



Faculty of Informatics and Computer Science  
Information System/Computer Networks/Software Engineering/  
Computer Science

Real-Time Traffic Control System by Using Object Detection  
*Algorithms*

**By: Abdulrhman Zakaria**

Supervised By

**Supervisor Name**

Dr. Walid Hussein




**June 2022**








## Abstract

Overcrowding traffic became a serious problem in Egypt with many buses, vans, and cars in the street. According to statistics, the capital Cairo hosts more than 19 million inhabitants, and it is expected to be increased to 24 million by 2027. [1] Hardly within the peak hours, reaching important meetings on time can't be done, as the traveler should expect at least double the normal time for a trip. The average speed on major streets is decreased to half, the normally expected speed should be from (60 to 80 km/h) it is reduced to (15 to 20km/h), and some local roads in central Cairo and downtown are even worse. Sometimes making it faster to make short trips on foot. Traffic congestion doesn't only waste time, but it also has a huge impact on the economy which includes wasted fuel, health impacts because of the poor air quality, and accidents. Egypt wastes more than 8 billion USD / year in traffic congestion which is higher than 4% of the country's GDP. [2] Solving traffic congestion by using the fixed cycle traffic lights doesn't solve the problem in all situations, and in some cases, a policeman is replacing the traffic lights controlling the movement instead of the traffic lights, he sees the availability and situation of the road and decides the period of each direction. From here the idea of self-adaptive smart traffic control by the neural network due to its high accuracy and speed which is once trained the prediction is fast and accurate. Which monitors the availability of the road with a camera instead of eyes and processes the road condition by machine learning instead of the human brain. This solution is more available than manual controlling because it can work the whole day, is less effective than automatic controlling as it is not based on the constant value loaded in the timer, and is less expensive than the electronic sensors, as it doesn't depend on expensive sensors. Applying this solution will decrease traffic, air pollution, waiting time, the number of accidents, and fuel consumption. [3] The proposed system has the ability to detect different vehicles types, and in varied light conditions as it can detect the vehicles in the morning sunlight or the night with the good light condition or poor light conditions then the detection results are passed to the scheduling algorithm to determine the green light & the red light for the road lanes based on the vehicle count and its weight, then the red and green timer counts down until the yellow timer count down which repeat the detection operation in a loop. This traffic control system was able to achieve the real-time traffic situation detection and take the most efficient decision to manage the intersected traffic lanes without waiting for a fixed time which always leads to over waiting for the cars when the cars are less than the suggested time or

some cars have to wait for 2 turns in the traffic because the car count was higher than the suggested time.

## Turnitin Report

My Submissions				
Part 1				
Title	Start Date	Due Date	Post Date	Marks Available
Graduation-projects SE - Part 1	5 Jun 2022 - 15:18	18 Jun 2022 - 23:18	30 Jun 2022 - 15:18	100
Refresh Submissions				
Submission Title	Turnitin Paper ID	Submitted	Similarity	Grade
View Digital Receipt <a href="#">abdulrhman186883</a>	1856758989	14/06/22, 17:19	39% 	-- Submit Paper   --

### Match Overview

# 39%

<

>

1	Submitted to British Un... Student Paper	37%	>
2	Submitted to Babes-Bo... Student Paper	1%	>
3	I. Gogul, V. Sathiesh Ku... Publication	1%	>
4	Submitted to Australia... Student Paper	1%	>
5	Submitted to Kakatiya I... Student Paper	<1%	>
6	Submitted to Institute ... Student Paper	<1%	>
7	Submitted to Massey U... Student Paper	<1%	>
8	Kaiming He, Georgia G... Publication	<1%	>
9	<a href="#">scholar.lib.vt.edu</a> Internet Source	<1%	>
10	<a href="#">scholar.uwindsor.ca</a> Internet Source	<1%	>

## Table of Contents

### Table of Contents

Abstract .....	2
Turnitin Report.....	3
Table of Contents.....	4
List of Figures.....	6
1 Introduction .....	10
1.1 Overview .....	10
1.2 Problem Statement .....	10
1.3 Scope and Objectives.....	11
1.4 Report Organization (Structure) .....	11
1.5 Work Methodology .....	12
2 Related Work (State-of-The-Art) .....	14
2.1 Background.....	14
2.2 Literature Survey .....	14
2.3 Analysis of the Related Work.....	32
3 Proposed solution.....	33

3.1	Solution Methodology .....	33
3.2	Functional/ Non-functional Requirements.....	34
3.3	Design / Simulation setup.....	35
4	Implementation .....	35
5	Testing and evaluation .....	44
5.1	Testing.....	44
5.2	Evaluation .....	53
6	Results and Discussions .....	56
7	Conclusions and Future Work.....	59
7.1	Summary .....	59
7.2	Future Work.....	60
	References .....	61
	Appendix I.....	65
	Appendix II.....	65

## List of Figures

Similarly, you can automatically generate a list of figures from paragraphs of style *Figure & Tables*. To update this after revisions, right-click in the table and choose *Update Field* for the entire table.

Figure 1 naive Bayes Theorem .....	14
Figure 2 Branches of Artificial Intelligence by AI Summary .....	15
Figure 3 ANN Model Structure .....	16
Figure 4 ANN Model .....	16
Figure 5 ANN Model code .....	17
Figure 6 sample code for ANN .....	17
Figure 7 phases for CNN model .....	18
Figure 8 example of max & average pooling .....	19
Figure 9 Full structure of CNN pt.1 .....	19
Figure 10 Full structure of CNN pt.2 .....	19
Figure 11 CNN Code implementation .....	20
Figure 12 Different detection of CNN for the same object in different angel and light conditions.....	21
Figure 13 difficulties of CNN detection pt.2.....	21
Figure 14 R-CNN example.....	22
Figure 15 R-CNN architecture .....	22
Figure 16 R-CNN architecture pt.2.....	23
Figure 17 fast R-CNN architecture .....	23
Figure 18 models comparisons .....	24
Figure 19 faster R-CNN.....	24
Figure 20 time test between models .....	25
Figure 21 Yolo model example .....	26
Figure 22 Yolo model architecture .....	26
Figure 23 fast R-CNN vs YOLO .....	26
Figure 24 Object Detection vs instance segmentation .....	27
Figure 25 masked RCNN architecture .....	27
Figure 26 masked RCNN model architecture in detail. ....	28
Figure 27 ROI pooling stage .....	28
Figure 28 YOLO architecture .....	29
Figure 29 Yolo handmade gride.....	29
Figure 30 Yolo matrix for each box.....	30
Figure 31 two roads intersect each other in a one-way direction .....	31
Figure 32 design flow.....	35
Figure 33 import files from the GitHub .....	36
Figure 34 add class labels.....	36
Figure 35 masked RCNN function .....	37
Figure 36 adds vehicles weights.....	38
Figure 37 green time calculations .....	38
Figure 38 timers.....	39

Figure 39 implementing threads and calling the function .....	40
Figure 40 run YOLO model .....	40
Figure 41 YOLO function .....	41
Figure 42 import detection values to the YAML file.....	41
Figure 43 Yaml file data .....	41
Figure 44 YOLO output function the results from YAML file .....	42
Figure 45 implementing threads and calling functions .....	42
Figure 46 masked RCNN with video files.....	43
Figure 47 the change of the code when it detects from video .....	43
Figure 48 first lane 1st iteration MASKED RCNN Results.....	44
Figure 49 detection details and execution time .....	45
Figure 50 scheduling algorithm.....	45
Figure 51 2nd lane 1st iteration masked RCNN results .....	46
Figure 52 1st lane 2nd MAsked RCNN iteration result.....	47
Figure 53 the last iteration scheduling system results .....	48
Figure 54 reading video frames Masked RCNN .....	49
Figure 55 video to frames by Masked RCNN video function.....	49
Figure 56 frame detection by Masked RCNN.....	50
Figure 57 detection results .....	51
Figure 58 1st lane 1st iteration YOLO detection .....	51
Figure 59 schedule algorithm results for 1st lane .....	52
Figure 60 detection summary including time.....	52
Figure 61 2nd lane 1st iteration Yolo detection .....	53
Figure 62 1st lane 2nd iteration YOLO detection .....	53
Figure 63 1st lane Masked RCNN detection .....	54
Figure 64 1st lane Yolo detection.....	54
Figure 65 2nd lane masked Rcnm detection .....	55
Figure 66 2nd lane YOLO detection.....	55
Figure 67 1st lane 2nd iteration Masked RCNN .....	55
Figure 68 1st lane 2nd iteration yolo.....	55





## List of Tables

## 1 Introduction

### 1.1 Overview

The project aims to solve the traffic congestion in Egypt through the self-adaptive traffic light which works in a synchronized smart method without any human involvement or any attached sensor. All that is needed is a CCTV camera and embedded computer which it can capture frames per second, and this computer process this image frame and detect the number of cars passing through the lanes by neural networks techniques as the convolutional neural network technique so it will be able to make its own decision by python code implementation, such that it will be able to switch between the traffic lights according to the car types as each car type has own weight then after retrieving these values the green and red timer start to count down until reaching to the yellow timer again which is responsible for repeating the detection process.

### 1.2 Problem Statement

Over twenty-one, million-person live in Cairo by 2021, because of urbanization, and industrialization the use of private and public vehicles accelerated, becoming essential in the Egyptian citizen's daily life. Citizens use main roads to reach their work, schools, and universities, travel from one government to the other government, and do appointments. For the students, they must attend on time to catch their lectures, while for employees they must reach their workplace on time to not have a deduction from their salaries, and people don't want to waste more than the double time of their trip to travel from one place to another. Traffic congestion is a serious problem because it affects the environment and has a huge impact on global warming because vehicles tend to consume more fuels when their car is stuck in the traffic, as their

trip duration sometimes can exceed twice the normal trip duration. Moreover, traffic issues increase the accidents rate, in which people tend more to break driving rules and be always in hurry to reach their work, or appointment on time. This problem costs unnecessary money increases death rates and prevents the country from elevating to a better position.

### **1.3 Scope and Objectives**

This project focuses on solving traffic congestion without any use of expensive sensors or installed hardware, fixed cycle traffic lights, or by changing the architecture of the road, and without the need for human power in it. This projects mainly focus to solve mentioned problem through Egypt in which it studies the Egyptian driver's behaviors, Egyptian road structure, and vehicle types that are used in Egypt. This project covers the solution to crowded traffic using artificial intelligence especially neural networks which can classify and detect objects giving them a prediction score, and then pass this data to a scheduling algorithm based on what the camera capture in each second. The Project's goal is to solve vehicle overcrowding during rush hours, produce the optimal green signal time based on the current traffic, remove the unwanted delays, reducing congestion and waiting time which will reduce the number of accidents.

### **1.4 Report Organization (Structure)**

- 1- Introducing the problem and the objectives and then defining the work methodology to solve this problem with a clear work plan.
- 2- Related work literature survey and analysis.
- 3- Proposed solution description and methodology.

- 4- Implementation and code.
- 5- Test cases and evaluation for this code.
- 6- Conclusion including a summary and any future changes.

### **1.5 Work Methodology**

- 1- Collecting information about the factors of traffic congestion in Egypt.
- 2- Searching for similar countries facing the same problem as India and record congestions factors.
- 3- Identify the applied solutions to decrease the traffic jams & go deeper to understand why these solutions are not effective.
- 4- Getting an alternative solution that is practical with low cost and doesn't need expensive hardware to be applied.
- 5- Doing more research about the self-adaptive traffic light which is based on the neural network.
- 6- Knowing the differences between the neural network techniques such as object recognition, class segmentation, object detection, and object segmentation to choose the fittest technique for the solution.
- 7- Comparing models' drawbacks helped to filter between all models and choose what will be used in the solution.
- 8- Does more research on how to increase the model accuracy through the test cases?
- 9- Implements a simple project has the model and records the results.
- 10- Add some features to this model as counting algorithms to detect and filter the non-required objects or the objects with a low percentage of certainty.
- 11- Capture different videos and pictures for more than one place day and night to see the result accuracy and how the model will perform in these test cases.

- 12- Creating a scheduling algorithm to fit the situation.
- 13- Implements more complex projects which use the test cases created and try the scheduling algorithms and record the results and accuracy.
- 14- Fix these projects' bugs & errors then record the response time, and accuracy, and compare the before and after applying this model to see if it is an effective solution or not.

## 2 Related Work (State-of-The-Art)

### 2.1 Background

There are many classification algorithms in machine learning. Based on this fact it was hard to choose which algorithm is the most suitable for the project. The first classification model is by using the Naïve Bayes theorem, and it is usually used for text classification or recommendation systems, secondly, there is an artificial neural network that is used for classification but not optimized for the images. The convolutional neural network is a very good solution, but it has some drawbacks because of this some models fixed these problems as the R-CNN and to increase its speed the fast R-CNN was invented but it was not the best for real-time so faster R-CNN is invented then they added to it the instance segmentation to be Masked R-CNN which has high accuracy and can detect the small objects better than YOLO model. While You look only once the model is faster due to its simple design. [4]

### 2.2 Literature Survey

The first common approach is known as the Naïve Bayes theorem, according to some research it has many advantages as it works quickly in real-time and saves a lot of time and memory, can work on a large dataset can solve multi-class prediction problems, and work with text categorization, spam filters, recommendation systems and classify the objects. [5]

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

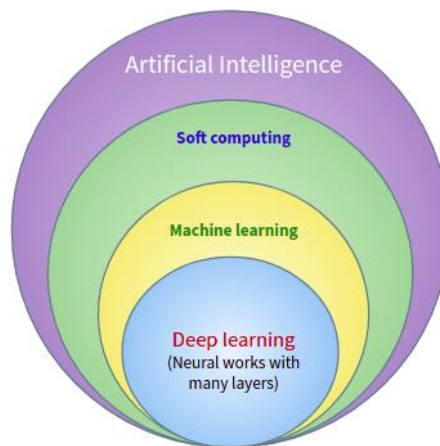
We can then substitute Eq. 4.2 into Eq. 4.1 to get Eq. 4.3:

$$\hat{c} = \operatorname{argmax}_{c \in C} P(c|d) = \operatorname{argmax}_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

*Figure 1 naive Bayes Theorem*

This algorithm was not enough for the project because of several reasons, as it assumes that all predictors are independent, and it happens rarely in real life. Assigns zero probability to any categorical variable whose category in the test data set and not available in the training dataset and its calculation can be wrong in some cases so this should be considered. [6]

Then after this moving to machine learning deep artificial neural networks are found which work for unsupervised learning consisting of nodes where each node does simple computation which is similar to the human brain's neuron and can take accurate decisions by itself but in its initial stages will need human supervision. [7]



*Figure 2 Branches of Artificial Intelligence by AI Summary*

Artificial neural networks (ANN) utilize weights and an activation function which shows how the weighted sum of the input data is changed into a result from a node or nodes in one layer of the network.

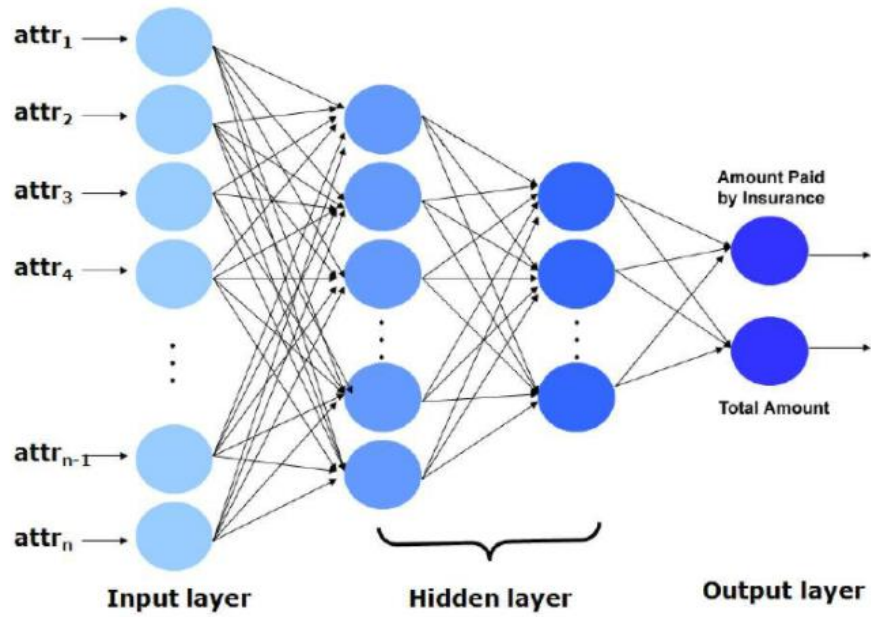


Figure 3 ANN Model Structure

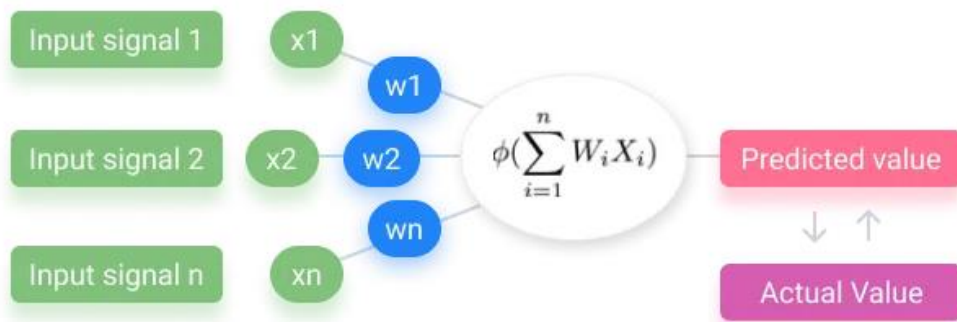
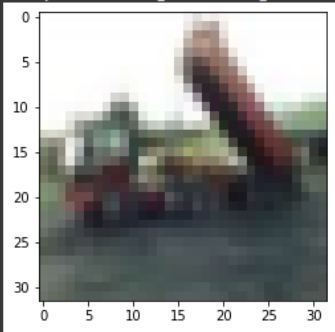


Figure 4 ANN Model

ANN was not the best solution due to several reasons such as the object's data points which must be provided in the form of length and width. Furthermore, the utilization of ANN in image classification will become difficult because this model must convert the 2-dimensional image to 1-dimension vectors which will increase storage and processing capability and thus will increase the number of trainable parameters. Therefore, the result will make it not efficient and expensive. [8] It is shown that the idea of the neural network is suitable for the project, but ANN is not the right model.



```
<matplotlib.image.AxesImage at 0x7f6596b531d0>

y_train is a 2D array, for our classification having 1D array is good enough. so we will convert this to now 1D array

[ ] y_train = y_train.reshape(-1,)
    y_train[:5]

array([6, 9, 9, 4, 1], dtype=uint8)

[ ]
    y_test = y_test.reshape(-1,)
```

Figure 5 ANN Model code

```
building ANN model

[ ] ann = models.Sequential([
    layers.Flatten(input_shape=(32,32,3)),
    layers.Dense(3000, activation='relu'),
    layers.Dense(1000, activation='relu'),
    layers.Dense(10, activation='softmax')
])

ann.compile(optimizer='SGD',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)

Epoch 1/5
1563/1563 [=====] - 86s 55ms/step - loss: 1.8165 - accuracy: 0.3522
Epoch 2/5
1563/1563 [=====] - 86s 55ms/step - loss: 1.6237 - accuracy: 0.4268
Epoch 3/5
1563/1563 [=====] - 87s 55ms/step - loss: 1.5445 - accuracy: 0.4553
Epoch 4/5
1563/1563 [=====] - 87s 56ms/step - loss: 1.4818 - accuracy: 0.4780
Epoch 5/5
1563/1563 [=====] - 87s 56ms/step - loss: 1.4351 - accuracy: 0.4944
<keras.callbacks.History at 0x7f37462be310>
```

Figure 6 sample code for ANN

The first alternative is the Convolutional neural network (CNN) which consists of layers that take in the image as input, perform a mathematical calculation with a non-linear activation function, and predict the class or the label probabilities at the output. the CNN adds multiple layers to images and uses filtration to analyze image data inputs. CNN automatically detects the important features without human presence or supervision. The first layer always detects both

horizontal and vertical edges, while the second filter is to identify more complicated features like objects, faces, bodies, etc. these layers output a set of decimal confidence scores values between 0 to 1 to give the possibility that the input image contains the objects which are in the ConvNet. In other words, CNN layers are not connected to all the other neurons but are connected only to a tiny region of neurons. For example, the first layer might detect the corners and edges (lowest level features). The following layer might detect the middle-level features like textures and shapes while the last level is responsible for the structure of the object (the car or the bus) and will be detected by higher layers in the network. [9]

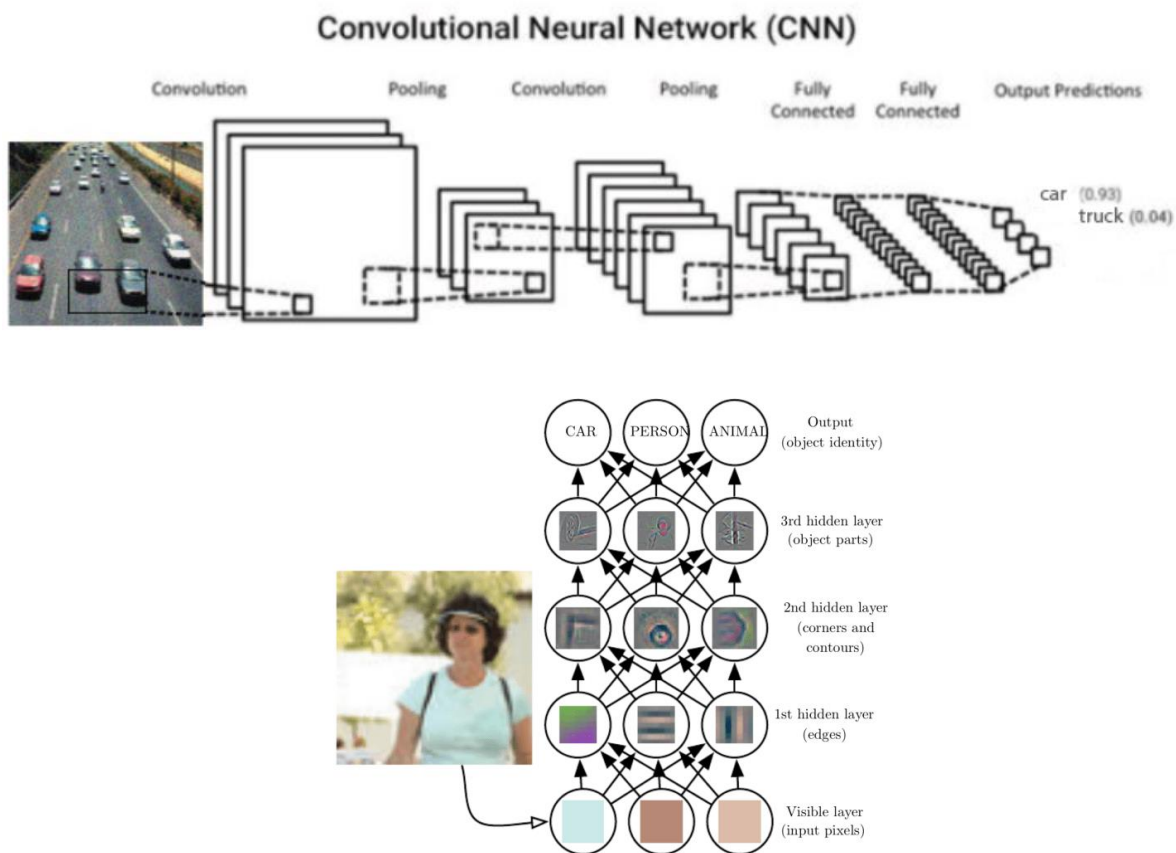


Figure 7 phases for CNN model

Finally, add optimization to the process to decrease the power required to process the data by lowering the source dimensions this optimization process is named pooling which has 2 types of max and average pooling methods. [10]

CNN model consists of the convolutional layer extracting the image as a feature map with filters and kernels and adding to it a pooling layer which aid to downsample and summarizes the appearance of features of the map and adding to this it connects every neuron in one layer to every

neuron in another layer. Adding all of those together makes the neural network model able to identify and recognize things but for this model, it is optimal to detect only one object in the image in more complex cases mask R-CNN can work for those situations.

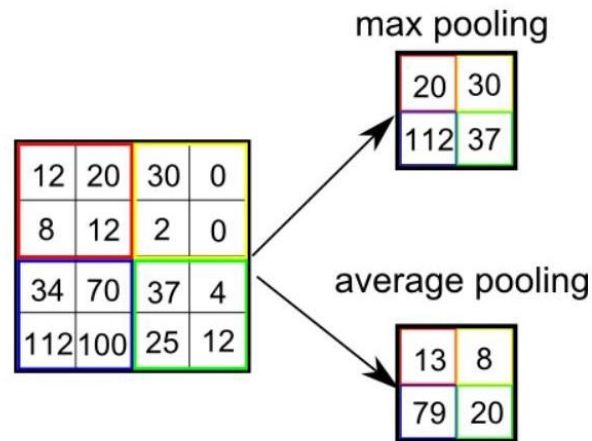


Figure 8 example of max & average pooling

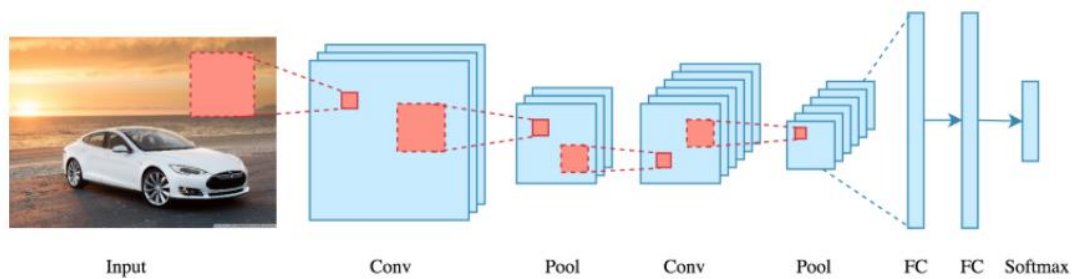


Figure 9 Full structure of CNN pt.1

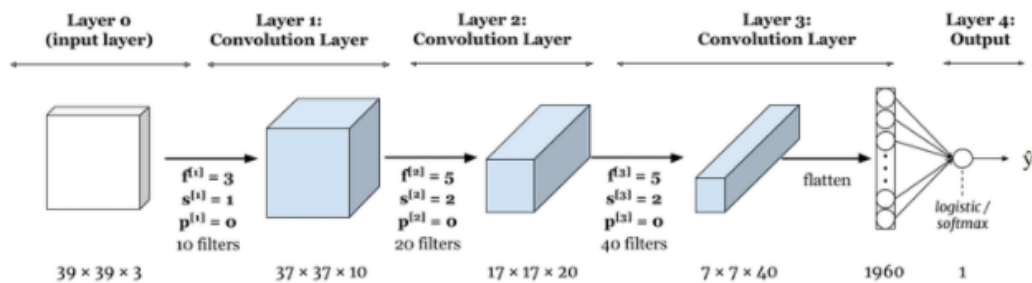


Figure 10 Full structure of CNN pt.2

```

by using cnn to improve the accuracy

[ ] cnn = models.Sequential([
    #cnn #detect image features ReLU function over other activation functions
    #is that it does not activate all the neurons at the same time.
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    # dense
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')

])

[ ] cnn.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

[ ] cnn.fit(X_train, y_train, epochs=10)

Epoch 1/10
1563/1563 [=====] - 54s 35ms/step - loss: 1.4801 - accuracy: 0.4673
Epoch 2/10
1563/1563 [=====] - 54s 34ms/step - loss: 1.1306 - accuracy: 0.6027
Epoch 3/10
1563/1563 [=====] - 54s 34ms/step - loss: 1.0039 - accuracy: 0.6505
Epoch 4/10
1563/1563 [=====] - 53s 34ms/step - loss: 0.9260 - accuracy: 0.6793
Epoch 5/10
1563/1563 [=====] - 53s 34ms/step - loss: 0.8713 - accuracy: 0.6975
Epoch 6/10
1563/1563 [=====] - 53s 34ms/step - loss: 0.8228 - accuracy: 0.7132
Epoch 7/10
1563/1563 [=====] - 53s 34ms/step - loss: 0.7830 - accuracy: 0.7286
Epoch 8/10
1563/1563 [=====] - 53s 34ms/step - loss: 0.7521 - accuracy: 0.7381
Epoch 9/10
1563/1563 [=====] - 53s 34ms/step - loss: 0.7163 - accuracy: 0.7504
Epoch 10/10

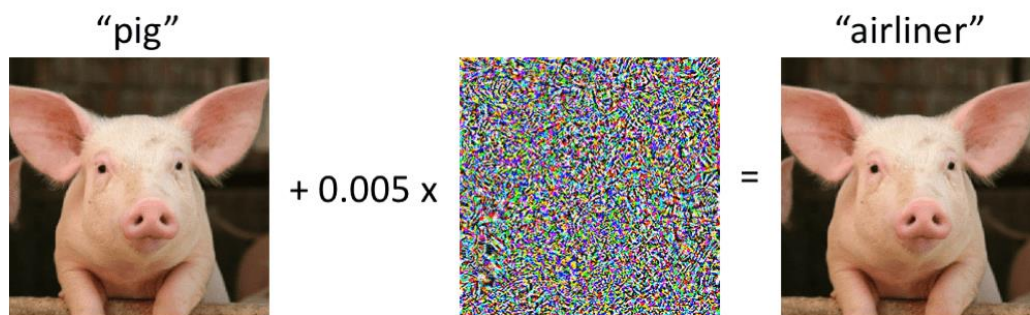
```

*Figure 11 CNN Code implementation*

CNN model was what is needed for the project, but because of some disadvantages it was hard to be chosen as it is very slow and cannot be applied in real-time due to an operation as max-pool, it consists of several layers then the training process takes a lot of time and needs good GPU, Moreover, ConvNet needs a large dataset to process and train the neural network and Convnet finds difficulty to recognize the same object in different light condition, different position or different backgrounds, in addition, there are some Adversarial problems as the image will be a completely different image if some noise is added to the image. [11]



*Figure 12 Different detection of CNN for the same object in different angel and light conditions*



*Figure 13 difficulties of CNN detection pt.2*

The above were the drawbacks that were taken into consideration in other versions of the convolutional neural network as the R-CNN which is regionally based and used for Object detection by creating squares isolated from each other. As its idea is to do a selective search to select the required object bounding it. This region of interest passes through the CNN to get the output features. Which passes to the classifier to classify the objects presented under a region of interest. [12]



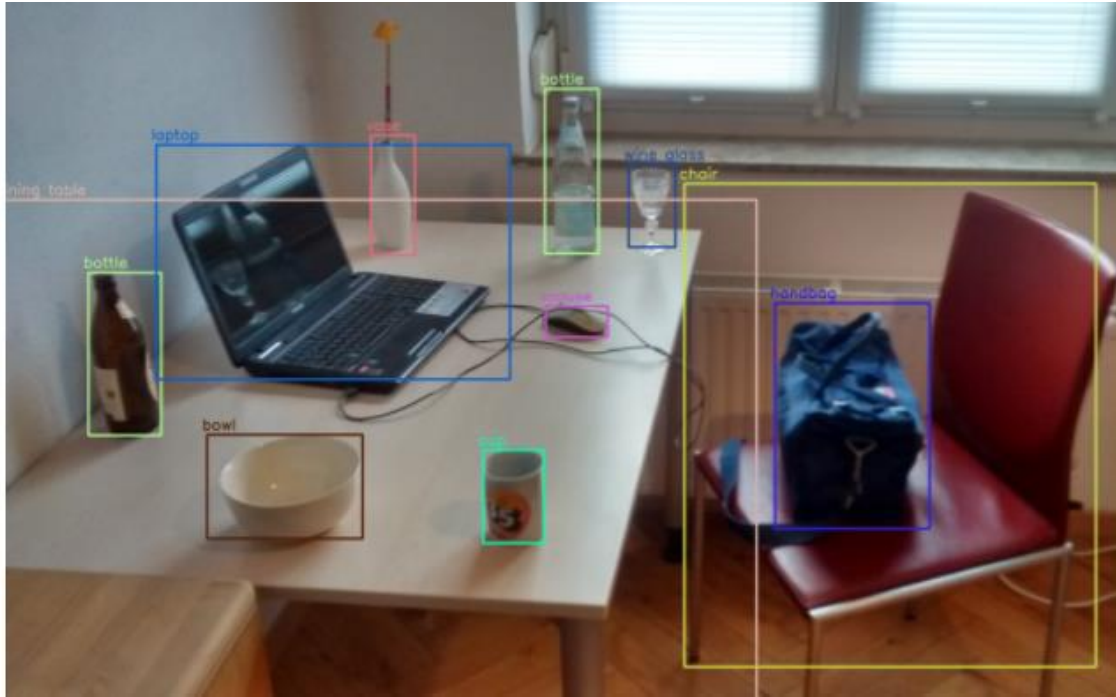


Figure 14 R-CNN example

The R-CNN algorithm logic was suitable for the project, but it has still minor drawbacks like the processing time as it takes a large amount of time to train the network to classify 2000 region proposals per image. Cannot be implemented in real-time as it takes around 47 seconds for each stage where the selective search is a constant algorithm and cannot be replaced in this model which could expose the model to generate bad candidate region proposals. [13]

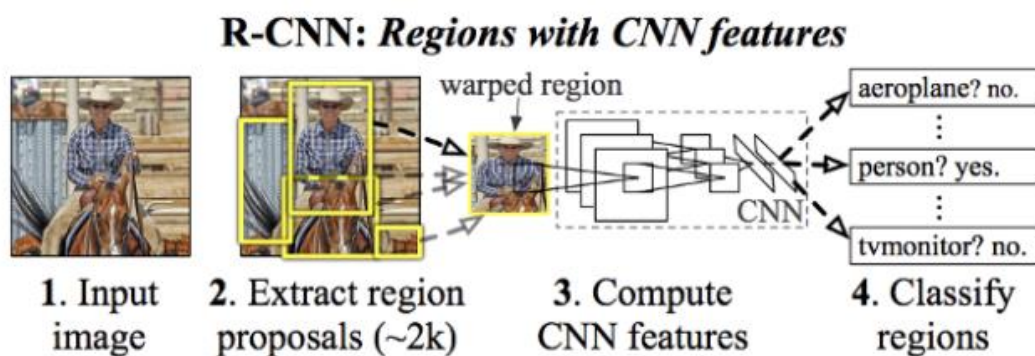


Figure 15 R-CNN architecture

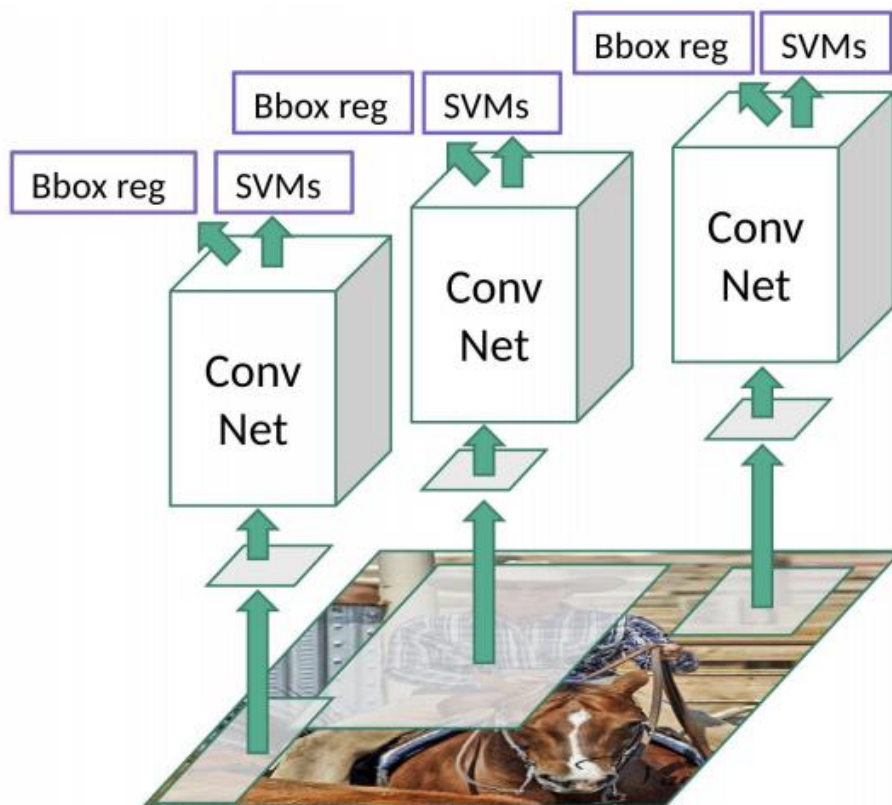


Figure 16 R-CNN architecture pt.2

So, a newer version of R-CNN was invented which is the fast R-CNN that pass image only once to the CNN, use feature maps generated from it to detect objects, and only use one R-CNN so the convolution happens only once. [14]

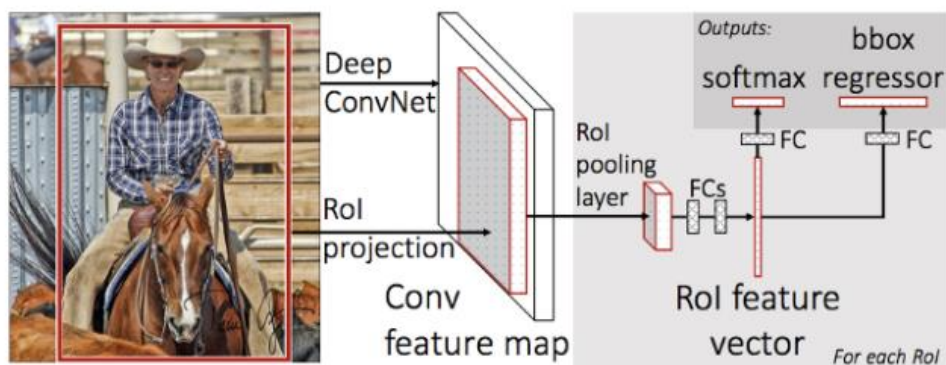


Figure 17 fast R-CNN architecture

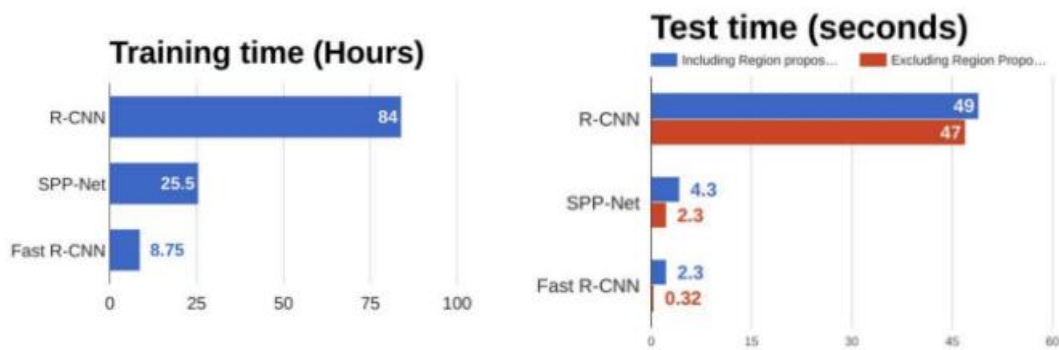


Figure 18 models comparisons

Based on the graph it indicates that the fast R-CNN is faster than the R-CNN in both training time and in the test time but still needs optimization in test time as it still reaches 2.3 seconds per image which is not efficient in the real-time process so according to this the Faster R-CNN is invented to remove the selective search algorithm and let network learn the region proposals by using Region proposal network (RPN) that give the option to determine where to search in the picture and use the Non-Maximum Suppression (NMS) to remove the similar unwanted boxes then send it to the region of interest (ROI).

[15]

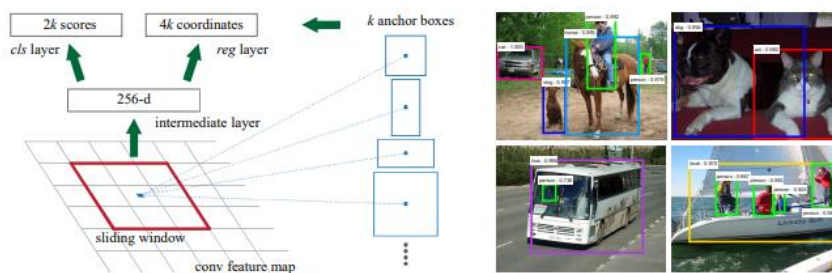
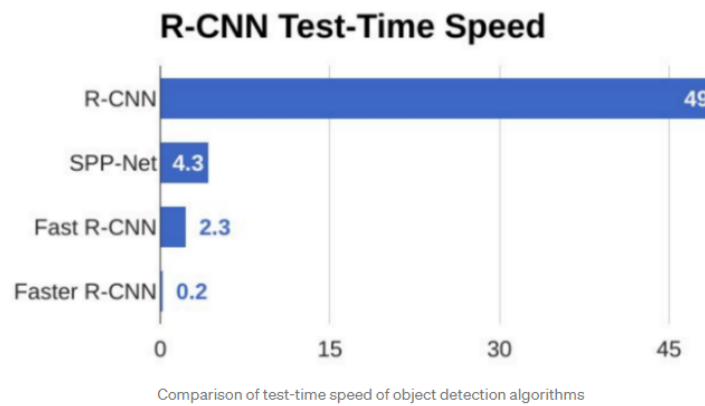


Figure 19 faster R-CNN





*Figure 20 time test between models*

As shown in the comparison of the test-time speed of object detection it shows that Faster R-CNN is the fastest compared to a fast R-CNN or the R-CNN which could determine the objects within 0.2 seconds.

You look once model (YOLO) is commonly used too as it doesn't look at the complete image instead, parts of the image which have a high probability, it is a single convolutional network predicts the bounding boxes and the class probability of this objects' boxes. But this model's drawback is that it finds it difficult to detect small objects so maybe it cannot detect distant objects. As it finds it too difficult to localize objects correctly while the fast R-CNN is 3x more able to predict background detections than YOLO. [16]

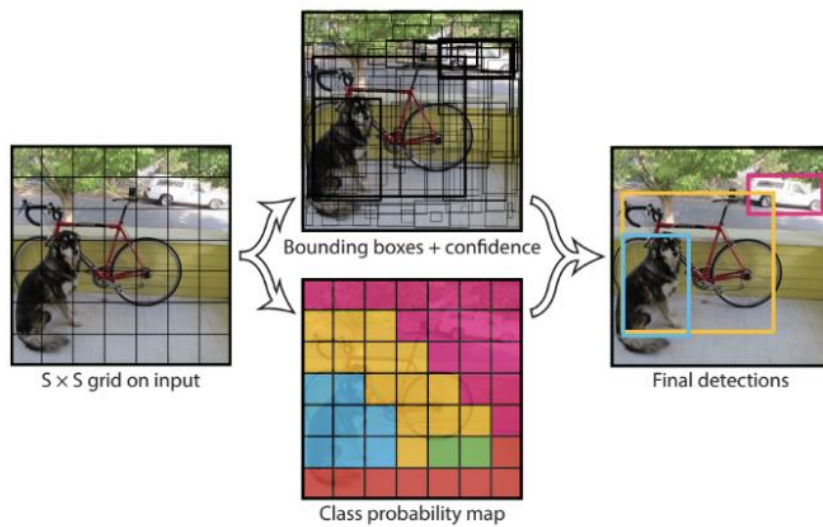


Figure 21 Yolo model example

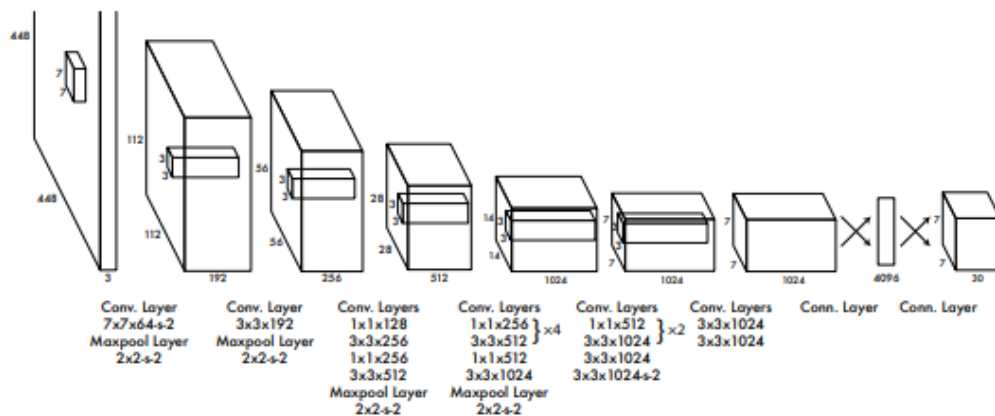


Figure 22 Yolo model architecture

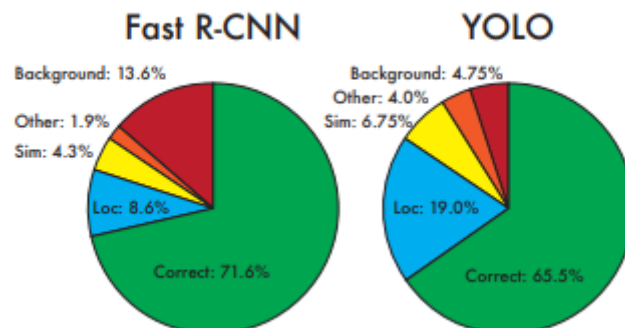


Figure 23 fast R-CNN vs YOLO

Finally, the Masked R-CNN works on the pixel level (image instance segmentation) by using an object mask in addition to faster R-CNN

which improves the region of interest (ROI). Instance segmentation is to detect all objects in an image and segment each instance. And it is easy to train, very efficient, and flexible. Although it is fast, it is not optimized for speed and better speed/accuracy trade-offs could happen by changing the image sizes and proposal numbers. [17]

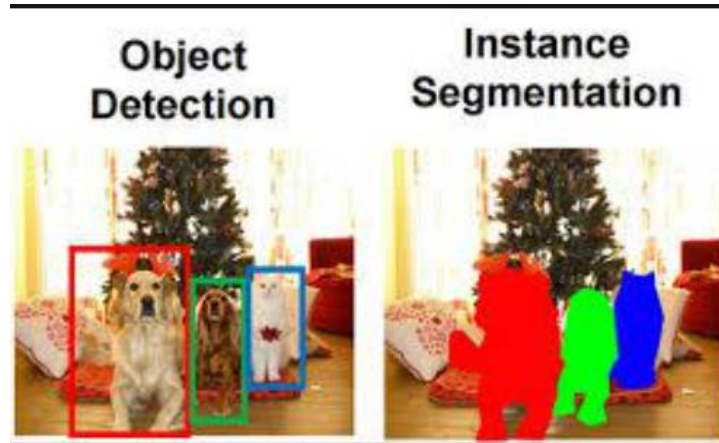


Figure 24 Object Detection vs instance segmentation

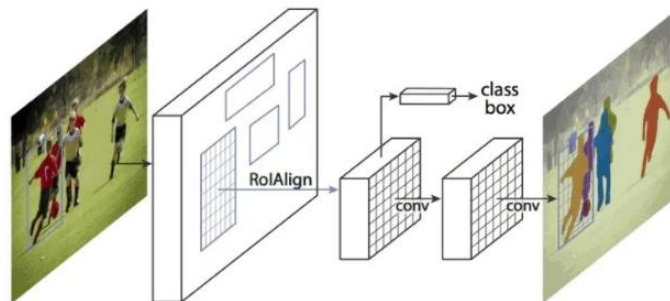


Figure 25 masked RCNN architecture

After studying all these models and according to the speed and the accuracy factors as the model should be suitable for the real-time systems the chosen models are the masked CNN, and the YOLO model, both achieve high speed and accuracy so both will be evaluated to make sure which one will be the most suitable for the Egyptian roads and the vehicles in Egypt as the models' detection are always affected by the environment. What makes the masked CNN is better than the other versions of the CNN its approach is divided into object detection which classifies a variable number of objects in the image and also it does the

semantic segmentation which understands the image at the pixel level as we want to assign the object to each pixel in the image, for the object detection it uses the faster R-CNN model, and for the segmentation, it uses the fully convolutional network (FCN). [17]

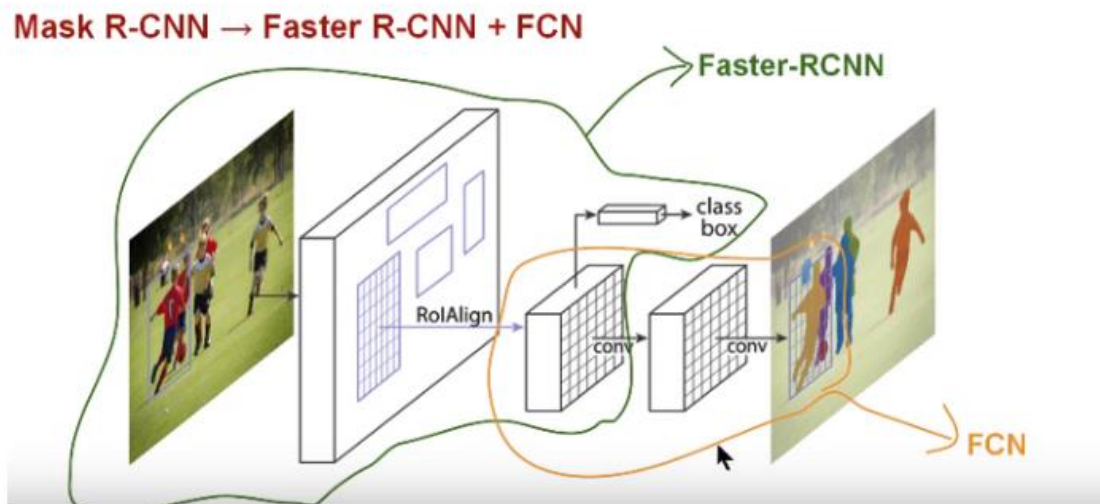


Figure 26 masked RCNN model architecture in detail.

Faster R-CNN uses a regional proposal network (RPN) as it performs the object detection in two stages determining the bounding box and determining the regions of interest by the RPN protocol and the for each ROI pooling we determine the class label of the object, Moreover, FCN is used to predict the mask from each ROI and also uses ROI Align preserves the spatial orientation of features with no loss of data

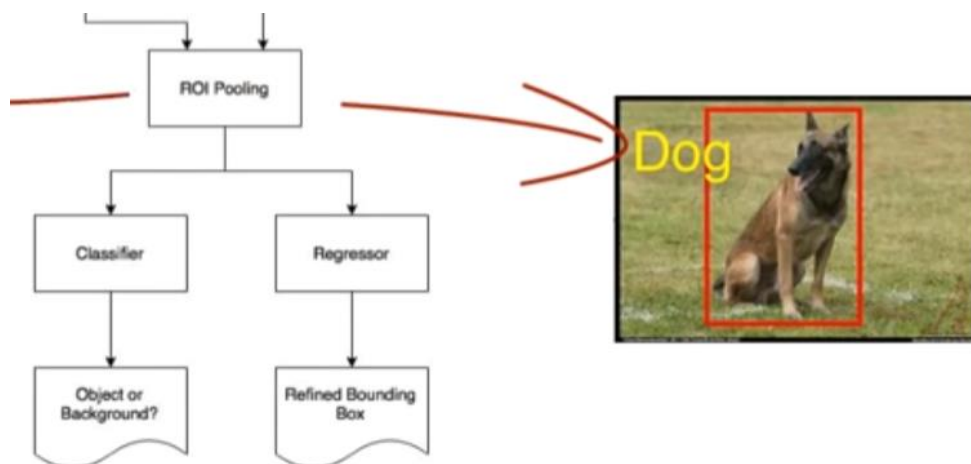


Figure 27 ROI pooling stage

On the other hand, the YOLO (you only look once) is very fast and famous for its real-time systems it was created to solve the problem of the bounding box prediction as the sliding window cannot get it

accurately and doesn't use the computer resources as much as the other model for testing, as firstly it divides the image to the grid and checks each box to check if there is an object or not, and it is checked through a matrix consist of  $P_c$  ( probability of each class),  $B_x$ ,  $B_y$  ( coordinate of each box),  $B_h$ ,  $B_w$  ( box width and height),  $C_n$  (classes that the comparison done related to it).

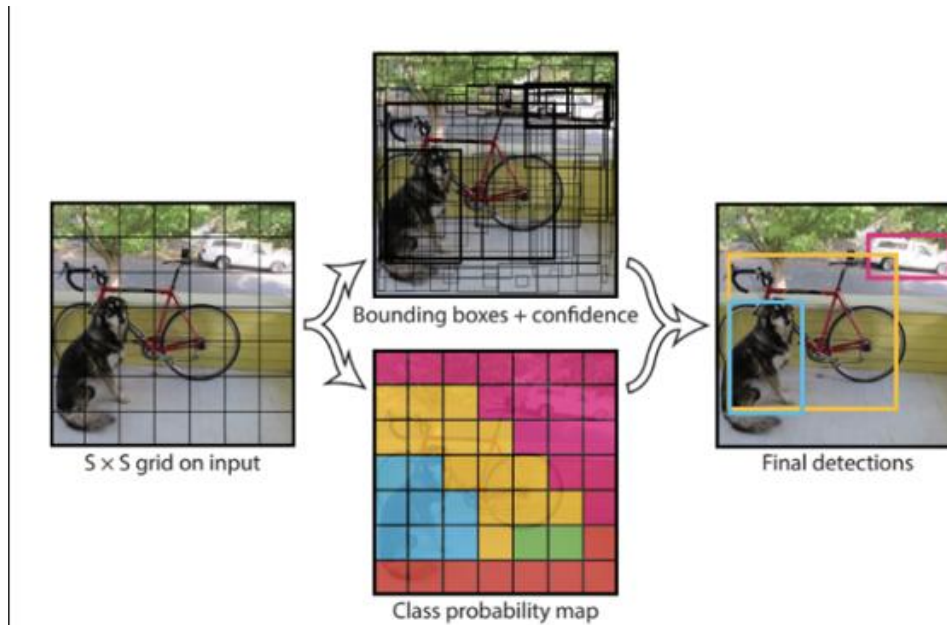


Figure 28 YOLO architecture

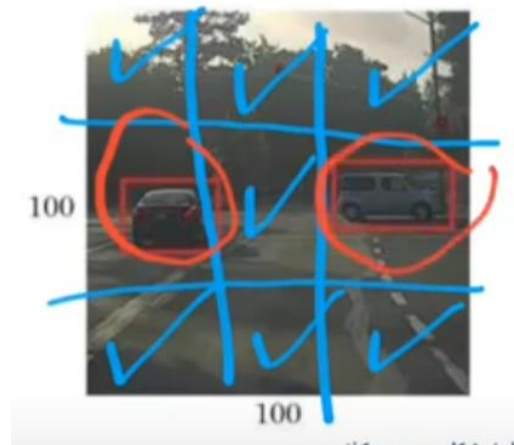


Figure 29 Yolo handmade gride

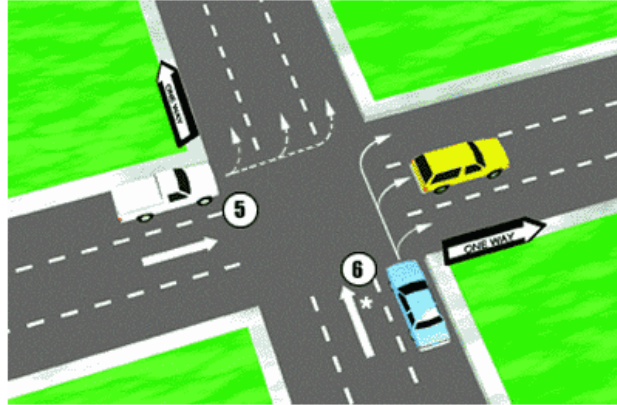
---

Pc  
bx  
by  
bh  
bw  
c1  
c2  
c3

*Figure 30 Yolo matrix for each box*

Moving to the scheduling algorithm, the objective of the object detection models is to pass the vehicle count to this algorithm in which it weights each class showing how many seconds are needed to pass each vehicle and the green time will be calculated by multiplying the counting number of each class by its weight (ex:  $\text{greenTimeCalculation} = \text{vehicleX} * \text{weight} + \text{vehicleY} * \text{weightY}$ ) and because this algorithm is applied on two-lanes road intersect each other in one way direction so by applying the green calculation on the first lane the same number will be applied on the other lane with the same amount, and before changing the lane there will be 4 seconds of yellow light which repeat the detection process and get the count for the lane which is its turn to be changed and this will keep looping and detect each lane correspondingly based on its turn. And in some cases, the green time calculation will not be considered as the time for the traffic lights when the calculation results are less than the minimum calculation time in this case the minimum calculation time will be applied instead, also if the calculation time is more than the traffic light maximum, in this case, the time taken will be the pre defined maximum time. To implement this system the use of threads is a must to be able to compute the timer values for both lanes at the same time in this system multiple threads are running at the same time and performing the timer as the difference between the real-time system and the sequential programming is that real-time is independent and each thread do its computation without a sequence while the sequential is executed step-by-step to produce correct results. And the benefit of the real-time system is that it can respond to events within predictable and specific time constraints that have higher predictability and reliability and it can provide results within a defined deadline. After applying this traffic light organizing algorithm the time will be exactly as needed for

each car so the accumulation of the cars will rarely happen, dislike the traditional system which uses the same time for the rush hours and the low-traffic time not based on the real-time situation.



*Figure 31 two roads intersect each other in a one-way direction*

The masked Rcnn & YOLO models are pre-trained on the COCO dataset, which is huge object detection, segmentation and recognition have more than 330k images with more than 200 thousand of them labeled with more than 1.5 million instances of objects. It is sponsored by Google, Microsoft, Facebook, and mighty AI. [18] [19]



### **2.3 Analysis of the Related Work**

Ann compared to CNN is not efficient because for each image frame it must be converted from 2D to 1D as shown in figure E then apply to it the model which doesn't give high accuracy as shown in the figure, so it needs more training time. While CNN has higher accuracy but is still not suitable for the project because it finds difficulties in detecting more than one object in the same frame, needs huge time to be trained with a good Graphic card and large dataset. While the R-CNN is effective in detecting objects but it needs more than 47 seconds and a huge number of hours to be trained which makes it not for the real-time and the same thing for the fast R-CNN as it needs 2 seconds to detect the objects from the image so the faster R-CNN is suitable and by adding the instance segmentation to it the accuracy will be accelerated which can detect a small object and it is simple to train and flexible which can be used in different approaches as the human pose estimation or OpenStreetMap mapping. And in addition, the YOLO model is also considered to be fast and efficient for the real-time system so it can also be tested for the proposed system and determine which model will be used for the system. The training time and test time comparison between R-CNN and fast R-CNN is shown in

figure S. while all R-CNN types test time comparison is shown in figure U. so the chosen models for the implementation are the Masked R-CNN and the YOLO, in which the scheduling algorithm is designed for the real-time system by the use of the threads to apply the concurrent programming instead of the sequential programming will be used and its objective is to calculate the vehicle count and organize the traffic light based on the current situation.



### 3 Proposed solution

Ascribed to the mentioned problem, the image processing by using Masked R-CNN or the YOLO (you look only once) neural network Model to detect the cars with high accuracy then passing this data to optimized schedule algorithm to manage the roads and traffic lights according to the real-time status is an effective solution as it will be referenced to the current situation no predictions or estimations for the traffic status so it will make sure that there is no over-estimated red light time or short green light duration and this will reduce the wasted fuel, help the environment and decrease the accidents. Both models have few drawbacks so because of this both will be used to evaluate their performance and choose one.

#### 3.1 Solution Methodology

Based on the requirements file attached with each model, the setup of the python environment is done which contains all the libraries needed such as TensorFlow, Keras, Numpy, and Scipy, then import the project for the Masked R-CNN & Yolo, and make some modification to make it able to take pictures or videos and convert this videos to frames. And also add the classes that needed to be defined. And the model will be able to take images or videos as input and classify the type of the objects in each frame then be able to decide on these classified images, a code for retrieving the types of the classes, and the amount of each is required for both models and after this, the scheduling algorithm will be implemented so it first will get the amount of the vehicles from the image and then for each of these classes the weight should be identified to know the duration taken by each vehicle to pass the traffic road then after this, the model will be able to count the green time for the needed lane, as the green time will be equal to the number of each vehicle type multiplied by each vehicle type weight and there will be condition if this calculation is less than the minimum time for the green traffic light time the minimum time for the green time will be taken, while if the green time is more than the green time maximum, the green time maximum will be taken and the opposite lane will have exactly the same waiting time, so according to this the use of threads and the multithreading will make the ability of making each lane as independent and represented at the same time with no conflict, because the multithreading makes each thread works independently without any interference with the other lane. [20]

### **3.2 Functional/ Non-functional Requirements**

#### **Functional requirements**

The system should be able to detect the vehicles and define each vehicle type despite the changes in the surrounding environment then return the count of the vehicles and do the calculation of the next green and red traffic light, in addition to, managing each lane based on the real-time situation.

The inputs to the system are images captured by the CCTV camera or 1-second video, the vehicle's weight, and the minimum green light duration minimum and maximum while the output will be the green/red light duration for each lane.

The vehicle count and green time / red time calculation should be considered.

The system shall start by selecting the first lane and then apply the detection model return the vehicle count results and calculate the green time duration after finish the yellow duration takes place so the model detects the other lane and the loop keeps working.

#### **No-Functional requirements**

The speed must be considered in which the whole process must not exceed the time limit to not break the system.

The system must be portable working with different CCTV camera models and different Operating systems.

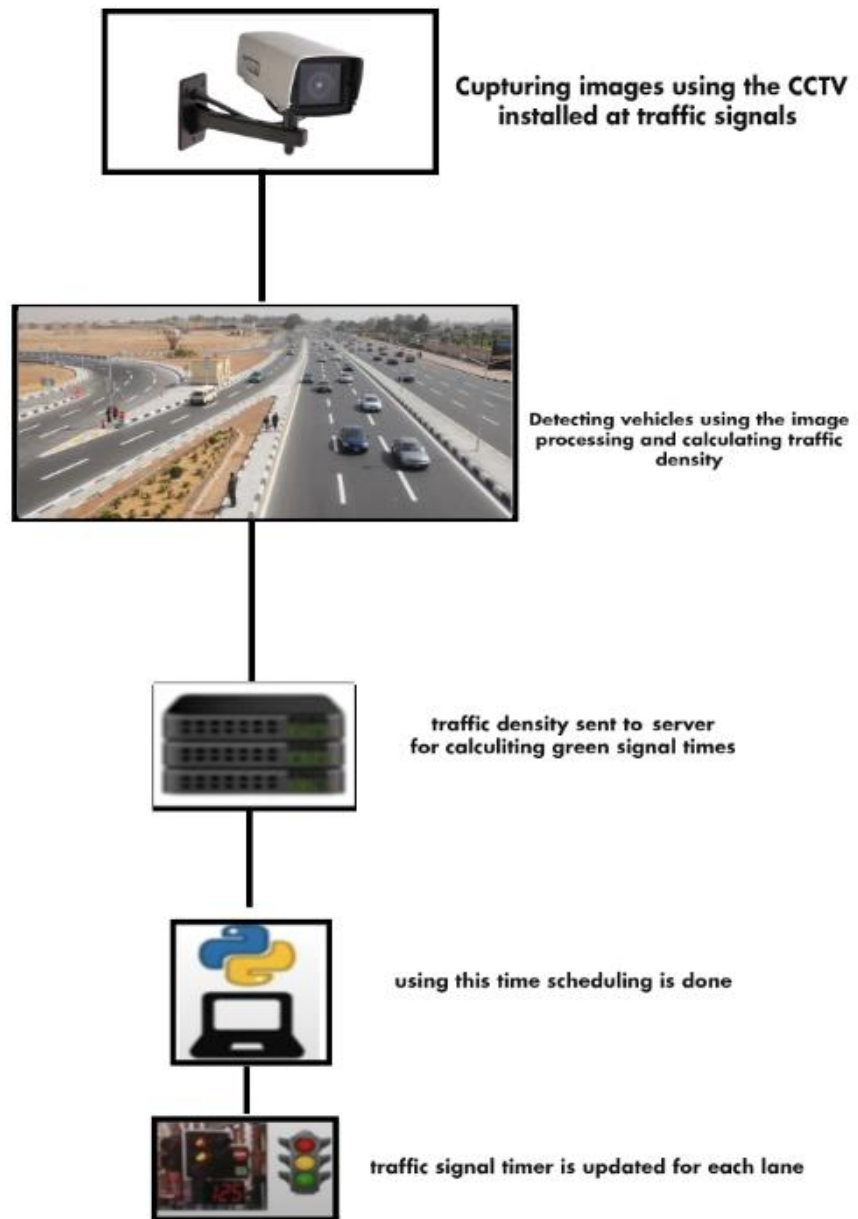
The system must be reliable to avoid system failure.

The system must be available.

The cost must be less than the alternative solutions to be efficient.

The response time must be within four seconds of the yellow traffic light.

### 3.3 Design / Simulation setup



*Figure 32 design flow*

## 4 Implementation

The masked RCNN, the pre-trained model was Matterport Masked RCNN [21].

For the Yolo V5, the pre-trained model was Ultralytics YOLO v5 [22]

```

import os
from os.path import exists, join, basename

project_name = "Mask_RCNN"
if not exists(project_name):
    # clone and install
    !git clone -q https://github.com/matterport/Mask_RCNN.git
    !cd $project_name && pip install -q -r requirements.txt
    !pip uninstall -y tensorflow keras
    !pip install -q keras==2.1.6 tensorflow==1.15.0 h5py==2.10.0

import sys
sys.path.append(project_name)

```

Figure 33 import files from the GitHub

In this block, I downloaded the masked RCNN Matterport files and downloaded the requirements needed for the project as the Keras, TensorFlow, and h5py with their versions

```

# Create model object in inference mode.
model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR, config=config)

# Load weights trained on MS-COCO
model.load_weights(COCO_MODEL_PATH, by_name=True)

# COCO Class names
# Index of the class in the list is its ID. For example, to get ID of
# the teddy bear class, use: class_names.index('teddy bear')
class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',
               'bus', 'train', 'truck', 'boat', 'traffic light',
               'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird',
               'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
               'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie',
               'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
               'kite', 'baseball bat', 'baseball glove', 'skateboard',
               'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup',
               'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
               'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
               'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed',
               'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote',
               'keyboard', 'cell phone', 'microwave', 'oven', 'toaster',
               'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors',
               'teddy bear', 'hair drier', 'toothbrush']

```

Figure 34 add class labels

Then in this block, I added the class names related to the coco dataset because this dataset is used to train the masked CNN model.

```

def RCNN(X):
    vidcap = cv2.VideoCapture(X)
    success,image = vidcap.read()
    count = 0

    print(image)
    while success:
        cv2.imwrite(r"C:\Users\pc\masked_RCNN\frame%d.jpg" % count, image)    # save frame as JPEG file
        success,image = vidcap.read()

    print('Read a new frame: ', success)

    count += 1
    print(count)
    for i in range (count):
        if(i%10==0):
            IMAGE_URL = 'frame'+str(i)+'.jpg'

            image_file = basename(IMAGE_URL)

            image = skimage.io.imread(image_file)
            plt.figure(figsize=(15, 10))
            plt.imshow(image)
            t = time.time()
            # Run detection
            results = model.detect([image], verbose=1)

            print("executed in %.3fs" % (time.time() - t))

            # Visualize results
            r = results[0]
            car_count =0
            truck_count =0
            bus_count =0
            motorcycle_count =0
            person_count=0

            for j in r['class_ids']:

                if(j==3):
                    car_count+=1
                elif(j==8):
                    truck_count+=1
                elif(j==6):
                    bus_count+=1
                elif(j==4):
                    motorcycle_count+=1
                elif(j==1):
                    person_count+=1

            print("car :"+str(car_count))
            print("bus :"+str(bus_count))
            print("truck :"+str(truck_count))
            print("motorcycle :"+str(motorcycle_count))
            print("person :"+str(person_count))

            visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                                       class_names, r['scores'])

    return car_count , truck_count ,bus_count, motorcycle_count , person_count

```

Figure 35 masked RCNN function

This function is responsible for receiving the image and applying the detection of the model on this image then visualizing the results which consist of the image with the scores, class\_ids, masked, and the region of interests border, then returning the counts of the cars, trucks, buses, motorcycles, and people from the image. And print the execution time for the detection process.

```

#range for the green light
default_green_min = 10
default_green_max = 60

#weights for every vechile type
CarTime = 2
BikeTime = 1
BusTime = 2.5
TruckTime = 2.5

```

Figure 36 adds vehicles weights

Here the min and max range for the green light is added with the weights for every vehicle type in seconds.

```

#green and Red time calculation function
def greentime_OnTurn_calc(noOfCars_lane,noOfBikes_lane,noOfBuses_lane,noOfTrucks_lane):

    GreenTimeLane = math.ceil(((noOfCars_lane*CarTime) + (noOfBuses_lane*BusTime) + (noOfBikes_lane*BikeTime) + (noOfTrucks_lane*TruckTime)
    print("green time for the other lane is :", GreenTimeLane)

    if(GreenTimeLane<default_green_min):

        GreenTimeLane = default_green_min
        next_G = GreenTimeLane
        next_R = next_G

    elif(GreenTimeLane>default_green_max):

        GreenTimeLane = default_green_max
        next_G = GreenTimeLane
        next_R = next_G

    else:

        next_G = GreenTimeLane
        next_R = next_G

    print(next_G , next_R)
    return next_G , next_R

```

Figure 37 green time calculations

Then the green time calculation function is responsible for the calculating the green time duration for each lane based on the parameters taken which are the number of the cars, buses, bikes, and trucks from the image multiplied by their weight then after getting this result it checks if the green time is less than the green time default minimum or not if it less than this minimum it will be equal to this value and the same happen if it is more the default maximum is also replaced with the green maximum, and if it Is within the maximum value and the minimum value it will take the exact value

```

#timer for the red
def red(time_sec):
    while time_sec:
        mins, secs = divmod(time_sec, 60)
        timeformat = '{:02d}:{:02d}'.format(mins, secs)

        print("Red Traffic :",timeformat, end="\n")

        time.sleep(1)
        time_sec -= 1
    Yellow(4)

#timer for the green time
def Greencountdown(time_sec):

    while time_sec:
        mins, secs = divmod(time_sec, 60)
        timeformat = '{:02d}:{:02d}'.format(mins, secs)
        print("Green Traffic :",timeformat, end="\n")

        time.sleep(0.99)
        time_sec -= 1

def Yellow(time_sec):
    while time_sec:
        mins, secs = divmod(time_sec, 60)
        timeformat = '{:02d}:{:02d}'.format(mins, secs)
        print("Yellow Traffic :",timeformat, end="\n")

        time.sleep(1)
        time_sec -= 1

```

Figure 38 timers

Then the timer functions will take place by adding the amount of time needed for each as shown the yellow is fixed with 4 seconds and gets executed after the red timer finishes counting down.

```

i = 1
n = 1
flag = 0
while i <= 3:
    if(flag == 0):
        print("~~~~~first lane turn !!!~~~~~")
        flag = 1
    else:
        print("~~~~~Second lane turn !!!~~~~~")
        flag = 0

    #int to string to concatenate in the image
    s = f'{n}'

    car_count, truck_count, bus_count, motorcycle_count, person_count = RCNN(r'C:\Users\pc\masked_RCNN\lane'+s+'.jpg')
    Next_Green, Next_RED = greentime_OnTurn_calc(car_count, motorcycle_count, bus_count, truck_count)

    if(Next_Green != 0):
        t = Thread(target= Greencountdown, args=(Next_Green,)).start()
        t2 = Thread(target= red, args=(Next_RED,)).start()

    #to iterate for the next iteration after the Red and green finish !
    time.sleep(Next_RED)
    n = n+1
    i = i+1

```

Figure 39 implementing threads and calling the function

This is the part that executes the RCNN model and executes the green time calculation then if the green time function has a value and not equal to zero it starts to trigger the green countdown timer and also the red for the other lane and the number of iterations can be infinite but in this case, to test the program it will iterate 3 times only. And as shown each timer is implemented in a thread in which the green countdown is in (t) Thread while the red is in t2 and the frames will go from n=1 and increment by 1 while the loop is looping.

While for the Yolo model

```

In [7]: #first time only
        #!git clone https://github.com/ultraLytics/yolov5
        #!pip install -r requirements.txt

fatal: destination path 'yolov5' already exists and is not an empty directory.

In [7]: #change directory
        %cd yolov5

```

Figure 40 run YOLO model

The first block will install the YOLO v5 model and apply its requirements to the environment

And then change the directory to apply the YOLO detection process on the same images



```
def yolo_detection(x):
    !python detect.py --source C:\Users\pc\masked_RCNN\lane(x).jpg --weights yolov5s.pt
    #Image(filename=r'C:\Users\pc\masked_RCNN\yolov5\yolov5\runs\detect\exp3\lane2.jpg', width=300)
```

Figure 41 YOLO function

The YOLO detection function is created by adding the image number variable in the parameter

```
if len(det):
    # Rescale boxes from img_size to im0 size
    det[:, :4] = scale_coords(im.shape[2:], det[:, :4], im0.shape).round()
    dict = {}
    # Print results
    for c in det[:, -1].unique():
        n = (det[:, -1] == c).sum() # detections per class
        s += f"{n} {names[int(c)]}'s' * (n > 1)}, " # add to string

        count = n.item()

        if count > 0:
            dict[names[int(c)]] = n.item()
            import yaml
            dict_file = dict;

    with open(r'C:\Users\pc\masked_RCNN\yolov5\OUTPUT.YAML', 'w') as file:
        documents = yaml.dump(dict_file, file)
    # Write results
    for *xyxy, conf, cls in reversed(det):
        if save_txt: # Write to file
            xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn.view(-1)).tolist() # normalized xywh
            line = (cls, *xywh, conf) if save_conf else (cls, *xywh) # label format
            with open(txt_path + '.txt', 'a') as f:
                f.write('%g ' * len(line)).rstrip() % line + '\n')

        if save_img or save_crop or view_img: # Add bbox to image
            c = int(cls) # integer class
            label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
            annotator.box_label(xyxy, label, color=colors(c, True))
            if save_crop:
                save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)
```

Figure 42 import detection values to the YAML file

Then this part of the code I added to the detect.py file of the Yolo model to get the number of items and add it to the count then if the count has a value more than zero it means that there are objects found in the image so these items will be saved in direct variable holding the detected classes names and then open a Yaml file and save this classes names to the Yaml file so it will output the classes names corresponding to it the amount of each.

```
Debug Help
C:/Users/pc/masked_RCNN/yolov5/OUTPUT.YAML (Getting Started) - Brackets

1 car: 1
2 person: 2
```

Figure 43 Yaml file data

```

import yaml

def yolo_output():
    car=0
    bus=0
    bike=0
    truck=0

    with open(r'C:\Users\pc\masked_RCNN\yolov5\OUTPUT.YAML') as file:
        documents = yaml.full_load(file)

    if('bus' in documents):
        bus = documents['bus']
    if('bike' in documents):
        bike = documents['motorcycle']
    if('truck' in documents):
        truck = documents['truck']
    if('car' in documents):
        car = documents['car']
    print(documents)

    return car , bus , truck , bike

```

Figure 44 YOLO output function the results from YAML file

Then these results are saved in the YAML file and then this output will be used to return the count of the vehicles as car count, bus count, trucks count, and motorcycle count.

```

i = 1
n = 1
flag = 0
while i <= 3:
    if(flag == 0):
        print("~~~~~first lane turn !!!~~~~~")
        flag =1
    else:
        print("~~~~~Second lane turn !!!~~~~~")
        flag =0

    #int to string to concatenate in the image
    s = f'{n}'
    yolo_detection(i)
    car , bus , truck , bike = yolo_output()
    Next_Green , Next_RED = greentime_OnTurn_calc(car,bike,bus,truck)

    if(Next_Green != 0):
        t = Thread(target= Greencountdown, args=(Next_Green,)).start()
        t2 = Thread(target= red, args=(Next_RED,)).start()
        time.sleep(Next_RED)
        n = n+1
        i = i+1

```

Figure 45 implementing threads and calling functions

this part is the main in which all the functions are called such as the YOLO detection work and then getting the count values by the output function and then calculating the green time on each turn and if it has a value it goes to the timers to work as the masked RCNN in threads

## Masked RCNN from video

```
l2]: import cv2
vidcap = cv2.VideoCapture('lane.mp4')
success, image = vidcap.read()
count = 0

print(image)
while success:
    cv2.imwrite(r"C:\Users\pc\masked_RCNN\frames\frame%d.jpg" % count, image)    # save frame as JPEG file
    success, image = vidcap.read()
    print('Read a new frame: ', success)
    count += 1
```

Figure 46 masked RCNN with video files

This part can be replaced for the masked RCNN function if the camera only captures videos as this function take the video and sperate the frames.

```
: print(count)
for i in range (count):
    if(i%10 ==0 ):
        IMAGE_URL = r"C:\Users\pc\masked_RCNN\frames\frame"+str(i)+".jpg"

        image_file = IMAGE_URL

        image = skimage.io.imread(image_file)
        plt.figure(figsize=(15, 10))
        plt.imshow(image)
        t = time.time()
        # Run detection
        results = model.detect([image], verbose=1)
        print("executed in %.3fs" % (time.time() - t))

        # Visualize results
        r = results[0]
        car_count =0
        truck_count =0
        bus_count =0
        motorcycle_count =0
        person_count=0

        for j in r['class_ids']:

            if(j==3):
                car_count+=1
            elif(j==8):
                truck_count+=1
            elif(j==6):
                bus_count+=1
            elif(j==4):
                motorcycle_count+=1
            elif(j==1):
                person_count+=1

        print("car :"+str(car_count))
        print("bus :"+str(bus_count))
        print("truck :"+str(truck_count))
        print("motorcycle :"+str(motorcycle_count))
        print("person :"+str(person_count))

        visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                                   class_names, r['scores'])
```

Figure 47 the change in the code when it detects from video

Then this function loops on the frames count and every 10 frames it takes only one frame to process because the condition pass in  $i\%10 ==0$  and does the object detection, displaying the instances with masks and class ids and scores.

## 5 Testing and evaluation

### 5.1 Testing

Moving to the testing of this real-time system I have tested it in different environments and different places.

**Masked RCNN** picked the first lane image with good results as shown



*Figure 48 first lane 1st iteration MASKED RCNN Results*

As this image was taken at 8 pm while vehicles were moving at a speed of 40~60km/hr it could detect the variation between the vehicles as it detected the cars and the van as a truck

```

Read a new frame: False
1
Processing 1 images
image          shape: (648, 459, 3)      min:  0.00000 max: 255.00000 uint8
molded_images  shape: (1, 1024, 1024, 3) min: -123.70000 max: 151.10000 float64
image metas    shape: (1, 93)          min:  0.00000 max: 1024.00000 float64
anchors        shape: (1, 261888, 4)   min: -0.35390 max:  1.29134 float32
executed in 4.056s
car :7
bus :0
truck :1
motorcycle :0
person :0

```

Figure 49 detection details and execution time

As shown, it was detected in 4.056 seconds and detected 7 cars and one truck.

```

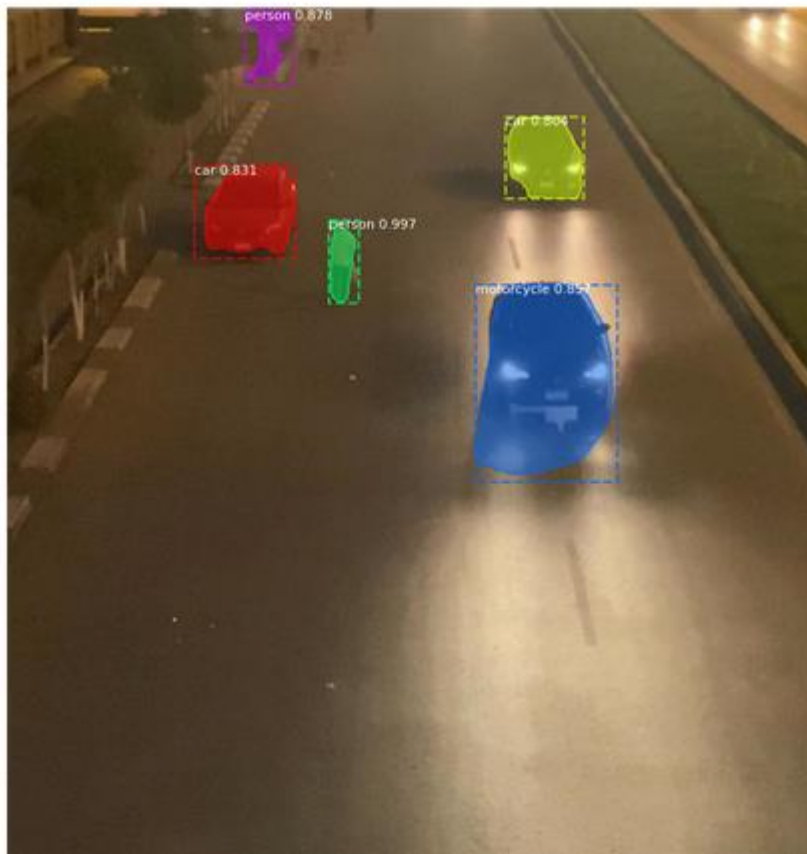
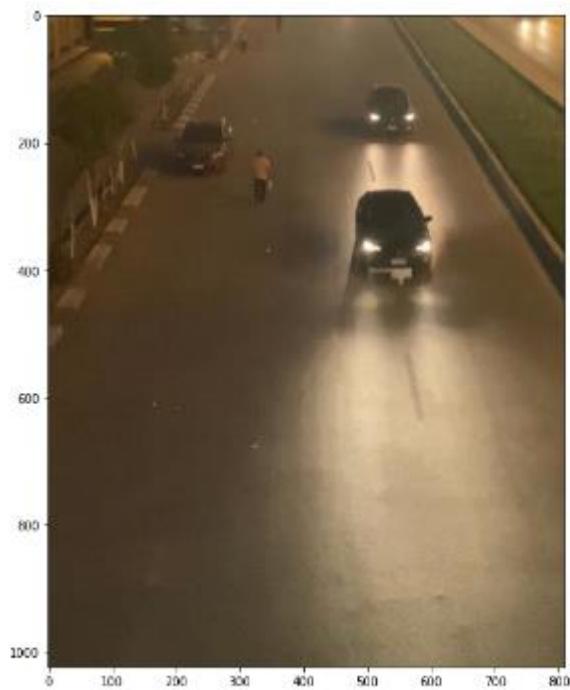
green time for the other lane is : 17
17 17
Green Traffic : 00:17
Red Traffic : 00:17
Green Traffic : 00:16
Red Traffic : 00:16
Green Traffic : 00:15
Red Traffic : 00:15
Green Traffic : 00:14
Red Traffic : 00:14
Green Traffic : 00:13
Red Traffic : 00:13
Green Traffic : 00:12
Red Traffic : 00:12
Green Traffic : 00:11
Red Traffic : 00:11
Green Traffic : 00:10
Red Traffic : 00:10
Green Traffic : 00:09
Red Traffic : 00:09
Green Traffic : 00:08
Red Traffic : 00:08
Green Traffic : 00:07
Red Traffic : 00:07
Green Traffic : 00:06
Red Traffic : 00:06
Green Traffic : 00:05
Red Traffic : 00:05
Green Traffic : 00:04
Red Traffic : 00:04
Green Traffic : 00:03
Red Traffic : 00:03
Green Traffic : 00:02
Red Traffic : 00:02
Green Traffic : 00:01
Red Traffic : 00:01
~~~~~Second lane turn !!!~~~~~

```

Figure 50 scheduling algorithm

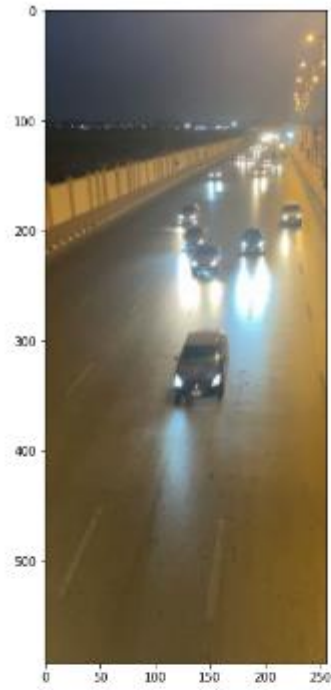
Then the green time was calculated then the real-time timer worked both the green and red traffic light was counting at the same time and

when it finished the second lane turn came to be detected at the yellow duration.



*Figure 51 2nd lane 1st iteration masked RCNN results*

The detection here was accurate by 3 out of 5 and this was because of the light condition as it was darker than the other lane.



*Figure 52 1st lane 2nd Masked RCNN iteration result*

Then by moving to the second turn the detection was better than in the previous lane but there were 2 cars not detected because of the motion

blur while moving and this was the last in the loop, so the detection was done and the real-time count after finishing the system stopped

```
green time for the other lane is : 8
10 10
Green Traffic : 00:10
Red Traffic : 00:10
Red Traffic :Green Traffic : 00:09
  00:09
Green Traffic : 00:08
Red Traffic : 00:08
Red Traffic :Green Traffic : 00:07
  00:07
Green Traffic :Red Traffic : 00:06
  00:06
Red Traffic :Green Traffic : 00:05
  00:05
Green Traffic : 00:04
Red Traffic : 00:04
Red Traffic :Green Traffic : 00:03
  00:03
Green Traffic : 00:02
Red Traffic : 00:02
Green Traffic : 00:01
Red Traffic : 00:01
Yellow Traffic : 00:04
Yellow Traffic : 00:03
Yellow Traffic : 00:02
Yellow Traffic : 00:01
```

*Figure 53 the last iteration scheduling system results*

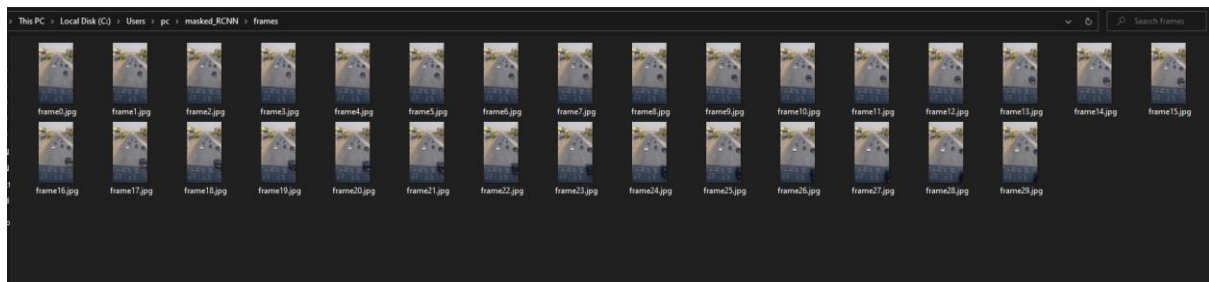
Moving to the detection from the video after adding the video the frames were separated into frames and they kept separating until no frames is left as shown



```
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: True
Read a new frame: False
```

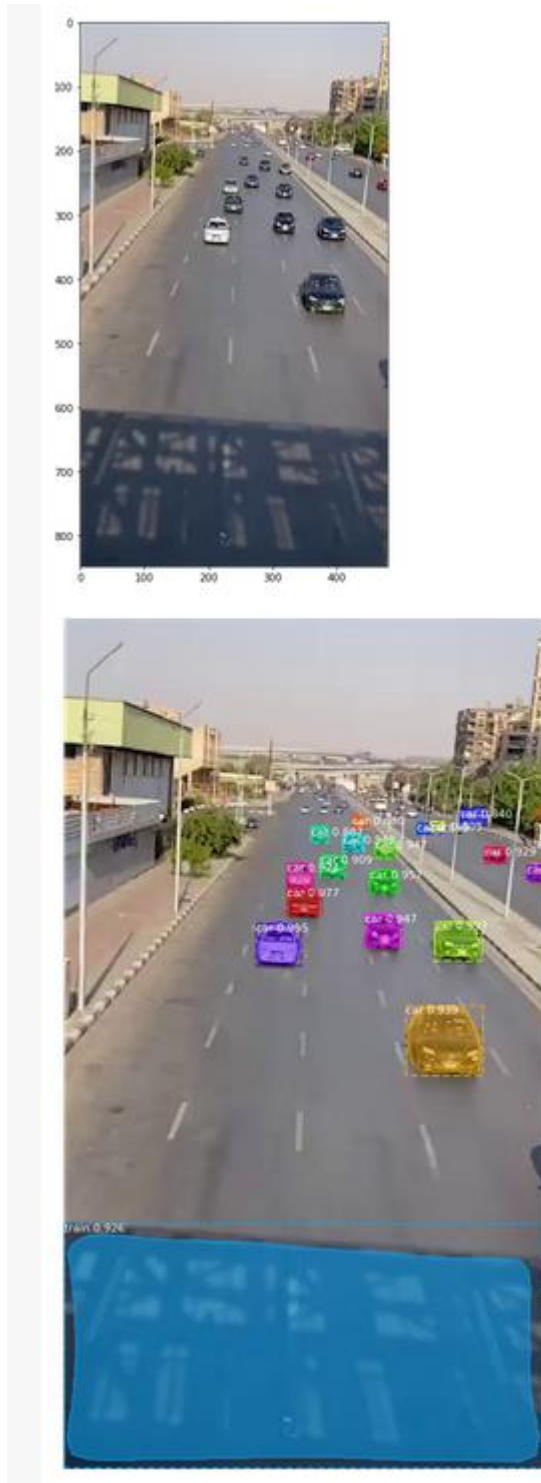
*Figure 54 reading video frames Masked RCNN*

The video condition was taken at 12:30 pm while cars were moving at 40~60km/hr the video is 30 frames per second and it is 1 second so the frames were 30 frames



*Figure 55 video to frames by Masked RCNN video function*

Then these frames were detected by the model



*Figure 56 frame detection by Masked RCNN*

As shown and because of the shadow it detected a train and this was not truly good but on the other hand, all the cars were detected which is very good. The duration taken for this was 2.595 seconds

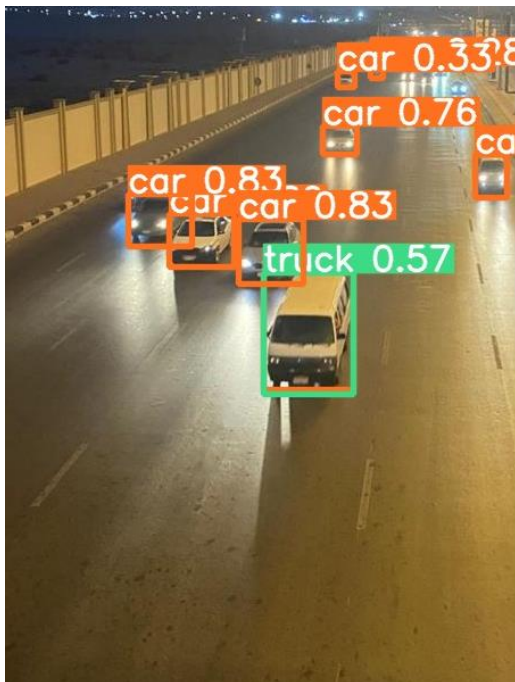
```

30
Processing 1 images
image          shape: (848, 480, 3)      min:  0.00000 max: 255.00000 uint8
molded_images  shape: (1, 1024, 1024, 3) min: -123.70000 max: 151.10000 float64
image metas    shape: (1, 93)       min:  0.00000 max: 1024.00000 float64
anchors        shape: (1, 261888, 4) min: -0.35390 max:  1.29134 float32
executed in 2.595s
car :17
bus :0
truck :0
motorcycle :0
person :0

```

*Figure 57 detection results*

While the YOLO detection



*Figure 58 1st lane 1st iteration YOLO detection*

It detected 8 cars and 1 truck so as shown the green time and red time was counting down and the same time until the yellow time and within the yellow, the new detection was ready to start after.

```

{'car': 8, 'truck': 1}
1
8
green time for the other lane is : 19
19 19
Green Traffic : 00:19
Red Traffic : 00:19
Green Traffic : 00:18
Red Traffic : 00:18
Green Traffic : 00:17
Red Traffic : 00:17
Green Traffic : 00:16
Red Traffic : 00:16
Green Traffic : 00:15
Red Traffic : 00:15
Green Traffic : 00:14
Red Traffic : 00:14
Green Traffic : 00:13
Red Traffic : 00:13
Green Traffic : 00:12
Red Traffic : 00:12
Green Traffic : 00:11
Red Traffic : 00:11
Green Traffic : 00:10
Red Traffic : 00:10
Green Traffic : 00:09
Red Traffic : 00:09
Green Traffic : 00:08
Red Traffic : 00:08
Green Traffic : 00:07
Red Traffic : 00:07
Green Traffic : 00:06
Red Traffic : 00:06
Green Traffic : 00:05
Red Traffic : 00:05
Green Traffic : 00:04
Red Traffic : 00:04
Green Traffic : 00:03
Red Traffic : 00:03
Green Traffic : 00:02
Red Traffic : 00:02
Green Traffic : 00:01
Red Traffic : 00:01
~~~~~Second lane turn !!!~~~~~
Yellow Traffic : 00:04
Yellow Traffic : 00:03
Yellow Traffic : 00:02
Yellow Traffic : 00:01

detect: weights=['yolov5s.pt'], source=C:\Users\pc\masked_RCNN\lane2.jpg, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=, view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=False
YOLOv5 v6.1-176-gaa7a0e9 torch 1.11.0+cpu CPU

```

Figure 59 schedule algorithm results for 1st lane

The time taken for the first detection was 0.160 seconds

```

detect: weights=['yolov5s.pt'], source=C:\Users\pc\masked_RCNN\lane1.jpg, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=, view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=False
YOLOv5 v6.1-176-gaa7a0e9 torch 1.11.0+cpu CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients
image 1/1 C:\Users\pc\masked_RCNN\lane1.jpg: 640x480 8 cars, 1 truck, Done. (0.160s)
Speed: 1.0ms pre-process, 160.0ms inference, 1.0ms NMS per image at shape (1, 3, 640, 640)

```

Figure 60 detection summary including time

In the second lane, it detected the 3 cars and the 3 persons even with the bad light condition and also within 0.160 s

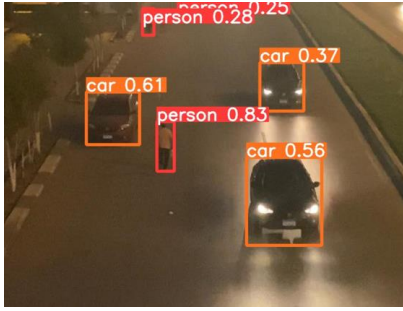


Figure 61 2nd lane 1st iteration Yolo detection

In the last lane, the YOLO model could detect 4 cars and calculate the time then after finishing it the system stopped as the loop ended.



Figure 62 1st lane 2nd iteration YOLO detection

## 5.2 Evaluation

**Time evaluation** for both models on the same images the first image was processed with the masked RCNN in 4.056 seconds while the Yolo was just processed in 0.160s

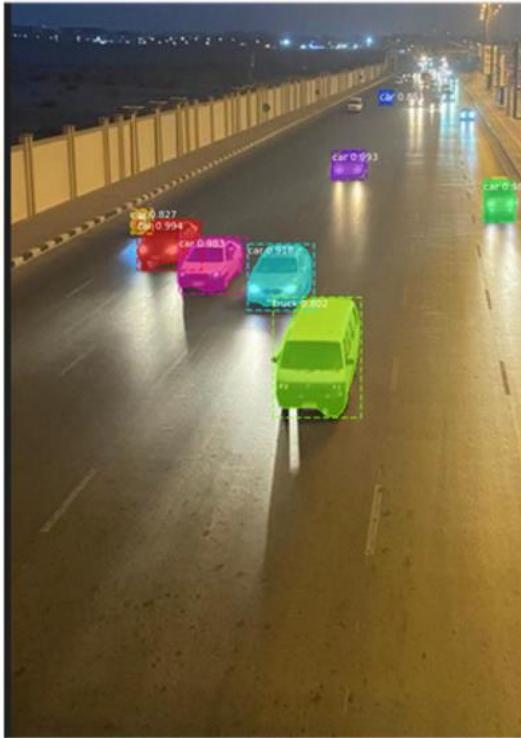
While the second image was processed with the masked RCNN in 2.701s while the YOLO was just processed in 0.179 seconds

The third picture was processed with the masked RCNN in 2.674s while with YOLO just in 0.104 seconds

Performance evaluation for both models on the same images

The first image

Figure 63 1st lane Masked RCNN detection



VS

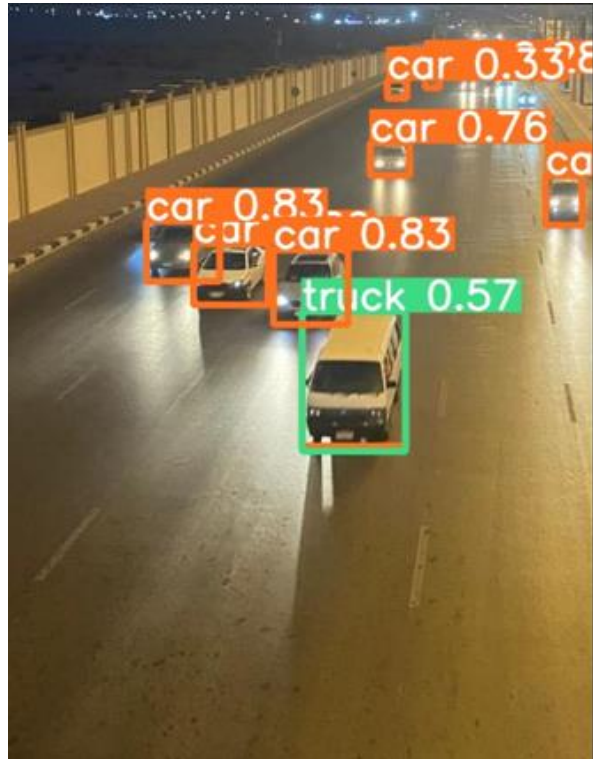


Figure 64 1st lane Yolo detection

In masked RCNN

Car count = 7

Truck = 1

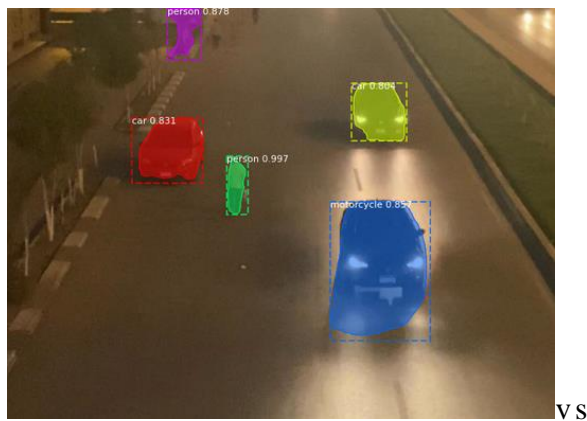
In YOLO V5

Car count = 8

truck = 1



Figure 65 2nd lane masked Rcnm detection



VS



Figure 66 2nd lane YOLO detection

In masked RCNN

Car count = 2

Motorcycle = 1 (wrong detection)

Persons = 2 (with 1 wrong detection)

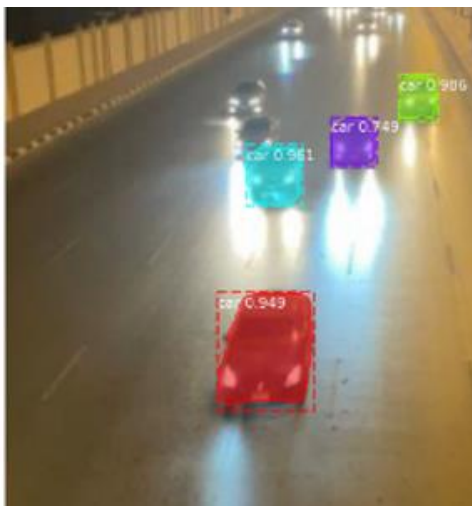
In YOLO V5

Car count = 3

Motorcycle = 0

person = 3

Figure 67 1st lane 2nd iteration Masked RCNN



VS



Figure 68 1st lane 2nd iteration yolo

In masked RCNN

Car count = 4

In YOLO V5

Car count = 4

Total traffic system duration

In masked RCNN

49-seconds

In YOLO V5

51-seconds

## 6 Results and Discussions

According to the testing and evaluation stages, some advantages for the models appeared and also some drawbacks appeared the main advantage for the two models is that both were able to detect the vehicles and count them and calculate the traffic light duration in real-time with difficult environments conditions as the lights were poor and the detection was at the night and the vehicles were moving and the images were in an angle while the drawbacks appeared clearly in the masked RCNN as in one of the detections it was exactly finished as it finished in 4 seconds while also it detected some wrong classes, for example, it detected the car as a motorcycle, and detected the tree as a person, Moreover, it detected the bridge shadow as a train, while the Yolo did not miss any class but has some difficulties in detecting faraway objects. And according to this, the Yolo detection took more few seconds than the Masked RCNN for the run time of the system. The following tables show that both models passed nearly the same amount of cars in duration less than the old technique by the double amount as the Yolo took only 51 seconds and the Masked RCNN took only 49 seconds while the traditional method took 112 seconds



Masked RCNN TABLE

LANE	Object COUNT	GREEN TIME <b>Red opposite lane</b>	CAR PASSED
First Lane	7 cars 1 truck	17 second	All the cars and the van
Second lane	2 cars 1 motorcycle 2 persons	The green time needed is just 5 sec < green min so Green time = 10 seconds	All the cars and the (wrong )motorcycles pass
First lane	4 cars	The green time needed is just 8 sec < green min so Green time = 10 seconds	All the cars pass

Yolo v5 on the same lane

LANE	object COUNT	GREEN TIME <b>Red opposite lane</b>	CAR PASSED
First Lane	8 cars 1 truck	19 second	All the cars (but did not detect the far ones)
Second lane	3 cars 3 persons	Green time needed is just 6 sec < green min so Green time = 10 seconds	All the cars passed  As the other model detected the wrong motorcycle!
First lane	4 car	The green time needed is just 8 sec < green min so Green time = 10 seconds	All the cars pass  (Did not detect the far ones)

Traditional traffic lights on the same lane

LANE	object COUNT	GREEN TIME <b>Red opposite lane</b>	CAR PASSED
First Lane	8 cars 1 truck	30 seconds (fixed)	All the cars passed
Second lane	3 cars 3 persons	30 seconds (fixed)	All the cars passed
First lane	8 cars	30 seconds (fixed)	All the cars passed

Traditional system lane time	YOLO V5 system lane time	Masked RCNN system lane time
112 seconds	51 seconds	49 seconds

## 7 Conclusions and Future Work

### 7.1 Summary

Overall, applying the real-time traffic light system is a breakthrough in technology, as using the artificial intelligence in traffic will decrease the waiting time and the traffic congestion, moreover, it will decrease the car accidents as the roads will be more organized and the majority of cars will not wait for 2 turns of the traffic light to pass the road and will decrease the costs spent on the expensive sensors, and will decrease the human-power usage, lower the unwanted delays and the waiting time which will affect the fuel usage and consumption and the pollution. According to the simulation, the system shows about 40% improvement and this percentage is not fixed and depends on the car flow, and according to the situation of crossing the intersection, it shows a significant improvement. This system can be integrated with the CCTV cameras in the Egyptian cities to give better management of traffic.

## 7.2 Future Work

According to the results, the masked RCNN was neither efficient in detection nor time, as in poor light conditions it detects wrong classes, also it detects the shadows as objects which can cause wrong calculations, and regarding the time the detection time was between the 2 seconds as minimum detection value and the highest value was 4.5 seconds and this was very late because the yellow traffic light duration is only limited to 4 seconds and this delay can cause failure in detection. While the Yolo, the results were more accurate even in the poor light conditions, moreover the detection speed was less than 1 second, so the detection speed was in the very safe zone in all turns. So according to the results what went well is the scheduling algorithm was efficient for the two roads intersection and the detection with YOLO was very good, while the things that need to be improved in both models is the moving object detection as both models were not accurate in that detection type, and the masked R-CNN the COCO dataset was not enough so in the future the custom dataset from the Egyptian roads can be used, the time taken for detection need to be optimized to be in the safe zone and can be used in the future because it can detect faraway objects so it will increase the ability of the system. There should be more schedule algorithms with the same idea but on different roads types, as the straight roads there can be schedule algorithms to detect the people on the side of the road and count them and do its calculation to make them all pass the road without huge delay for the cars, also in the future work, identify vehicles running red lights, accidents or breakdown detection and adapting the emergency vehicles such as ambulance or the fire trucks can be considered to decrease the waiting time based on their appearance in the future.

## References

- [ worldbank. [Online]. Available:  
1 <https://www.worldbank.org/en/country/egypt/publication/cairo-traffic->  
] [congestion-study-executive-note](https://www.worldbank.org/en/country/egypt/publication/cairo-traffic-congestion-study-executive-note).
- [ T. C. I. CAIRO. [Online]. Available:  
2 [https://documents1.worldbank.org/curated/en/990291468038079976/pdf/](https://documents1.worldbank.org/curated/en/990291468038079976/pdf/718520WP0Box370tudy0Overview0Final.pdf)  
] [f/718520WP0Box370tudy0Overview0Final.pdf](https://documents1.worldbank.org/curated/en/990291468038079976/pdf/718520WP0Box370tudy0Overview0Final.pdf).
- [ K. Zaatouri. [Online]. Available:  
3 [https://www.semanticscholar.org/paper/A-Self-Adaptive-Traffic-Light-](https://www.semanticscholar.org/paper/A-Self-Adaptive-Traffic-Light-Control-System-Based-Zaatouri-Ezzedine/3482004fe0998133cdf3b7409659cd710bd35d0d)  
] [Control-System-Based-Zaatouri-](https://www.semanticscholar.org/paper/A-Self-Adaptive-Traffic-Light-Control-System-Based-Zaatouri-Ezzedine/3482004fe0998133cdf3b7409659cd710bd35d0d)  
Ezzedine/3482004fe0998133cdf3b7409659cd710bd35d0d.
- [ E. blog, 10 8 2018. [Online]. Available:  
4 [https://everitt257.github.io/post/2018/08/10/object\\_detection.html#:~:](https://everitt257.github.io/post/2018/08/10/object_detection.html#:~:text=YOLO%20stands%20for%20You%20Only,regression%20at%20the%20same%20time..)  
] [text=YOLO%20stands%20for%20You%20Only,regression%20at%20the%20s](https://everitt257.github.io/post/2018/08/10/object_detection.html#:~:text=YOLO%20stands%20for%20You%20Only,regression%20at%20the%20same%20time..)  
ame%20time..
- [ D. Jurafsky, “23,” 21 7 2021. [Online]. Available:  
5 <https://web.stanford.edu/~jurafsky/slp3/4.pdf>.  
]
- [ P. Vadapalli, 5 1 2021. [Online]. Available:  
6 <https://www.upgrad.com/blog/naive-bayes-explained/>.  
]
- [ K. Goyal, “Machine Learning vs Neural Networks: What is the  
7 Difference?,” 13 2 2020. [Online]. Available:  
] <https://www.upgrad.com/blog/machine-learning-vs-neural-networks/>.
- [ V. Meel, “ANN and CNN: Analyzing Differences and Similarities,” 1 2  
8 2021. [Online]. Available: [https://viso.ai/deep-learning/ann-and-cnn-](https://viso.ai/deep-learning/ann-and-cnn-analyzing-differences-and-similarities/)  
] [analyzing-differences-and-similarities/](https://viso.ai/deep-learning/ann-and-cnn-analyzing-differences-and-similarities/).
- [ S. Kumar, “Artificial-Neural-Networks-ANN-and-Convolutional-Neural-  
9 Networks,” 3 2017. [Online]. Available:  
] [https://www.researchgate.net/figure/Artificial-Neural-Networks-ANN-](https://www.researchgate.net/figure/Artificial-Neural-Networks-ANN-and-Convolutional-Neural-Networks-CNN_fig2_320746968#:~:text=The%20major%20difference%20between%20a,neurons%20as%20shown%20in%20Fig..)  
and-Convolutional-Neural-Networks-  
CNN\_fig2\_320746968#:~:text=The%20major%20difference%20between%20a,neurons%20as%20shown%20in%20Fig..
- [ M. Mandal, “Introduction to Convolutional Neural Networks (CNN),” 1 5  
1 2021. [Online]. Available:

0 [https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-](https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/)  
] [networks-cnn/](https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/).

[ S. Bhuiya, "The Drawbacks with CNN," 1 5 2021. [Online]. Available:  
1 <https://iq.opengenus.org/disadvantages-of-cnn/>.

1  
]

[ Y. VERMA, 10 7 2021. [Online]. Available:  
1 [https://analyticsindiamag.com/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-a-](https://analyticsindiamag.com/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-a-comparative-guide/)  
2 [comparative-guide/](https://analyticsindiamag.com/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-a-comparative-guide/).

]

[ R. Girshick, "Rich feature hierarchies for accurate object detection and  
1 semantic segmentation," [Online]. Available:  
3 <https://arxiv.org/pdf/1311.2524v5.pdf>.

]

[ R. Girshick, "Fast R-CNN," 27 7 2015. [Online]. Available:  
1 <https://arxiv.org/pdf/1504.08083.pdf>.

4  
]

[ S. Ren, "Faster R-CNN: Towards Real-Time Object," 6 1 2016. [Online].  
1 Available: <https://arxiv.org/pdf/1506.01497.pdf>.

5  
]

[ J. Redmon, "You Only Look Once," 9 5 2016. [Online]. Available:  
1 <https://arxiv.org/pdf/1506.02640.pdf>.

6  
]

[ K. He, "Mask R-CNN," 1 4 2019. [Online]. Available:  
1 <https://arxiv.org/pdf/1703.06870.pdf>.

7  
]

[ M. COCO, 2021. [Online]. Available: <https://cocodataset.org/#home>.

1  
8  
]

[ M. COCO, "Microsoft COCO: Common Objects in Context," [Online].  
1 Available: <https://arxiv.org/pdf/1405.0312.pdf>.

9  
]

[ ChilkuriDinesh. [Online]. Available:  
2 [http://www.cs.iit.edu/~cs561/cs450/ChilkuriDineshThreads/dinesh's%20](http://www.cs.iit.edu/~cs561/cs450/ChilkuriDineshThreads/dinesh's%20files/Advantages%20and%20disadvantages.html)  
0 [files/Advantages%20and%20disadvantages.html](http://www.cs.iit.edu/~cs561/cs450/ChilkuriDineshThreads/dinesh's%20files/Advantages%20and%20disadvantages.html).  
]

[ W. Abdulla, "Mask R-CNN for object detection and instance  
2 segmentation on Keras and TensorFlow," 2017. [Online]. Available:  
1 [https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN).  
]

[ Ultralytics. [Online]. Available: <https://github.com/ultralytics/yolov5>.  
2  
2  
]

[ V. Meel, "ANN and CNN: Analyzing Differences and Similarities," 1 2  
2 2021. [Online]. Available: [https://viso.ai/deep-learning/ann-and-cnn-](https://viso.ai/deep-learning/ann-and-cnn-analyzing-differences-and-similarities/)  
3 [analyzing-differences-and-similarities/](https://viso.ai/deep-learning/ann-and-cnn-analyzing-differences-and-similarities/).  
]

[ M. Mandal, "Introduction to Convolutional Neural Networks (CNN)," 1 5  
2 2021. [Online]. Available:  
4 [https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-](https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/)  
] [networks-cnn/](https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/).

[ Y. VERMA, 10 7 2021. [Online]. Available:  
2 [https://analyticsindiamag.com/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-a-](https://analyticsindiamag.com/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-a-comparative-guide/)  
5 [comparative-guide/](https://analyticsindiamag.com/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-a-comparative-guide/).  
]

[ R. Girshick, "Rich feature hierarchies for accurate object detection and  
2 semantic segmentation," [Online]. Available:  
6 <https://arxiv.org/pdf/1311.2524v5.pdf>.  
]

[ R. Girshick, "Fast R-CNN," 27 7 2015. [Online]. Available:  
2 <https://arxiv.org/pdf/1504.08083.pdf>.  
7  
]

[ S. Ren, "Faster R-CNN: Towards Real-Time Object," 6 1 2016. [Online].  
2 Available: <https://arxiv.org/pdf/1506.01497.pdf>.  
8  
]

[ J. Redmon, "You Only Look Once," 9 5 2016. [Online]. Available:  
2 <https://arxiv.org/pdf/1506.02640.pdf>.  
9  
]





## Appendix I

## Appendix II

