | 51 | 64 | 37 | 43 | 195 |

Table 10.1 SIMULATION RESULTS OF CURRENT STATIC SYSTEM

| No | Distribution | Lane 1 | Lane2 | Lane3 | Lane4 | Total |
|---|---|---|---|---|---|---|
| 1 | [300,600,800,1000] | 132 | 118 | 38 | 48 | 322 |
| 2 | [500,700,900,1000] | 165 | 60 | 72 | 23 | 320 |
| 3 | [250,500,750,1000] | 104 | 60 | 70 | 68 | 302 |
| 4 | [300,500,800,1000] | 99 | 56 | 79 | 69 | 303 |
| 5 | [700,800,900,1000] | 195 | 35 | 33 | 38 | 301 |
| 6 | [500,900,950,1000] | 104 | 128 | 21 | 26 | 279 |
| 7 | [300,600,900,1000] | 97 | 78 | 80 | 43 | 298 |
| 8 | [200,700,750,1000] | 66 | 118 | 29 | 88 | 301 |
| 9 | [940,960,980,1000] | 203 | 16 | 15 | 17 | 251 |
| 10 | [400,500,900,1000] | 107 | 39 | 110 | 44 | 300 |
| 11 | [200,400,600,1000] | 36 | 62 | 77 | 109 | 284 |
| 12 | [250,500,900,1000] | 62 | 85 | 111 | 17 | 275 |
| 13 | [850,900,950,1000] | 164 | 27 | 22 | 28 | 241 |
| 14 | [350,500,850,1000] | 74 | 63 | 90 | 57 | 284 |
| 15 | [350,700,850,1000] | 76 | 92 | 50 | 58 | 276 |

Table 10.2 SIMULATION RESULTS OF PROPOSED ADAPTIVE SYSTEM

The proposed adaptive system always outperforms the current static system, as shown in fig. 10.9, regardless of the distribution. The amount of improvement in performance is determined by how skewed the traffic distribution is across the lanes. The higher the skewness of the traffic distribution, the better the performance.

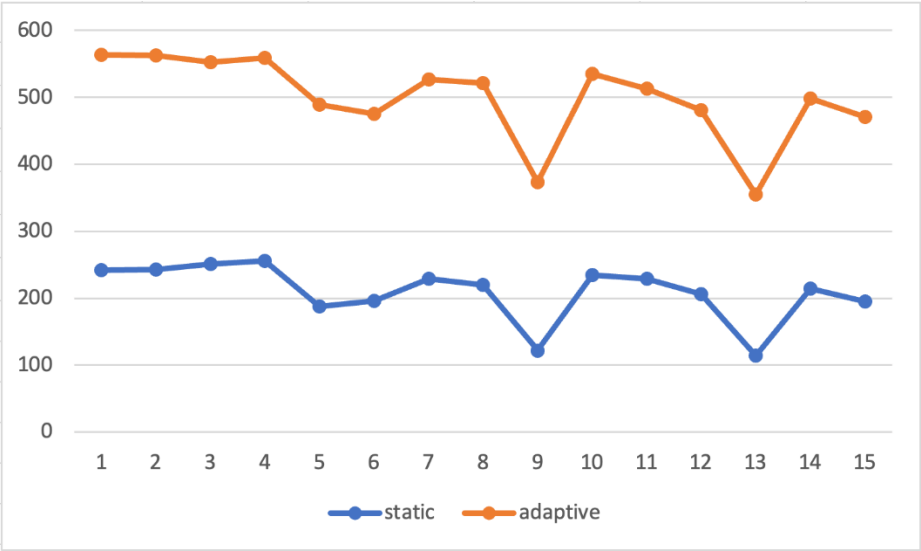



Fig 10.11 Comparison of current static system and proposed adaptive system

# CHAPTER 11

CONCLUSION

By following the above-mentioned steps and performing image processing, we can be able to achieve the following results:

1. This Adaptive Traffic Signal Timer uses live images from the cameras at traffic junctions for traffic density calculation using YOLO(You Only Look Once) object detection and sets the signal timers accordingly.

2. It can detect the emergency vehicles such as Ambulance and adapt to the given situation accordingly thus reducing the traffic congestion for the critical vehicles. Thus Signal switching would reduce the amount of time it takes for critical vehicles to pass.

3. People can anticipate to spend substantially less time stuck in traffic if the dynamic traffic control system is implemented, and they can use the timers to turn off their engines, decreasing fuel consumption and emissions accordingly and Reducing the amount of time spent at a traffic signal can help people be more productive.

This idea can be implemented for a larger network by using algorithms to ensure safety and stability of systems. Also, the extended time can be calculated by the system itself. By keeping records of traffic patterns and using an algorithm, the timing can be chosen according to the traffic patterns.

The project can be further expanded to include the following functionalities to improve traffic management and reduce traffic congestion:

1. Identification of vehicles violating traffic rules: Vehicles going through red lights can be spotted in an image or video stream by drawing a violation line and capturing the number plate if the line is crossed when the signal is red. Similarly, lane switching can be recognised. Background subtraction and image processing techniques can be used to accomplish these results.

2. Accident or breakdown detection: Intersections are prone to serious wrecks because numerous types of injurious collisions, such as angle and left-turn collisions, frequently occur there. As a result, precise and timely identification of accidents at junctions has enormous benefits in terms of saving property and lives while also reducing congestion and delay. This can be accomplished by identifying automobiles that remain motionless for an extended period in an inconvenient location, such as in the middle of the road, while excluding parked vehicles.

# CHAPTER 12

REFERENCES

[1]. A. Vogel, I. Oremović, R. Šimić and E. Ivanjko, "Improving Traffic Light Control by Means of Fuzzy Logic," 2018 International Symposium ELMAR, Zadar, 2018, pp. 51-56, doi: 10.23919/ ELMAR.2018.8534692.

[2] A. Kanungo, A. Sharma and C. Singla, "Smart traffic lights switching and traffic density calculation using video processing," 2014 Recent Advances in Engineering and Computational Sciences (RAECS), Chandigarh, 2014, pp. 1-6, doi: 10.1109/RAECS.2014.6799542.

[3] Ms. Saili Shinde, Prof. Sheetal Jagtap, Vishwakarma Institute Of Technology, Intelligent traffic management system:a Review, IJIRST 2016

[4] Renjith Soman "Traffic Light Control and Violation Detection Using Image Processing"." IOSR Journal of Engineering (IOSRJEN), vol. 08, no. 4, 2018, pp. 23-27

[5]. A. Maria, Introduction to modelling and simulation, in Proceed- ings of the 29th conference on Winter simulation, IEEE Computer Society, Atlanta, GA, USA, 1997, pp. 7 13.

[6]. Rajeshwari S., Santhoshs Hebbar, Varaprasad Golla " Implementing Intelligent Traffic Control System for Congestion Control, Ambulance Clearance and Stolen Vehicle Detection " DOI 10.1109/ JSEN.2014.2360288, IEEE Sensors Journal.

[7]. Abdoos M., Mozayani N and Bazzan A.L.C., "Traffic Light Control in Non- Stationary Environments based on Multi agent Q-learning", in Proc. IEEE Conference on Intelligent Transportation Systems, pp.580- 1585, 2011.

[8]. "Traffic Congestion in Bangalore-A Rising Concern", http://www.commonfloor.com/guide/traffic-congestion-in-bangalore-a-rising-concern27238.html.

[9]. I. Kosonen, "Multi-agent fuzzy signal control based on real-time simulation," vol. 11, no. 5, pp. 389– 403, 2003.

[10] Tzutalin, 'LabelImg Annotation Tool', 2015. [Online]. Available: https://github.com/tzutalin/labelImg

# CHAPTER 13

PUBLISHED PAPERS

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 10 Issue VII July 2022- Available at www.ijraset.com*

# To Develop an Efficient Critical Vehicle Seamless Movement Technique using AI and ML Methods

Prof. Dr. Soumya Patil[1], Sujith N E[2], Yalamanchili Sai Gokul[3]
[1]*Guide,* [2,3]*Student, Department Of Computer Science & Engineering, Sir M Visvesvaraya Institute Technology, Bengaluru, Karnataka, India.*

***Abstract:*** ***Traffic congestion is becoming one of the critical issues by posing significant hurdles to critical vehicles, to nature by posing significant pollution. Not only can traffic congestion delay the movement of essential vehicles, but it can also extend travel times, contribute to driver stress, and increase fuel consumption and pollution. Megacities struggle the most from traffic congestion, despite the fact that they seem to be everywhere. Because of this, real-time road traffic density calculations are necessary for better signal control and traffic management. The traffic controller is one of the key elements influencing traffic flow. Therefore, in order to better fulfil this expanding demand, traffic control needs to be improved. Our system's objective is to use real-time traffic junction camera photos to compute traffic density using AI and image processing. It also focuses on the algorithm for changing traffic signals based on the movement of essential vehicles and vehicle density to reduce congestion, speed up the movement of essential vehicles and reduce pollution.***

***Keywords: Traffic control, Traffic light system, Traffic management, Intelligent transport systems, Smart surveillance, Computer Vision, Machine Learning, Object detection, YOLO.***

## I. INTRODUCTION

One of the many problems the globe is experiencing as a result of population growth and the quick increase in vehicles is traffic congestion. The rate of road construction is only one-third that of vehicle growth in nations like India. Statistics show that the current annual growth rate of autos is above 11%, whereas the annual expansion rate of roads is just approximately 4%. The consequences of increased traffic congestion are numerous. Congestion stifles economic progress by delaying services, wasting fuel, and harming the environment. According to studies, traffic congestion wastes 2.5 lakh liters of non-renewable fuel in a single day. This can not only slow the flow of vehicles, but also obstruct the movement of emergency vehicles. Traditional traffic signals make it challenging for an emergency vehicle to pass through a signal. Since every second is as valuable as a life, emergency vehicles are crucial for saving lives. Delays in the emergency firemen services in an emergency situation have resulted in several lives and properties being lost. By creating an intelligent automated system that is integrated with a traffic control system and recognises and prioritises emergency vehicles, we can resolve these problems. We must develop a system that can identify cars and categorise them as either emergency or non-emergency vehicles. In this study, the problem is solved by using CCTV footage of the traffic junction to find the emergency vehicle. Images are captured by a CCTV camera at regular intervals. After discovering each automotive, they classified it as either an emergency vehicle or a regular vehicle. Our initiative focuses on the detrimental effects that traffic congestion has on the transportation system for emergency vehicles. Critical vehicles have a tough time navigating through traffic in places like India, where the road width and length make it hard to build a dedicated lane for emergency vehicles.

## II. LITERATURE SURVEY

### *A. Arduino-UNO based traffic control system*

Vogel, I. Oremović, R. Šimić and E. Ivanjko proposes an Arduino-UNO based system that aims to reduce traffic congestion and waiting time. Through the camera, this system captures photos, which are then processed in MATLAB to remove saturation and colours and transform the image to a threshold image from which the traffic density is estimated. The simulation packages are preconfigured, and USB is used to connect MATLAB and Arduino. The Arduino determines each lane's green light time based on traffic volume and density. But there are a number of problems with this approach. It is challenging to accurately count the number of vehicles on the road since the cars frequently overlap. Additionally, diverse things hampered detection because they were also turned to black and white, making it unable to distinguish between common objects like billboards, poles, and trees with vehicles. Vehicle Detection using Image Processing. Kanungo, A. Sharma and C. Singla, makes use of a support vector machine algorithm along with image processing techniques. From live video. images in small frames are captured and the algorithm is applied. Small frames of photos are taken, and the algorithm is then used. OpenCV is used for image processing, and before SVM is used, the images are transformed to grayscale. This device may identify red light violations in addition to traffic density.

*B. Comparing Various existing methods*

Prof. Sheetal Jagtap and Ms. Saili Shinde discuss the various traffic light management approaches. This paper observes that each technique has a common architecture: choose input data, acquire traffic parameters from input data, process it, determine density, and update parameters. The first technique makes use of VANETS to gather data on each vehicle's location, which is then forwarded to the closest Intelligent Traffic Light with the assistance of installed GPS. Additionally, these ITLs will send updated statistics to adjacent automobiles. Drivers would receive information in the event of accidents so they may select an alternate route to avoid traffic. However, this technique is not feasible as its deployment is quite expensive. In the second method, infrared sensor-based microcontrollers are used, which capture the unique ID of every car using transmitter and receiver. In case of an emergency situation, vehicle's radio frequency tags can be used to identify them and let other vehicles move. This method detects red light violations. However, this technique is not flexible due to the fact that infrared sensors need to be in sight. The third method employs the fuzzy logic methodology, which employs two fuzzy logic controllers, one for signal optimization and the other for extending the green phase of a road in an intersection. Video cameras are utilised to collect input data and are placed at the incoming and outgoing lines. The controller then uses the data gathered by these sensors to make the best judgments possible and minimise the objective function. In the fourth method, fuzzy logic is used, and the system takes in the number of vehicles and the average speed of traffic flow in each direction as the input parameters. The number of vehicles and the average speed of traffic flow can be determined using sensors placed on the road. In the fifth method, photoelectric sensors are used, which are set at some distance apart, that capture data and send it to the traffic cabinet, which calculates the weight off each road and then set the traffic light accordingly. However, the maintenance cost is quite high. In the sixth method, video imaging is used to capture the data. Dynamic background subtraction and various morphological operations are performed to capture a clear image of the vehicle. Every time a new vehicle enters the area of interest, a new rectangle is drawn and vehicle count is incremented. The algorithm is easy to implement but does not handle occlusion and shadow overlapping.

*C. Smart Traffic system using ANN and fuzzy Controller*

Renjith Soman proposes a smart traffic light system using ANN and fuzzy controller. This system makes use of images captured from cameras installed at traffic site. The image is first converted to a grayscale image before further normalization. Then, segmentation is performed using sliding window technique to count the cars irrespective of size and ANN is run through the segmented image, the output of which is used in fuzzy controller to set timers for red and green light using crisp output. Results had an average error of 2% with execution time of 1.5 seconds.

*D. Simulation Environment*

According to A. Maria, a simulation is an imitation of a model based on a real-world system. A simulation model can be used to adjust many parameters and then test a system that is expensive or complex to build. Different states can be investigated using a simulation model. In comparison to the current FCTL system, a simulation environment can be constructed to show the situation using the new traffic light application. It can also use the new programme to simulate operations in various traffic conditions.

## III. METHODOLOGY

Our proposed system takes images from the CCTV cameras at traffic junctions as input for real-time critical vehicle detection and traffic density calculation using image processing and object detection. This image is initially sent to the vehicle detection algorithm. The number of vehicles in each class, such as ambulance, cars, bicycles, buses, and trucks are calculated.The traffic density is then calculated using the weightage allotted to different classes of vehicles. The signal switching algorithm uses this density, along with a few other factors including the presence of critical vehicles, to determine the green signal time for each lane. The red signal times have been modified to reflect this. In order to prevent lane starvation, the green signal time is limited to a maximum and minimum value. To demonstrate the system's effectiveness, a simulation is created.
This project can be broken down into 3 modules:

*A. Vehicle Detection Module*

This module is in charge of detecting class of vehicles and number of vehicles in the image using YOLO algorithm. More specifically, each class of vehicle is given a specified weightage based on its importance, size, and the approximate number of passengers that may travel in it, so that a realistic traffic density can be determined. The dataset for training the model will be prepared by scraping images from google and labelling them manually.

YOLO is a smart convolutional neural network (CNN) that can conduct real-time object detection. The technique divides the image into areas and predicts bounding boxes for each region using a single neural network applied to the entire image.


Fig 1A : Input Image

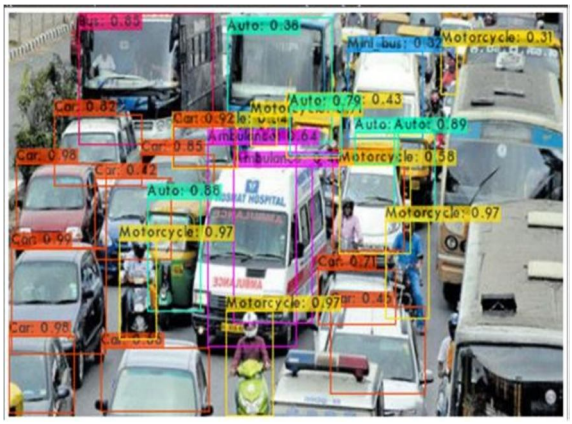


Fig 1B : Vehicles Detected


Fig 1C : Ambulance Detected

*B. Signal Switching Module*

This algorithm updates the red, green, and yellow timers of all signals. These timers are set bases on the count of vehicles of each class received from the vehicle detection module and several other factors such as the number of lanes, average speed of each class of vehicle, and so on. The findings from the vehicle detection module are used to distinguish critical vehicles from other vehicles. The signal would then be changed based on the critical vehicles' position.

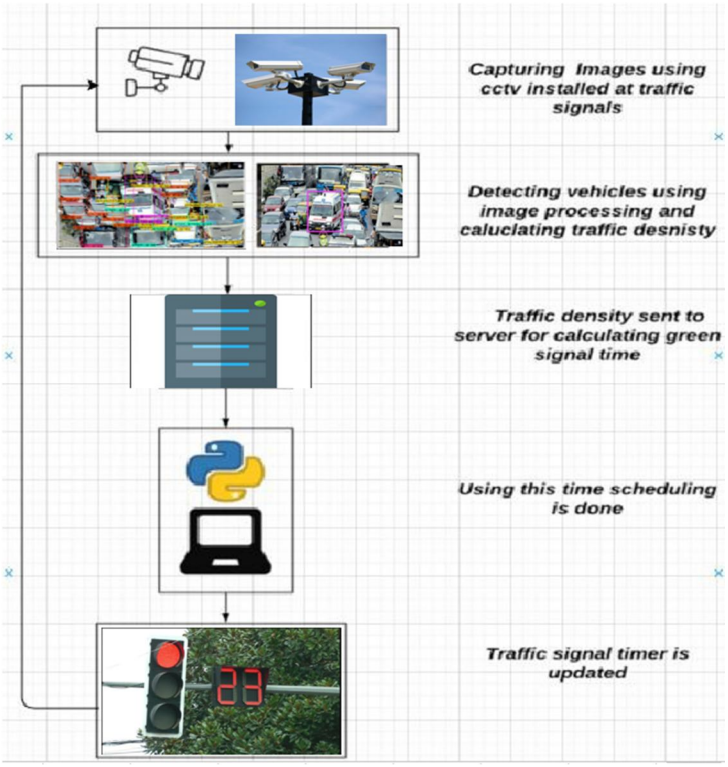



Fig 2 Signal Switching

*C. Simulation Module*

To simulate traffic lights and automobiles moving across a traffic intersection, a simulation is created from scratch using the Pygame module. Pygame is a set of crossplatform Python tools for creating video games. It offers sound and graphics libraries that can be utilised with the Python programming language. The simulation contains a four-way intersection with four traffic lights. On top of each signal is a timer that displays the amount of time until the signal changes from green to red or red to green. Cars, bicycles, buses, trucks, and rickshaws arrive from all directions. Some of the vehicles in the rightmost lane turn to cross the intersection to make the simulation more realistic.


Fig 3 Simulation output

## IV. RESULTS

*A. Evaluation of Vehicle Detection Module*

The vehicle detection module was put to the test using a variety of test photos containing various numbers of vehicles, and it was discovered that the detection accuracy was between 85 and 90%. Fig. 1(A)(B)(C) above displays some test findings. This is adequate but not ideal. Lack of a suitable dataset is the main cause of the low accuracy. This can be addressed by training the model with actual traffic camera footage, which will increase the system's accuracy.

*B. Evaluation of the Proposed Adaptive System*

In order to compare the proposed adaptive system to the current static system, 15 simulations of each system were run for 5 minutes each, with varying traffic distributions across the 4 directions. The number of cars that could pass the crossing in a certain amount of time was used to gauge performance. To put it another way, the signal's idle time—the period of time when it is green but no cars are passing the intersection—is compared. This affects how long the lines are at the other signals and how long it takes for vehicles to wait. According to the distribution [a,b,c,d], the probabilities of a vehicle being in lanes 1, 2, 3, and 4 are, respectively, a/d, (b-a)/d, (c-b)/d, and (d- c)/d. For instance, the distribution in simulation 1 has the value [300,600,800,1000], which denotes probabilities of 0.3, 0.3, 0.2, and 0.2. The results were tabulated in terms of the total number of vehicles passed as well as the number of vehicles that were passed in each lane.

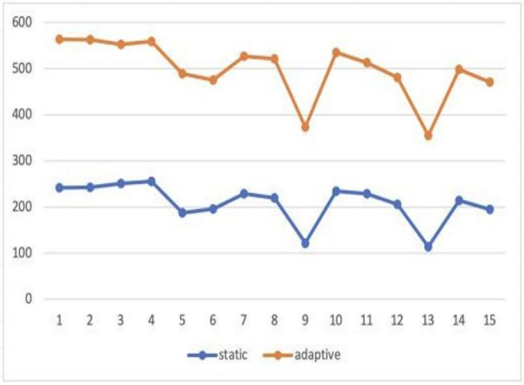
| No | Distribution | Lane 1 | Lane2 | Lane3 | Lane4 | Total |
|---|---|---|---|---|---|---|
| 1 | [300,600,800,1000] | 72 | 52 | 52 | 66 | 242 |
| 2 | [500,700,900,1000] | 113 | 50 | 48 | 32 | 243 |
| 3 | [250,500,750,1000] | 73 | 51 | 65 | 62 | 251 |
| 4 | [300,500,800,1000] | 76 | 45 | 64 | 71 | 256 |
| 5 | [700,800,900,1000] | 88 | 33 | 25 | 42 | 188 |
| 6 | [500,900,950,1000] | 95 | 71 | 16 | 14 | 196 |
| 7 | [300,600,900,1000] | 73 | 63 | 69 | 24 | 229 |
| 8 | [200,700,750,1000] | 54 | 89 | 10 | 67 | 220 |
| 9 | [940,960,980,1000] | 100 | 10 | 8 | 4 | 122 |
| 10 | [400,500,900,1000] | 81 | 29 | 88 | 37 | 235 |
| 11 | [200,400,600,1000] | 42 | 47 | 54 | 86 | 229 |
| 12 | [250,500,900,1000] | 39 | 52 | 93 | 22 | 206 |
| 13 | [850,900,950,1000] | 74 | 10 | 13 | 17 | 114 |
| 14 | [350,500,850,1000] | 49 | 46 | 69 | 50 | 214 |
| 15 | [350,700,850,1000] | 51 | 64 | 37 | 43 | 195 |

Table I. Simulation Results Of Current Static System

| No | Distribution | Lane 1 | Lane2 | Lane3 | Lane4 | Total |
|---|---|---|---|---|---|---|
| 1 | [300,600,800,1000] | 132 | 118 | 38 | 48 | 322 |
| 2 | [500,700,900,1000] | 165 | 60 | 72 | 23 | 320 |
| 3 | [250,500,750,1000] | 104 | 60 | 70 | 68 | 302 |
| 4 | [300,500,800,1000] | 99 | 56 | 79 | 69 | 303 |
| 5 | [700,800,900,1000] | 195 | 35 | 33 | 38 | 301 |
| 6 | [500,900,950,1000] | 104 | 128 | 21 | 26 | 279 |
| 7 | [300,600,900,1000] | 97 | 78 | 80 | 43 | 298 |
| 8 | [200,700,750,1000] | 66 | 118 | 29 | 88 | 301 |
| 9 | [940,960,980,1000] | 203 | 16 | 15 | 17 | 251 |
| 10 | [400,500,900,1000] | 107 | 39 | 110 | 44 | 300 |
| 11 | [200,400,600,1000] | 36 | 62 | 77 | 109 | 284 |
| 12 | [250,500,900,1000] | 62 | 85 | 111 | 17 | 275 |
| 13 | [850,900,950,1000] | 164 | 27 | 22 | 28 | 241 |
| 14 | [350,500,850,1000] | 74 | 63 | 90 | 57 | 284 |
| 15 | [350,700,850,1000] | 76 | 92 | 50 | 58 | 276 |

Table 2. Simulation Results Of Proposed Adaptive System

Graph 1 illustrates how, regardless of the distribution, the proposed adaptive system always outperforms the existing static system. Depending on how unevenly traffic is distributed among the lanes, performance will improve. The performance improves as the traffic distribution becomes more skewed.




Graph 1 : Comparison between static and adaptive

## V. CONCLUSION

By following the above-mentioned steps and performing image processing, we can be able to achieve the following results:

*1)* This Adaptive Traffic Signal Timer uses live images from the cameras at traffic junctions for traffic density calculation using YOLO(You Only Look Once) object detection and sets the signal timers accordingly.

*2)* It can detect the emergency vehicles such as Ambulance and adapt to the given situation accordingly thus reducing the traffic congestion for the critical vehicles. Thus Signal switching would reduce the amount of time it takes for critical vehicles to pass.

*3)* People can anticipate to spend substantially less time stuck in traffic if the dynamic traffic control system is implemented, and they can use the timers to turn off their engines, decreasing fuel consumption and emissions accordingly and Reducing the amount of time spent at a traffic signal can help people be more productive.

This idea can be implemented for a larger network by using algorithms to ensure safety and stability of systems. Also, the extended time can be calculated by the system itself. By keeping records of traffic patterns and using an algorithm, the timing can be chosen according to the traffic patterns.

## VI. FUTURE WORK

The project can be further expanded to include the following functionalities to improve traffic management and reduce traffic congestion:

*1)* Identification of vehicles violating traffic rules: Vehicles going through red lights can be spotted in an image or video stream by drawing a violation line and capturing the number plate if the line is crossed when the signal is red. Similarly, lane switching can be recognised. Background subtraction and image processing techniques can be used to accomplish these results.

*2)* Accident or breakdown detection: Intersections are prone to serious wrecks because numerous types of injurious collisions, such as angle and left-turn collisions, frequently occur there. As a result, precise and timely identification of accidents at junctions has enormous benefits in terms of saving property and lives while also reducing congestion and delay. This can be accomplished by identifying automobiles that remain motionless for an extended period of time in an inconvenient location, such as in the middle of the road, while excluding parked vehicles.

*3)* Synchronization of traffic signals at multiple intersections: Synchronizing traffic signals along a street can help commuters since once a vehicle enters the street, it can continue without halting.

## REFERENCES

[1] Vogel, I. Oremović, R. Šimi ć and E. Ivanjko, "Improving Traffic Light Control by Means of Fuzzy Logic," 2018 International Symposium ELMAR, Zadar, 2018, pp. 51-56, doi: 10.23919/ ELMAR.2018.8534692.

[2] A. Kanungo, A. Sharma and C. Singla, "Smart traffic lights switching and traffic density calculation using video processing," 2014 Recent Advances in Engineering and Computational Sciences (RAECS), C h a n d i g a r h , 2 0 1 4 , p p . 1 - 6 , d o i : 1 0 . 11 0 9 / RAECS.2014.6799542.

[3] Ms. Saili Shinde, Prof. Sheetal Jagtap, Vishwakarma Institute Of Technology, Intelligent traffic management system:a Review, IJIRST 2016

[4] Renjith Soman "Traffic Light Control and Violation Detection Using Image Processing"." IOSR Journal of Engineering (IOSRJEN), vol. 08, no. 4, 2018, pp. 23-27

[5] A. Maria, Introduction to modelling and simulation, in Proceed- ings of the 29th conference on Winter simulation, IEEE Computer Society, Atlanta, GA, USA, 1997, pp. 7–13.

[6] Rajeshwari S., Santhoshs Hebbar, Varaprasad Golla " Implementing Intelligent Traffic Control System for Congestion Control, Ambulance Clearance and Stolen Vehicle Detection " DOI 10.1109/ JSEN.2014.2360288, IEEE Sensors Journal

[7] Abdoos M., Mozayani N and Bazzan A.L.C., "Traffic Light Control in Non- Stationary Environments based on Multi agent Q-learning", in Proc. IEEE Conference on Intelligent Transportation Systems, pp.580- 1585, 2011.

[8] "Traffic Congestion in Bangalore-A Rising Concern", http://www.commonfloor.com/guide/trafficcongestion-in-bangalore-a-rising-concern-27238.html.

[9] I. Kosonen, "Multi-agent fuzzy signal control based on real-time simulation," vol. 11, no. 5, pp. 389– 403, 2003.

[10] Tzutalin, 'LabelImg Annotation Tool', 2015. [Online]. Available: https://github.com/tzutalin/ labelImg