SMART TRAFFIC MANAGEMENT SYSTEM

Abdul Junaid

Marium Hasan

Urwa Fatima

HABIB UNIVERSITY

KARACHI, PAKISTAN

2019

SMART TRAFFIC MANAGEMENT SYSTEM

The *Kaavish* Report
presented to the academic faculty

by

Abdul Junaid

Marium Hasan

Urwa Fatima

in partial fulfillment of the requirements for
Bachelor of Science
Computer Science
Dhanani School of Science and Engineering

Habib University

Spring 2019

SMART TRAFFIC MANAGEMENT SYSTEM

This *Kaavish* project was advised by:

Dr.Umair Azfar Khan
Faculty of Computer Science
Habib University

Approved by the Faculty of Computer Science on September 13, 2019.

If you thought that science was certain - well, that is just an error on your part.

Richard Feynman

Dedicated to those,

to whom we entrust the future

ACKNOWLEDGEMENTS

We would like to acknowledge CS faculty and our advisor for guiding us throughout the project. We are thankful to our fellow students especially Ramsha Saad, and Muhammad Ali who helped us in one way or another. We also acknowledge IT department in helping resolve issues.

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

This project aims to propose a possible solution to traffic congestion using Machine learning and video/image processing in a simulated environment.Techniques are discussed and identified in this report

KEYWORDS: Machine Learning, Image Processing, Simulation, Python, Object detection, Traffic flow, Traffic timer, Data Structures

CHAPTER 1
INTRODUCTION AND BACKGROUND

Transportation is a crucial part of a moving and growing economy. It delivers good and manpower across the city and countries. The use of public transport for mobility is not always a viable option provided the conditions and burden on it. The effective and efficient management of vehicles on road is important for sustainability of the transportation sector. The blockages on road, mismanagement at intersection etc increases the average waiting time of the commuter. The increased burden of private transports on roads is one of the causes for delays in arrival and departure times. The increase in number of vehicles is causing apparent increase in traffic congestion. Traffic congestion is faced by each and every individual in some form or the other. The problem is of individual nature and impacts the environment and society at large. The fuel consumption while waiting in long queues increases pollution. The cost is an added burden for the commuter.

1.1 Terminologies

Traffic is referred as flow or movement of motorized vehicles, non-motorized vehicles, pedestrian etc.

Traffic Congestion is a condition on any network as user increases and is characterized by slower speeds, longer trip times, and increased queuing. Traffic congestion is of two types; namely recurring traffic congestion and non-recurring traffic congestion.

Recurring traffic congestion is the congestion which happens at same place during same time of the day. There is a underlying pattern which can be studied and can be predicted. Non-recurring traffic congestion is the congestion which happens rarely. It is the congestion which happens due to emergency situations like accident, weather conditions etc.

1.2 Current state

Over the last decade number of cars have increased greatly. It is estimated that millions of cars exist on the roads of Karachi. Each day new cars are added to already existing cars. With lack of urban planning and poor transportation system, this increase adds to the existing loads of the road. This in turn increases the average waiting time of the commuter.

1.3 Problem statement

With rapid increase in number of vehicles, traffic congestion is becoming more apparent. Poor signal timer is one of the contributing factor. It increases average waiting time and waste fuel resources. This project aims to address traffic congestion and intervene in the situation to propose a solution to manage traffic congestion.

1.4 Solution

We intend to propose a solution which changes the traffic timer dynamically according to the situation of the lanes. The process involves acquiring the video of traffic scene, process the video, detect objects in the scene and alter the traffic timer accordingly. This will give more time to congested lane to clear out first and then action for other lanes is taken. This will reduce the average waiting time of the traveller.

The problem of traffic congestion have been addressed in past using different approaches. In this regard, a review paper by Nellore and Hancke [**ref1**] different approaches have been identified and discussed with their advantages and disadvantages. Ultrasonic sensor, RFID, magnetic sensor, inductive loops and satellite imaging. Few require boring under he road, few require expensive technology and for most installation and maintenance is not cost effective.

The video/image processing technique require installing camera on the traffic sig-

nal and capturing videos to be processed. These videos are then sent to workstation where it is processed and analyzed to take relevant decision. The analysis is transformed into some form of traffic data.

The scope of the project employing video processing technique is quite large spanning from acquiring videos from accurate position to processing videos and analyzing it to modelling the system to work on average waiting time to installing the device on traffic signal. Therefore, the scope this project is brought to simulate the real traffic scene and then perform the desired actions of detecting traffic congestion through some measure and taking relevant actions to reduce the waiting time of the commuter.

CHAPTER 2
DESIGN DETAILS

The problem we are trying to address of managing traffic congestion is a wicked problem. A wicked problem is one which is complex, interconnected to wide variety of problems, there is no definite solution, solutions can have with unforeseen consequences, involves many stakeholders and have incomplete knowledge about the problem.

Design thinking is one viable way to intervene in the problem and come up with a solution. Design thinking was followed for this project, the five stages of the process are shown in the figure 2.1;

1. Empathy: The audience we are targeting is each and every individual who travel on roads of Karachi and get stuck in traffic which increases their travelling time.

2. Define: Explore different ways to adjust traffic signal timer. Traffic timer is one possible option which can be modified according to the current situation of the traffic and give ore time to congested lane to clear out first.

3. Ideate: Find out ways to detect and objectify the current congestion situation. Object detection of traffic scene acquired through cameras can help in this process.

4. Prototype: Come up with ways to ease the process of object detection and shaping it into some form of traffic data. A decision of changing the timing of signal timer is then taken.

5. Test: Test the object detection model with customize objects to include local transport option available in Karachi.

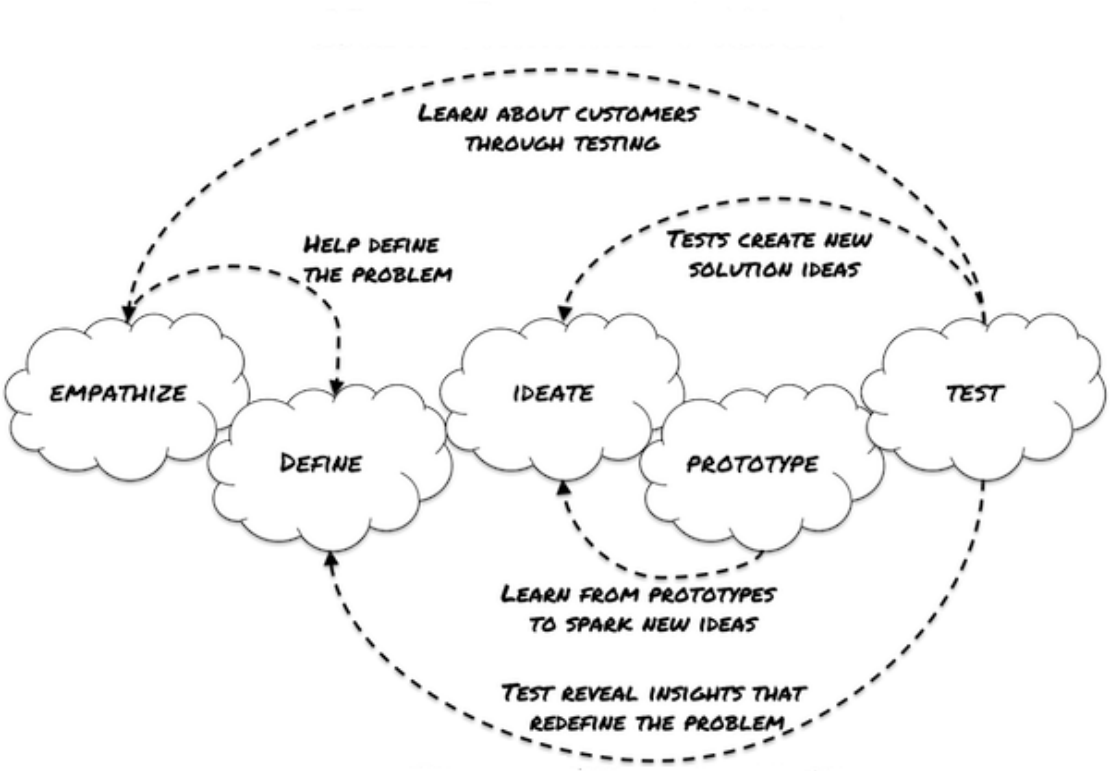


Figure 2.1: Design thinking process (Source: interaction-design.org)

2.1 Beginning

The project began by identifying the features for the user. The software methodology used for this project was SCRUM. Features were identified during the first meeting with the project advisor and user stories were developed. feature backlog was maintained throughout the process of developing this project. Features from the product backlog was taken into each of the four sprints.

2.2 Project requirements

Project required a working system with a GPU setup. Unfortunately GPU could not be arranged and provided in this duration of project. Hence, most of the work was done using CPU.

2.3 Main components

The entire the project was divided into three parts namely object detection, modelling traffic flow and creating a simulation of the proposed solution. Each component is discussed in detail below;

2.3.1 Object detection

Object detection is a process of finding instances of real-world object in images and videos. Objects includes but not limited to faces, pedestrian, vehicles and trees.There are numerous techniques for achieving the task of detecting objects in images and videos through automation. The conventional techniques are based on feature extraction from the images.

Feature are the informative regions of image which may be edges, pixel intensity values, corners, contours etc. The object detection technique based on these features creates a meaningful definition of the object by using a combination of extracted features and attempts to find similar object from the test image(s).

These techniques proved to be insufficient to fully ripe the diverse information and its underlying interactions in the images, therefore, are used for limited scale and problems. With increase in computation power of computers and boom in deep learning, new avenues of research and application have been identified. Deep learning requires a lot of data to learn the interaction, invariant of scale and orientation, between different features of an image. This helps obtain robust model for detecting objects from an image.

The deep learning deploys deep models of neural network for acquiring knowledge from the data and giving off desired results. A neural network mimics a natural human brain functionality and is composed of neuron which are computational units in each layer of an architecture. These neurons are connected to neurons in preceding layers, take input from them,perform some mathematical function on the input and feed this as output which is taken up by the neurons in layers after it. Convolution Neural Network (CNN) is the representative model for the deep

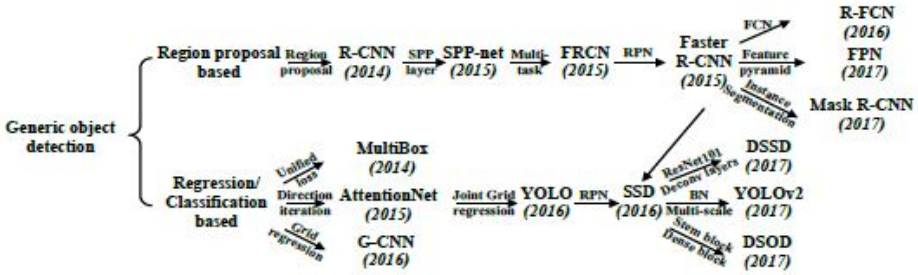


Figure 2.2: Two frameworks of object detection

learning.

The deep learning based object detection frameworks have been broadly divided into two classes, which are region-proposal based framework and regression /classification based framework [**ref2**] as shown in the figure 2.2.

Region-Proposal based framework: This framework first identifies the region of interest from the images and then scan these regions to extract features through neural network. It then performs classification and localization of object in an image. The frameworks which follow this methodology are Recurrent-CNN (R-CNN), Fast R-CNN, Faster R-CNN and Region Proposal Network . Each individual phase of this frame work is trained in a separate pipeline. Consequently, this takes greater amount of time and serves as bottleneck for real-time application.

Regression/Classification based framework: It is a one-step framework based on global regression, achieving bounding box coordinates and class probabilities from the image pixels. Architectures lying under this framework have CNN as base model and perform different task across the architecture of the neural network in one step. This in effect reduces the time complexity. You only look once (YOLO) and Single Shot MultiBox Detector (SSD) are common frameworks. YOLO have been subjected to improvements by incorporating different strategies. As per the requirement of the project we have worked with YOLOv3 (this will be discussed in later section).

2.3.2 Mathematical Model

One of the component of project was to understand the flow of traffic and model it using mathematical equations. Modelling the traffic flow has been subject of research for many years. As a result of which different models have emerged. Most of them understand the traffic as an analogous system to some natural system. This give rise to groups of models to study traffic flow. Macroscopic, Microscopic, cellular automaton and queue theory based models are common and discussed in various literature.

In initial models view the traffic flow as analogous to hydrodynamic fluid theory and kinetic theory of gas. Macroscopic model all vehicles are taken as one unit in traffic scene, their aggregated flow is studied. The model is based on hydrodynamic theory of fluids in one-dimension. Examples include capacity analysis equation, speed-flow-density relationship captured by Greenshields and Shock wave analysis. The microscopic model studies the vehicular level interaction during traffic and each vehicle behaviour is considered. examples include car-following theory and time-space diagram. Cellular automaton is also one of the model but it fails to captures certain important interactions taking place among vehicles in traffic.

One of the excessively used model which depicts the long queuing situation of traffic congestion is based on queuing theory. Queue theory have been used in many management system such as inventory system,production management, banking,medical services etc. The theory was used by Heidemann in 1990s for modelling traffic flow. It is used for predicting the average waiting time in queue, length of queue and number of objects in queue.

Since then queue theory have been deployed to model traffic flow considering different parameters such as the pattern of arrival of vehicles in the system, pattern of departure of vehicles in the system, number service channel, width of road, velocity of vehicles etc. It is used for predicting the average waiting time in queue, length of queue and number of objects in queue. The fundamental concept underlying this theory is explained in the block diagram(figure 2.3)

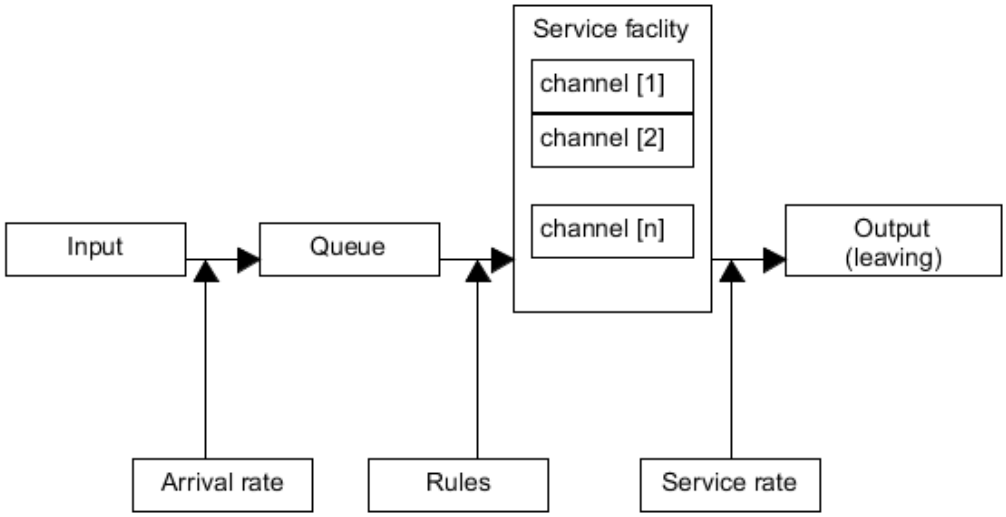


Figure 2.3: Queue theory fundamental idea

There are two classes of traffic flow, first is interrupted traffic flow which encapsulates for the traffic flow affected by the external factor such as signal. Second is uninterrupted traffic flow n which traffic flow steadily without any obstruction like on highways. As part of this project, interrupted traffic flow is considered and worked upon.

2.3.3 Simulation

Simulation is an imitation of operation of a system or modelling a natural or human system. It provides a visualization of the functioning of the system, enable us to derive insights from their functioning and get knowledge about it. It captures and show the effects of different actions and interactions taking place in the system. Therefore, it can be use to study and understand the behaviour of the system before deploying the model in a system which possess a lot of risk and is not fully understood. In our case, traffic congestion being a wicked problem there are potential risks in directly deploying the proposed solution.

Hence, we worked on developing simulation of the actual solution. This enabled us in understanding the functioning of the system and the possible consequences the

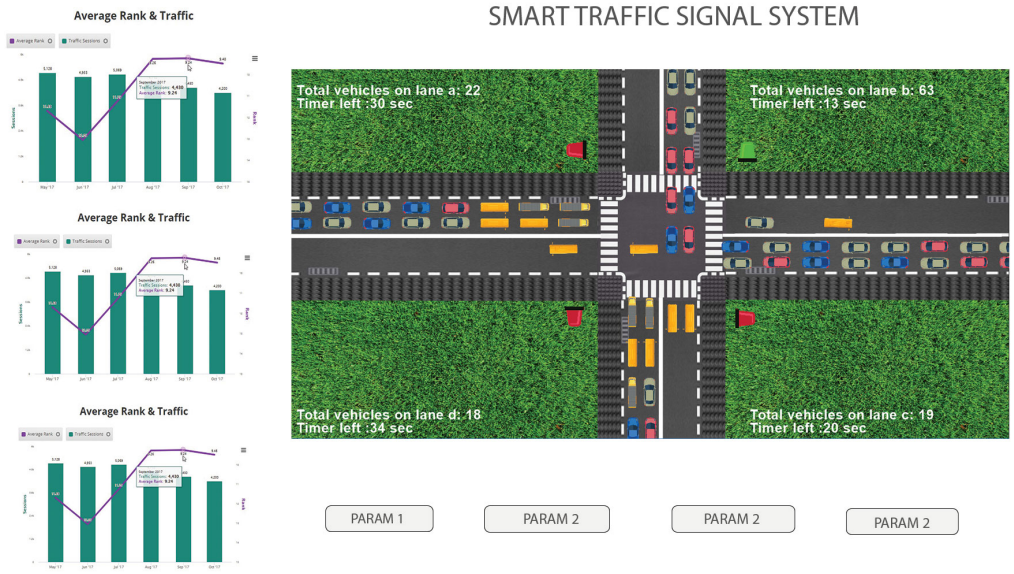


Figure 2.4: UI for Simulation

system will undergo. We provided a visualization of single intersection with four lanes as a subject of study. The features are correlation with real traffic scene, arrival rate, time for clearing a lane, sequence in which signals are turned green, departure rate etc.

```
import operator
import time
import math
import pyglet
import random
from pyglet.window import Window

#load the required images
def load(filename):
    return pyglet.sprite.Sprite(pyglet.image.load("resources/"+
    filename))

screen = Window(1366, 700,
                resizable=True,
                caption="Traffic Simulation",
                config=pyglet.gl.Config(double_buffer=True),
```

```
                vsync=False)

background = load("background.png")
background.scale = (min(background.height, 768) /max(background.
    height, 768) )  #setting the scale to resize the background image
    to perfect size of thr disply screen

#initiating list with the names of the possible images of cars
car_images = ["red_car.png", "green_car.png", "blue_car.png"]
truck_images =  ["truck.png"]
bike_images = ["blue_bike.png","yellow_bike.png"]
bus_images = ["bus.png"]

t0 = time.time()
#class for all type of vehicles
class Vehicle:
    def __init__(self, vehicle_type,lane,line):
        global car_images #use the globaly defined list of cars
        global  truck_images # use the globaly defined truck images
        self.vehicle_type = vehicle_type #initiate the car type
        self.lane1 = lane #set the lane of vehicle
        if  vehicle_type == "truck":
            self.car = load(random.choice(truck_images))
        elif vehicle_type == "car":
            self.car = load(random.choice(car_images))
        elif vehicle_type == "bike":
            self.car = load(random.choice(bike_images))
        elif vehicle_type == "bus":
            self.car = load(random.choice(bus_images))
        if vehicle_type in ["bus", "truck", "car"]:
            self.car.scale_x = 0.125 #scale the image width by the
    factor
            self.car.scale_y = 0.125 #scale the image height by the
    factor
        else:
```

```
47            self.car.scale_x = 0.2778 #scale the image of bike
48            self.car.scale_y = 0.2542 #scale the image of the bike
49
50        #setting the initial postion of vehicles for lane a
51        if lane.lane == "a":
52            self.car.x = 0 - (self.lane1.col*75) #setting the X
    cordinate of the car, 0.125*(sprite height=480) = 60 in which we
    are adding 15
53            self.car.y = 768-(315+(self.lane1.row*30)) #setting the Y
     cordinate of the car, 0.125*(sprite width=240) = 30
54            if self.lane1.row == 0: # the lane of cars from the top
    of the screen
55                self.lane2 = random.choice(["b", "c"])
56            else:
57                self.lane2 = random.choice(["c", "d"])
58            self.car.rotation = 90   # the clockwise rotation of
    sprite for lane a
59            self.lane1.col += 1 # changing y position
60
61        #condtion for lane "b"
62        elif lane.lane == "b":
63            self.car.rotation = 180   # the clockwise rotation of
    sprite for lane b
64            self.car.x = 740 + (self.lane1.row*33)      #setting the
    X cordinate of the car to draw the next sprite to be 33 pixels
    away from the first car sprite
65            self.car.y = 768 + (self.lane1.col*75) #setting the Y
    cordinate of the car, to start appearing behind 75 pixels of the
    background pixels cordinate
66            if self.lane1.row == 0:    # If the cars are in the first
     lane that is from the right of the screen then these cars can go
    to following lanes
67                self.lane2 = random.choice(["a", "d"])
68            else:
69                self.lane2 = random.choice(["d", "c"])
```

```
70            self.lane1.col += 1
71
72        #condtion for lane "c"
73        elif lane.lane == "c":
74            self.car.x = 1366 + (self.lane1.col*75) #setting the X
    cordinate of the car, 0.125*(sprite height=480) = 60 +15
75            self.car.y = 768 -(420+(self.lane1.row*30)) #setting the
    Y cordinate of the car, 0.125*(sprite width=240) = 30
76            if self.lane1.row == 0:
77                self.lane2 = random.choice(["b", "a"])
78            else:
79                self.lane2 = random.choice(["a", "d"])
80            self.car.rotation = -90   # the clockwise rotation of
    sprite for lane a
81            self.lane1.col += 1 # changing y position
82
83        #condtion for lane "d"
84        elif lane.lane == "d":
85            self.car.x = 635 + (self.lane1.row*33)      # setting the
     x  coordinate of the cr sprite to appear from the 635 pixel of
    the  background image with next 33 pixels away
86            self.car.y = 0 - (self.lane1.col*75)  # setting y
    coordinate of the car sprite to be 75 pixels behind the visible
    background image
87            if self.lane1.row == 0:   #
88                self.lane2 = random.choice(["a", "b"])
89            else:
90                self.lane2 = random.choice(["b", "c"])
91            self.lane1.col += 1
92
93    def move(self):
94
95        #condition when traffic is coming from lane "a"
96        if self.lane1.lane == "a":
```

```
97          if self.lane1.light == "red" and self.car.x < 510 and
    self.lane1.moving:
98              self.car.x += 3
99              if self.car.x >= 510:
100                 self.lane1.moving = False
101                 self.lane1.reached = True
102         elif self.lane1.light == "green" or (self.lane1.light ==
    "red" and self.car.x >= 530):
103             #condition when destination lane is "b"
104             if self.lane2 == "b":
105                 if self.car.y >= 453:   #the intersection point
    of lane1 of car (1080 - 494 = 586)
106                     if self.car.x == 668:
107                         self.car.y += 3
108                     elif self.car.y == 453 and self.car.x < 668:
109                         if self.car.x < 628.36:
110                             self.car.x += 3
111                         else:
112                             self.car.rotation -= 4.6
113                             self.car.x += 2
114
115             #condition when destination lane is "c"
116             elif self.lane2 == "c":
117                 self.car.x += 3
118
119             #condtion when destination lane is "d"
120             elif self.lane2 == "d":
121                 if self.car.y <= 423:   #the intersection point
    of lane1 of car (1080 - 494 = 586)
122                     if self.car.x >= 732:
123                         self.car.y -= 3
124                     elif self.car.y <= 423 and self.car.x < 732:
125                         if self.car.x < 607.87:
126                             self.car.x += 3
127                         else:
```

```
                                self.car.x += 3.5
                                self.car.rotation += 2.5
                                self.car.y -= 0.9
        #condition when traffic is coming from lane "b"
        elif  self.lane1.lane == "b":
            if self.lane1.light == "red" and self.car.y > 550 and
self.lane1.moving:
                self.car.y -= 3
                if self.car.y <= 555:
                    self.lane1.moving = False
                    self.lane1.reached = True
            elif self.lane1.light =="green" or (self.lane1.light == "
red" and self.car.y < 550):
                #condition when destination lane is "b"
                if self.lane2 == "a":
                    if self.car.x <= 740:  # 740 is for setting the
display sprite postion
                        if self.car.y <= 345: # 768 - 444
                            self.car.x -= 3
                        elif self.car.x <= 740 and self.car.y > 345:
   # 768 - 423 = 345
                            if self.car.y > 423:
                                self.car.y -= 3
                            else:
                                self.car.rotation += 3.46
                                self.car.y -= 3
                                self.car.x -= 1


                #condtion when destination lane is "c"
                elif self.lane2 == "c":
                    if self.car.x >= 773:
                        if self.car.y <= 425:
                            self.car.x += 3

```

```
159                     elif self.car.x >= 773 and self.car.y > 425:
    # 768 - 423 = 345
160                         if self.car.y > 482:
161                             self.car.y -= 3
162                         else:
163                             self.car.y -= 2
164                             self.car.rotation -= 3.17
165                             self.car.x += 0.9
166
167             #condition when destination lane is "d"
168             elif self.lane2 == "d":
169                 self.car.y -= 3
170
171         #condition when traffic is coming from lane "c"
172     elif self.lane1.lane == "c":
173         if self.lane1.light == "red" and self.car.x > 869 and
    self.lane1.moving:
174             self.car.x -= 3
175             if self.car.x <= 878:
176                 self.lane1.moving = False
177                 self.lane1.reached = True
178         elif self.lane1.light == "green" or (self.lane1.light ==
    "red" and self.car.x < 875):
179             #condtion when destination lane in "d"
180             if self.lane2 == "d":
181                 if self.car.y <= 318:   #the intersection point
    of lane1 of car (1080 - 494 = 586)
182                     if self.car.x <= 745:
183                         self.car.y -= 3
184                     elif self.car.y == 318 and self.car.x > 745:
185                         if self.car.x > 800:
186                             self.car.x -= 3
187                         else:
188                             self.car.rotation -= 3.27
189                             self.car.x -= 2
```

```
190
191                 #condtion when destination lane is "a"
192                 elif self.lane2 == "a":
193                     self.car.x -= 3
194
195                 #condtion when destination lane is "b"
196                 elif self.lane2 == "b":
197                     if self.car.y >= 298:   #the intersection point
    of lane1 of car (1080 - 494 = 586)
198                         if self.car.x <= 668:
199                             self.car.y += 3
200                         elif self.car.y >= 298 and self.car.x > 668:
201                             if self.car.x > 792:
202                                 self.car.x -= 3
203                             else:
204                                 self.car.x -= 3.5
205                                 self.car.rotation += 2.5
206                                 self.car.y += 0.9
207         #condition when traffic is coming from lane "d"
208         elif self.lane1.lane == "d":
209             if self.lane1.light == "red" and self.car.y < 210 and
    self.lane1.moving:
210                 self.car.y += 3
211                 if self.car.y >= 210:
212                     self.lane1.moving = False
213                     self.lane1.reached = True
214             elif self.lane1.light =="green" or (self.lane1.light == "
    red" and self.car.y > 215):
215                 #condition when destination lane is "b"
216                 if self.lane2 == "a":
217                     if self.car.x <= 635:
218                         if self.car.y >= 345:
219                             self.car.x -= 3
220                         elif self.car.x == 635 and self.car.y <= 345:
    # 768 - 423 = 345
```

```
                            if self.car.y < 288:
                                self.car.y += 3
                            else:
                                self.car.rotation -= 4.7
                                self.car.y += 3


                #condtion when destination lane is "c"
                elif self.lane2 == "c":
                    if self.car.x >= 668:
                        if self.car.y >= 425:
                            self.car.x += 3

                        elif self.car.x >= 668 and self.car.y < 425:
    # 768 - 423 = 345
                            if self.car.y < 347:
                                self.car.y += 3
                            else:
                                self.car.y += 2
                                self.car.rotation += 2.3
                                self.car.x += 0.9

                #condition when destination lane is "d"
                elif self.lane2 == "b":
                    self.car.y += 3
        return

    def draw(self):
        self.car.draw()

class lane_segment:
    def __init__(self, lane):
        self.light = "red" # to keep trck of signl setting by defut
   to red
```

```
253         self.timer_logic = False # flg to check the logic for timer to be pre-timed or our logic
254         self.reached = False # to check hs the crs reched the zebr crossing when signl is red
255 
256         #loading sprites for signal lights and storing them in a list
257         self.lights = []
258         self.lights.append(load("green_light.png"))
259         self.lights.append(load("red_light.png"))
260 
261         scale = 0.03 #factor to scale down the sprite of light to appropriate size
262         self.lane = lane # it is a string tht keep track of the name of the lane
263         self.moving = True # to mke the crs rech zebr crossing while the signl is red
264         self.col = 1 # for positioing of the car
265         self.row = 0
266         self.count = 0
267         self.t0 = 0
268         self.t1 = 0
269 
270         #setting the x-axix, y-axis, and rotation according the lane
271         if lane == "a":
272             if self.timer_logic:
273                 self.t0 = 10
274                 self.t1 = 10
275             self.l_x = 10
276             self.l_y = 650
277             x = 563.475
278             y = 244.622
279             rotation = -90
280         elif lane == "b":
281             if self.timer_logic:
282                 self.t0 = 20
```

```
283                 self.t1 = 10
284             self.l_x = 872
285             self.l_y = 650
286             x = 844.50
287             y = 244.622
288             rotation = 0
289         elif lane == "c":
290             if self.timer_logic:
291                 self.t0 = 30
292                 self.t1 = 10
293             self.l_x = 872
294             self.l_y = 35
295             x = 844.50
296             y = 516.26
297             rotation = 90
298         elif lane == "d":
299             if self.timer_logic:
300                 self.t0 = 40
301                 self.t1 = 10
302             self.l_x = 10
303             self.l_y = 35
304             x = 563.475
305             y = 516.26
306             rotation = 180
307         for i in self.lights:
308                 i.x = x
309                 i.y = 768-y
310                 i.rotation = rotation
311                 i.scale = scale
312         self.label = pyglet.text.Label("", font_name = "Castellar",
    font_size = 20, x = self.l_x, y = self.l_y, color =
    (255,255,255,255), bold = True)
313         self.label_time = pyglet.text.Label("", font_name = "
    Castellar", font_size = 20, x = self.l_x, y = self.l_y-25, color =
     (255,255,255,255), bold = True)
```

```
314
315     def draw(self):
316         global check
317         global count
318         if self.light == "red":
319             self.lights[1].draw()
320         else:
321             self.lights[0].draw()
322         if check:
323             if self.t0 == self.t1:
324                 self.light = "green"
325         if self.t0 > 0 and check:
326             if count == 60:
327                 count = 0
328                 self.t0 = self.t0 - 1
329             else:
330                 count+=1
331         else:
332             self.light = "red"
333         self.label.text = "Total vehicles on lane "+self.lane+": "+
    str(self.count)
334         self.label_time.text = "Timer left :"+str(self.t0)+" sec"
335         self.label.draw()
336         self.label_time.draw()
337
338 count = 0
339 check = False
340 def on_draw():
341     global lanes
342     global sarak
343     global check
344     check = True
345     screen.clear()
346     background.draw()
347     for i in lanes:
```

```
348         if lanes[i].reached == False:
349             check = False
350         lanes[i].draw()
351     for i in sarak:
352       for j in sarak[i]:
353           if (j.car.x > 1366 or j.car.y > 768 or j.car.y < 0) and j.
    lane1.lane == "a":
354               sarak[i].remove(j)
355               lanes[i].count -= 1
356           elif (j.car.x < 0 or j.car.x > 1366 or j.car.y < 0) and j.
    lane1.lane == "b":
357               sarak[i].remove(j)
358               lanes[i].count -= 1
359           elif (j.car.x < 0 or j.car.y > 768 or j.car.y < 0) and j.
    lane1.lane == "c":
360               sarak[i].remove(j)
361               lanes[i].count -= 1
362           elif (j.car.x < 0 or j.car.x > 1366 or j.car.y > 768) and j
    .lane1.lane == "d":
363               sarak[i].remove(j)
364               lanes[i].count -= 1
365           else:
366               j.move()
367               j.draw()
368
369 sarak = {} #is a dictionary that holds the list of vehicles for the
    specific lane
370 lanes = {"a":0,"b":0,"c":0,"d":0}
371 for i in lanes:
372     lanes[i] = lane_segment(i) #
373
374 def main(line1,line2,lane):
375     global lanes
376     global sarak
```

```
377     a = [] #list to keep record of all the vehicles on specific
        segment
378
379     ##populating the segment
380     #populating line1 of the segment
381     for i in line1:
382         b = Vehicle(i,lanes[lane],"1")
383         a.append(b)
384     #populating line2 of the segment
385     lanes[lane].row = 1 # changing the line
386     lanes[lane].col = 1 # setting X to default
387     for i in line2:
388         b = Vehicle(i,lanes[lane],"2")
389         a.append(b)
390     sarak[lane] = a
391
392 def updated(dt):
393     on_draw()
394
395 pyglet.clock.schedule_interval(updated, 1 / 60)
396
397 f = open("count1.txt")
398 l = ["a","b","c","d"]
399 j =  -1
400 cot = {}
401 for i in f.readlines():
402     if i.split(" ")[0] == "file":
403         a = []
404         j += 1
405         no = 0
406     else:
407         c = i.strip("\n")
408         c = c.split("=")[1].split(",")
409         no += len(c)
410         a.append(c)
```

```
411     if len(a) >=2:
412         lanes[l[j]].count = no
413         if no >= 37:
414             no = 15
415         elif no < 37 and no >= 20:
416             no = 10
417         else:
418             no = 5
419         if lanes[l[j]].timer_logic != True:
420             lanes[l[j]].t1 = no
421         cot[l[j]] = no
422         main(a[0],a[1],l[j])
423 
424 cot = dict(sorted(cot.items(), key = operator.itemgetter(1), reverse
    = True))
425 
426 total_timer = 0
427 for i in cot:
428     if lanes[i].timer_logic != True:
429         total_timer += cot[i]
430         lanes[i].t0 = total_timer
431 
432 # Launch the application
433 pyglet.app.run()
```

CHAPTER 3

RESULTS

These are the results of the models that we tested. We can see that HOG and Haar Cascade are not good enough. HOG (Figure 3.1 is detecting same car twice, and Haar Cascade (Figure 3.2 detects soil as a car. The best one is YOLO (Figure 3.3, but the problem with YOLO is that it is pre-trained on 80 classes, most of those classes were useless and it was slowing down the detection. There were few reasons to retrain the model, we had to remove those extra classes, add new classes, and make model to run for local vehicles. We added class of "rickshaw" which was not part of the original model.

Figure 3.1: HOG

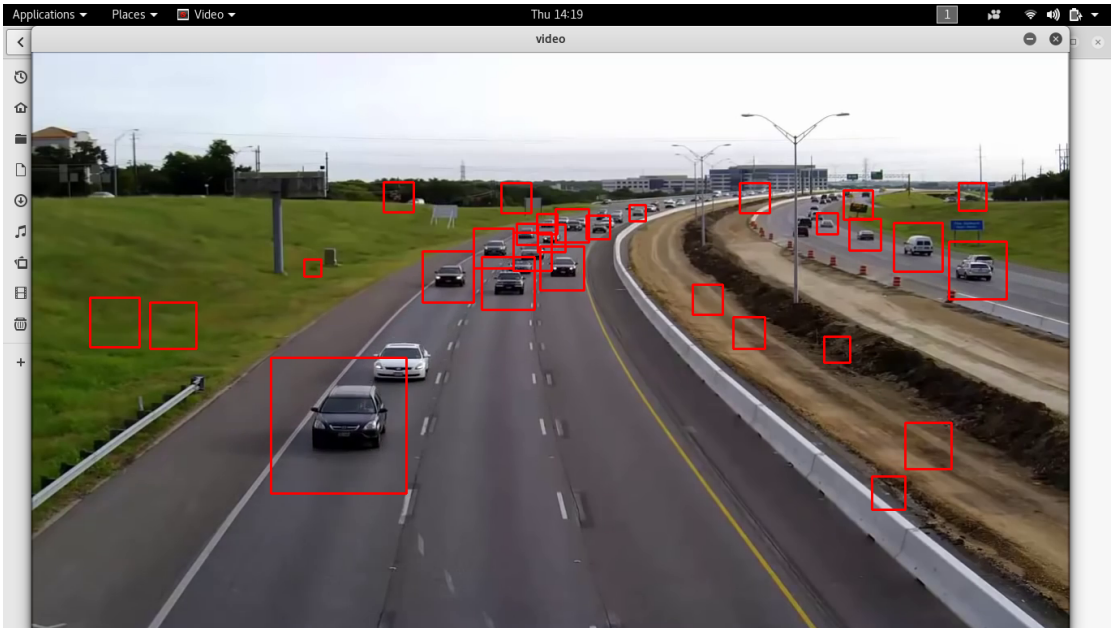


Figure 3.2: Haar Cascade

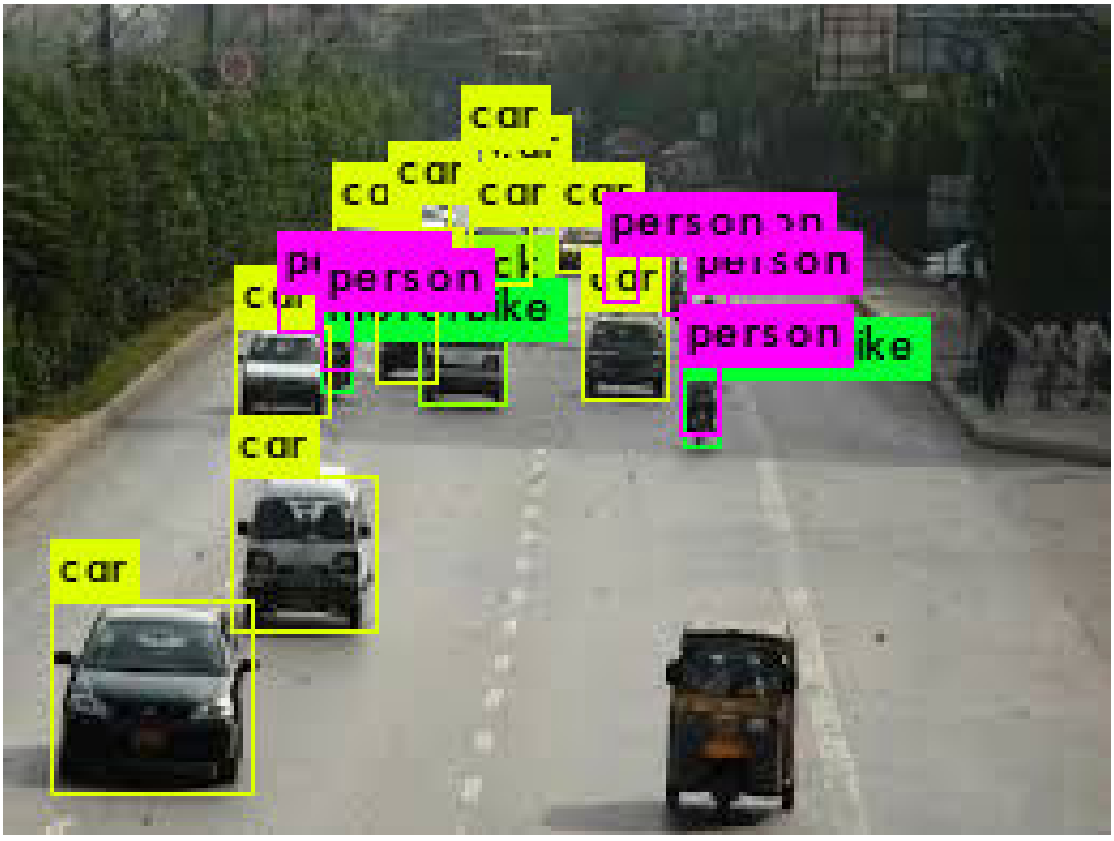


Figure 3.3: YOLO Pretrained

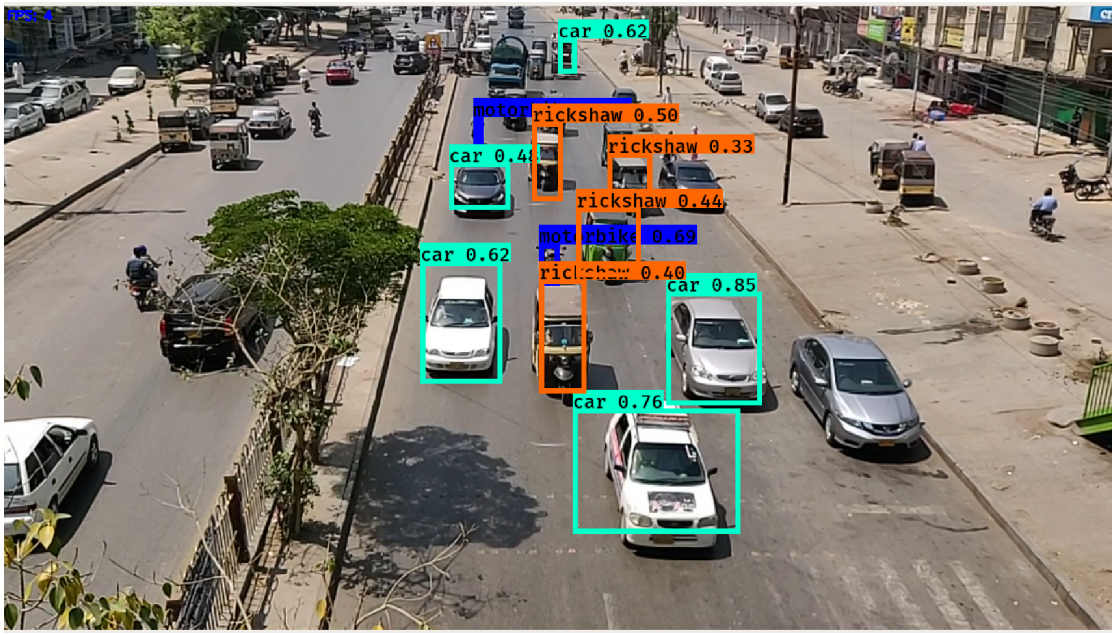


Figure 3.4: YOLO re-trained

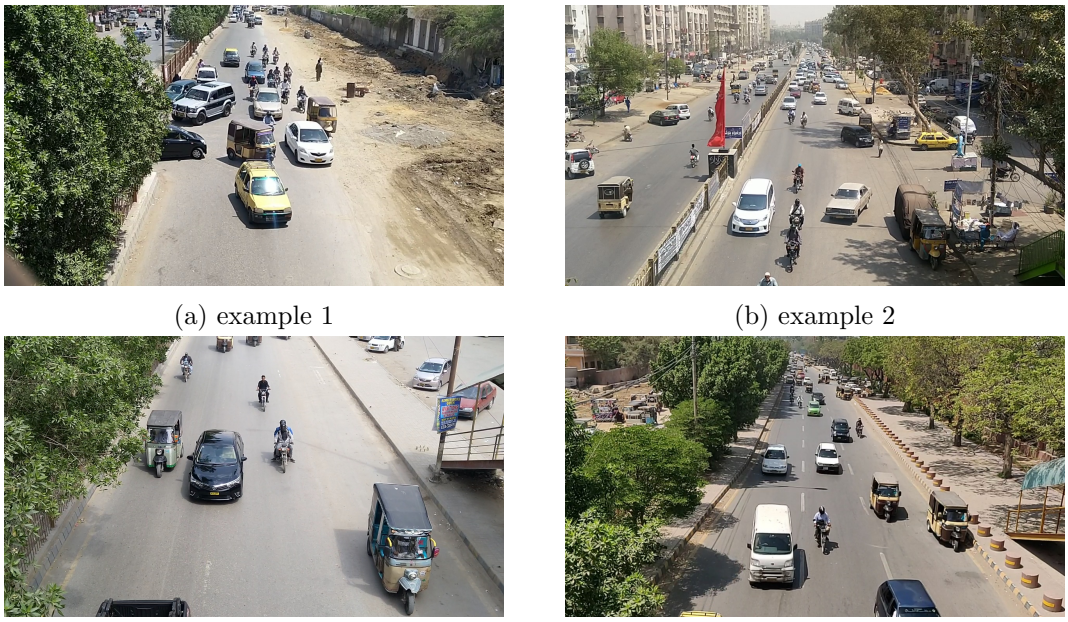
We trained the new model on a very small dataset (120 images), to test if YOLO will work. In Figure 3.4 we can see that even after training on such a small datset, it was able to detect "rickshaw" and only work on incoming traffic while ignoring other lane.

CHAPTER 4

DISCUSSION

One part of the project was based on object detection. To achieve object detection on traffic scene required us to explore different object detection techniques. Conventional techniques failed to detect objects accurately on provided images. Thus we moved to work with Neural Networks. We aimed at training YOLOv3 for customize objects.

The process of training a neural network began with providing with labelled dataset. This can be broken down into stages of collecting images and then labelling it with classes. Dataset was collected in form of video which were recorded on roads. Frames were extracted from each video using python code and each image was annotated using open source tools.



(a) example 1 (b) example 2

(c) example 3 (d) example 4

Figure 4.1: Example of some frames

```
# Program To Read video
# and Extract Frames
import os
import cv2
import math

# Function to extract frames
def FrameCapture(path, file):

    # Path to video file
    vidObj = cv2.VideoCapture(path+"\\"+file)
##    print(vidObj.get(5))
    frate = vidObj.get(5)
    # Used as counter variable
    count = 0

    # checks whether frames were extracted
    success = 1

    while success:

        # vidObj object calls read
        # function extract frames
        success, image = vidObj.read()
        fid = vidObj.get(1)

        if not os.path.exists("/home/fyp/Desktop\FYP/Frames/{0}".
    format(file.split(".")[0])):
            os.makedirs("/home/fyp/Desktop/FYP/Frames/{0}".format(
    file.split(".")[0]))

        if (fid % math.floor(frate) == 0):

        # Saves the frames with frame-count
```

```
            cv2.imwrite("/home/fyp/Desktop/FYP/Frames/{0}/frame{1}.
    jpg".format(file.split(".")[0],str(count)), image)

            count += 1

# Driver Code
if __name__ == '__main__':
    path = "/home/fyp/Desktop/FYP/Videos"
    files = os.listdir(path)
    for file in files:
        # Calling the function
        FrameCapture(path, file)
    print('done')
```

We worked with labelImg which allowed us to have annotation in VOC xml format. Then we used the code below to convert those xml files to required txt files. These annotation will be fed to the model for training over the desired classes.

```
import xml.etree.ElementTree as ET
from os import getcwd

classes = ["truck", "rickshaw", "bus", "car", "motorbike"]


def convert_annotation(image_id, list_file):
    in_file = open('test/%s.xml'%(image_id))
    tree=ET.parse(in_file)
    root = tree.getroot()

    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        cls = obj.find('name').text
        if cls not in classes or int(difficult)==1:
            continue
        cls_id = classes.index(cls)
```

```
        xmlbox = obj.find('bndbox')
        b = (int(xmlbox.find('xmin').text), int(xmlbox.find('ymin').
    text), int(xmlbox.find('xmax').text), int(xmlbox.find('ymax').text
    ))
        list_file.write(" " + ",".join([str(a) for a in b]) + ',' +
    str(cls_id))

wd = getcwd()

#for year, image_set in sets:
image_ids = open('list.txt').read().strip().split()
list_file = open('train.txt', 'w')
for image_id in image_ids:
 list_file.write('test/%s.jpg'%(image_id))
 convert_annotation(image_id, list_file)
 list_file.write('\n')
list_file.close()
```

These images were divided into training and test datasets.The dataset was augmented, as well, and labelled for increasing the robustness and invariance in object detection of custom objects.

We tested images using pre-trained model of YOLOv3. It was successful in detecting most of the objects correctly while coarse objects were not detected fully. Also the results were indicative of the potential of detecting objects with high accuracy.

To understand the implications of the traffic flow, simulation was created to analyze the traffic through the video feed that will be collected from the cameras placed over the signal pole.The position of the cameras are very important for understanding the region of interest to make the decision of changing the green signal of the lanes. To calculate the position of camera, we are taking the average pole height, an average length and width of a car, average width of a road, cameras coverage angle and distance.

There are multiple scenarios in which the simulation defines the flow of traffic

and the decisions of the signal timer. First, we can randomly generate traffic flow for the lanes. In each lane a random number of vehicles are generated that move within the static time frame.Then the traffic flows some define distribution. There is also traffic generation using the correlation from real traffic video.

CHAPTER 5

ANALYTICAL MODELS AND ANALYSIS OF ALGORITHMS

5.1 Object detection

YOLO is a one-step object detection model, implying it classifies and detects in one architecture of CNN. Bounding boxes and associated class probabilities are predicted in one evaluation. YOLO divides the image in S by S grid cell, predicts number of bounding boxes for each cell and confidence for each box and conditional class probability.

YOLO uses 24 convolutional layers followed by 2 fully connected layers. It takes an input images, resizes it and pass it as entire image through the network. The classes and bounding boxes boxes are predicted in last layers of the network. Then, each convolution layer works on sapling the image, batch normalization and non-maximum suppression is done. It uses sum of squared error as loss function during training and logistic regression is used as objectness score. Binary entropy loss is used for predicting classes.

However, YOLO fails to detect sparse details and there are localization error for background. We used YOLOv3 which is a more dense network with 53 convolutional layers. This helps in more details to be detected.

Pre-trained models are not customize for the local transports and hence dataset was extended. Then the model for trained to get desired results.

5.2 Traffic flow

The model for traffic flow is such that it studies the relation between rate of arrival and rate of departure with average waiting time of the commuter. Most parameter are taken arbitrarily and hence changing these can aid in improving the average waiting time. This in turn, will mean the physical factors to be modified which is

quite not possible.

There are few constraints which are in hands of the driver like speed. The results for average waiting time here is for signalized intersections only. In other cases parameters will change.

CHAPTER 6

MATHEMATICAL MODELS

As per the requirement of the project, modelling the traffic flow for getting the average waiting time have to be studied and incorporated with the simulation. Firstly, a model suggested by the Transport authority was studied and explored. It was based on principles of dynamic theory. The approach was to implement the equations and tweak the values to get relevant results. One of the relevant model was base don queue theory for interrupted traffic flow . Parameters under consideration are;

- average arrival rate λ
- average departure rate with no delay μo
- average departure rate μ_1
- number of vehicles (n)
- rate of occurrence of delay(f)
- rate of disappearance of delay (r)

The state of the system and number of vehicles can be represented as (Y(t), X(t)). Y(t) can be delay state (D) or normal state (N). Probability for each state is represented as $\pi_{(n,N)}$ and as $\pi_{(n,D)}$. The model predicts the number of cars and average waiting time of the vehicle. The parameters under consideration are the width of the road, number of car, service rate, delay function, travel speed at full length and length of each phase. This model uses Websters equation for the delay function. This equation is further worked for difference number of service channels.

We intend to visualize the results graphically and provide the results in simulation.

This will provide the understanding of how waiting time in affected with traffic flow.

CHAPTER 7

CONCLUSION

The purpose of this project is to simulate the traffic behavior, once we introduce new model on it. Simulating the model is better than, implementing a real model on our traffic systems. It is both cost and time effective. As of now, simulation works on fixed timer and object counting approach. It improves the system significantly if the approach of object count is considered and lanes are prioritized accordingly.

Appendices

APPENDIX A
EXPERIMENTAL EQUIPMENT

To complete this project a working computer with GPU was needed. Data was collected using the mobile phone rather than camera mounted on some height.

APPENDIX B

DATA COLLECTION

Drone was ordered to record the videos, but due to some problems we did not get the drone. We worked around that and recorded videos using cellphones, from various locations in Karachi and a dataset was created from it.

APPENDIX C

SOURCE CODE

Link to the Source codes:

- Simulation: `https://github.com/BluesFYP/FYP_STMS/blob/master/Simulation1.py`
- Frame: `https://github.com/BluesFYP/FYP_STMS/blob/master/frame.py`
- VOC: `https://github.com/BluesFYP/FYP_STMS/blob/master/voc_annotation.py`
- Keras: `https://github.com/qqwweee/keras-yolo3`

VITA

Abdul Junaid

EDUCATION

- Bachelors of Computer Science (May 2019), Habib University, Karachi.
- A levels (June 2015), Sarghodian Spirit Trust Public School, Tando-Allahyar.
- O levels (June 2013), Sarghodian Spirit Trust Public School, Tando-Allahyar.

Marium Hasan

EDUCATION

- Bachelors of Computer Science (May 2019), Habib University, Karachi.
- HSSC Pre-Engineering (June 2015), Aga Khan Higher Secondary School, Karachi.
- SSC (June 2013), SMS Aga Khan School, Karachi.

Urwa Fatima

EDUCATION

- Bachelors of Computer Science (May 2019), Habib University, Karachi.
- HSSC Pre-Engineering (June 2014), Khatoon-e-Pakistan, Karachi.
- SSC (June 2012), Al-Murtaza School, Karachi.