The BRITISH UNIVERSITY IN EGYPT

Faculty of Informatics and Computer Science

Artificial Intelligence

Intelligent Transportation System

By: Abdelrahman Walid Zaky

Supervised By

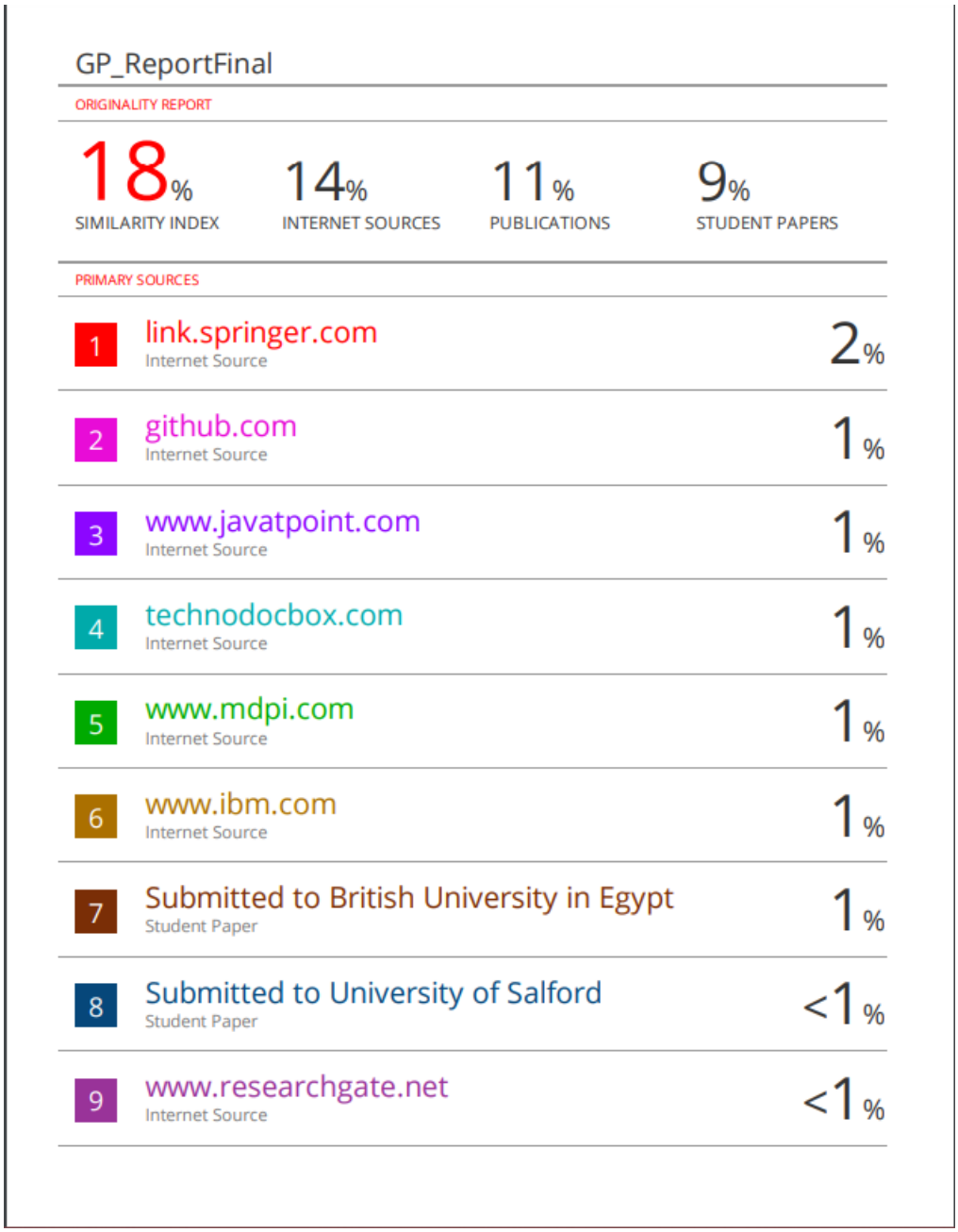
Professor Mohamed Ahmed Hussein

June 2022

Abstract

The growing problem of traffic congestion and the vehicles emissions based on these congestions demands innovative solutions to improve traffic management and enhance overall road safety. By taking advantage of the power of computer vision and deep learning, this system proposes an automated system to monitor and analyze the movement of vehicles in real-time. This system focuses on the application of machine vision techniques for detecting, tracking, and counting vehicles on the road, with the aim of collecting data to develop a flow prediction model that can inform road users about the congestion level. The main goal is to build a reliable vehicle recognition and tracking system that can detect and track vehicles reliably from video data captured by traffic light cameras. By applying state-of-the-art object detection models such as Faster R-CNN or YOLO, the system can efficiently identify vehicles and assign unique identifiers to track them throughout their trajectory. To obtain valuable traffic data, the system incorporates vehicle counting functions to quantify the flow of vehicles passing through specific road sections. To keep track of traffic patterns and vehicle density, this data is then gathered and archived. Using deep learning methods like recurrent neural networks (RNNs) or their extensions like long short-term memory (LSTM) networks or gated recurrent units (GRU), the acquired data is used to develop a traffic flow prediction model. This model aims to predict the traffic flow and estimate congestion levels based on historical data, and time of day. These predictions can then be disseminated to road users allowing them to make informed decisions about their travel routes and potentially avoid congested areas. The proposed system offers numerous benefits, including real-time monitoring of road traffic, accurate vehicle counting, and reliable flow predictions. By alerting road users about congestion levels, it has the potential to prevent traffic congestion, enhance road safety, improve overall transportation efficiency, and lower the overall Carbone Dioxide emissions.

Turnitin Report



GP_ReportFinal

ORIGINALITY REPORT

18%	14%	11%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	link.springer.com Internet Source	2%
2	github.com Internet Source	1%
3	www.javatpoint.com Internet Source	1%
4	technodocbox.com Internet Source	1%
5	www.mdpi.com Internet Source	1%
6	www.ibm.com Internet Source	1%
7	Submitted to British University in Egypt Student Paper	1%
8	Submitted to University of Salford Student Paper	<1%
9	www.researchgate.net Internet Source	<1%

Acknowledge

I would like to express my utmost appreciation to my family, whose unconditional love, support, and encouragement are a constant source of motivation, enabling me to overcome challenges along the way throughout my academic journey.

I extend my heartfelt appreciation to my professor Mohamed Ahmed Hussein, whose guidance, expertise, and insightful feedback have been very valuable in shaping this project.

I would also love to thank my professors throughout the years as their dedication to teaching has inspired me to strive for academic excellence.

I am grateful to my friends for their encouragement and support. Their friendship has made this journey even more enjoyable and memorable.

Table of Contents

List of Figures

List of Tables

1. Introduction

1 Introduction

According to the Environmental Protection Agency in the USA, a personal vehicle emits around 4.6 tons of carbon dioxide per year, that is assuming that the average petrol car drives about 18,000 kilometers per year [1]. In Egypt alone, there are 10.9 million registered vehicle, 2.2 million of these registered cars are registered in Cairo [2]. Egypt is ranked 116 with the highest road traffic accidents in the world. There are forty-two people out of 100,000 that fell victim to a road traffic fatality. Road traffic injuries are also responsible for 2.4 per cent of the people that suffer from disability every year [3]. Intelligent Transportation Systems aims to improve the traffic flow by integrating the cars on the street with each other's, with the street cameras all around the city.

1.1 Overview

By using the cameras that are on the roads as input for the object detection model, the model can process the input, and classify the data, whether it sees a pedestrian, a personal vehicle, or a bus and then store this data. This data is the number of vehicles that pass in 5 minutes intervals, the type of vehicle. The number of vehicles every 5 minutes will be fed to a traffic flow prediction model, the model will keep predicting traffic flow and if the model predicts a huge number of vehicles in the next interval of time, the model can suggest choosing alternative route accordingly. Since data is an asset, other data can also be used in some data analysis and can be used in other systems and solutions as well.

1.2 Motivation

In urban places, traffic congestion has become a serious problem that is detrimental to both overall quality of life and transportation efficiency. Traditional methods of traffic monitoring often suffer from limitations such as manual data collection and delayed information transportation. This project seeks to address these challenges by using machine vision techniques to create an automated system for vehicle detection, tracking, traffic data gathering, and flow prediction. By accurately detecting and tracking vehicles, collecting comprehensive traffic data, and utilizing deep learning algorithms, real-time insights can be provided into the congestion levels. The motivation for this project lies in the potential to improve traffic management systems, enhance travel experiences, and contribute against climate change by utilizing machine vision technology to address the issue of road congestion.

1.3 Problem Statement

Given a video stream, the system should detect the vehicle type. A traffic flow predictor should use the data coming from the video stream to predict the flow of the traffic in the next interval of time, and act accordingly.

1.4 Scope and Objectives

The system focuses on developing a system for vehicle detection, tracking, data collection, traffic flow prediction using machine vision and deep learning techniques.

The system will process video data obtained from traffic light cameras to identify and classify the type of vehicles present in the footage. The primary objective is to accurately detect various types of vehicles, such as cars, buses, trucks, and motorcycles. The system will track the detected vehicles across consecutive video frames to monitor their movement throughout the scene. The tracking mechanism aims to assign unique IDs to vehicles and maintain their identities throughout the video sequence. The system will collect relevant traffic and road conditions information, such as vehicle counts, and congestion levels. This data collection process aims to build a comprehensive dataset for analysis and model training. The system will store the collected video data, vehicle information, and traffic-related data. The storage system will ensure data integrity and accessibility. Using the collected data, the system has a traffic flow prediction model. The model will utilize deep learning techniques, to forecast traffic conditions and estimate congestion levels. Based on the traffic flow predictions, the system will send whether the road is congested or not and recommends taking the road or not accordingly. The recommendation mechanism will consider factors such as predicted congestion levels to guide users in making informed decisions about their travel routes.

This creates a more dynamic traffic flow and improves the safety of the individuals while reducing the vehicles emissions and waste by improving the traffic flow.

1.5 Report Organization

This report is organized into the following chapters to provide a comprehensive understanding of the research project, Introduction, to introduce the problem of road congestion and the importance of traffic management systems. It outlines the objectives of the research and provides an explanation for utilizing machine vision techniques. Then the background, where papers and literatures related to Intelligent Transportation System, Artificial Intelligence Techniques, Vehicle Detection, and Traffic Flow Prediction are reviewed, discussing the strengths, limitations, and keeping up to date with all the possible approaches. Then the proposed architecture and methodology used for vehicle detection, tracking, and traffic flow prediction are presented in this chapter. In addition, a block diagram to display the flow of the system and how it works. The following chapter presents the details of the implementation of the proposed system. And how the software components were implemented and used. Explaining how the vehicle tracking, data collection process, and traffic flow prediction was implemented, and the challenges encountered during the implementation and how it was overcome. The following chapter evaluates the proposed and implemented system and whether it successfully implements the functional requirements with proposed scenarios on how the system works. And finally, the last chapter summarizes the findings and impact of the proposed system and discusses the area of improvement and limitations found for the future work.

1.6 Work Methodology

This system attempts to create a more dynamic traffic flow by incorporating Machine Vision to understand the traffic flow and Deep Learning to predict it.

1.7 Work Plan

The work plan for successfully research and implementing the system.

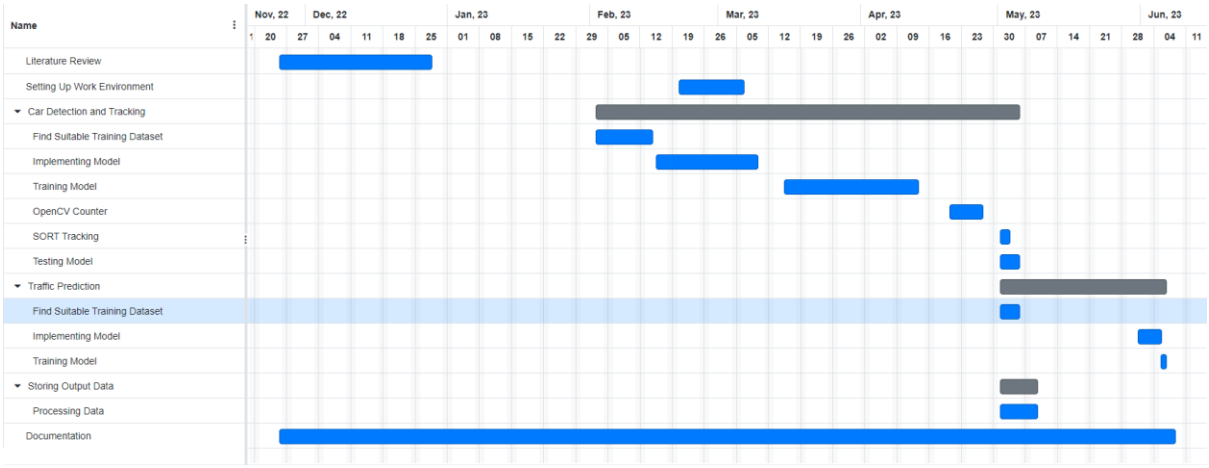


Figure 1, Gantt Chart for planning the system

2. *Background*

2 Background

In this chapter, the existing literature and research related to traffic monitoring, vehicle detection, and traffic flow prediction are reviewed. It discusses the strengths and limitations of previous approaches.

2.1 Early Days

Intelligent Transportation System first external infrastructure was in project Prometheus, in 1988. Both projects were critical in starting the development of ITS [4]. Prometheus aimed to increase the safety of road traffic, as well as capability, efficiency, and comfort. Pro-Road was a sub-project aimed on communication between infrastructure and vehicles for optimizing the road traffic using relevant sensors such as radars, laser scanner, and cameras. In 1992, PATH was aimed at forcing and influencing control of vehicles on highways for optimizing the traffic flow. The project was not accepted much by the population.

2.2 The Next Generation

Instead of trying to enforce the participants directly, projects went down the path of partially controlling and optimizing the general traffic flow, using Intelligent Traffic Light logic to control the vehicle instead. In 1998, ITS was able to analyze specific traffic situations. In 2013, the Compass4D project was installed in some European cities aiming to develop a system to provide warnings to road hazards. ITS introduced a situation adapted traffic light, whereas the cars arrived the traffic light would give the vehicles a speed recommendation to aid with the waiting time [4].

In 2016, Intelligent Cooperative Sensing for Improved Traffic Efficiency was implemented in some other European cities, it aimed for a highly scalable distributed ITS for data analysis. The mission was to develop and optimize the traffic flow and reduce emissions. In 2018, project FED4FIRE+ was created for the communication between vehicles and other vehicles or infrastructures, the project used antennas and cloud solution connected via 5G internet, and the system achieved a latency of 10-15 MS [4].

2.3 Intelligent Transportation Systems

ITS meaning alters and differs from one project to another and one paper to another, there are some projects the apply the concept of graph theories and find the minimized path, others are more focused on Real-Time Systems and Data Analysis. This project, however, is more focused on the Real-Time of the Intelligent Transportation Systems. Real-Time System applications such as traffic density.

2.3.1 Real-Time Systems

Real-Time Systems manages to take input of the present situation through video or sensors and deals with it.

A Real-Time optimization model was used in a paper, which had the model to take into consideration the traffic scenarios including pedestrians [5]. Another paper calculated the time it takes the vehicle to reach the intersection from a particular point, using sensors. Various calculations are made on the data to find the green light length to minimize the time a vehicle waits in traffic [6]. Another paper used Real-Time data to monitor the present traffic flows to control the traffic in a more convenient way using CCTV cameras [7]. Another paper Recommended a distributed wireless network of vehicular sensors to get a replicated view of the actual scenario to lower the congestion without taking decisions in Real-Time [8]. While other paper used hybrid Neural Network based multiagent system in solving large-scale traffic signal control problem, reducing the delay of each vehicle by 78% and the stoppage time by 85% [9].

2.3.2 Traffic Density

This paper proposed an Intelligent traffic Light that gather information density from the passing vehicles and dispatch these statistics back to other vehicles to avoid congestions, paper assumed that the vehicle was equipped with GPS, on board unit, and full map of the city including the exact coordinates of each traffic light [10]. Other paper used the concept of adaptive traffic light control algorithm, which manipulates the length of the traffic light according to the detected traffic, algorithm used Real-Time data such as volume of the traffic in each lane and waiting time of vehicles. To determine the traffic light sequence and optimal length of green light and bring down the average waiting time [11]. Other papers used IR sensors to determine the density of the traffic based on which of the traffic signals were updated to provide a smooth traffic flow [12]. This paper had an interesting approach to determine the number of vehicles using a weight sensor and used a programmable logic controller to analyze the data [13].

2.3.3 Path Optimization Technique

A paper was proposed to find an optimized path for transportation using the Ant Colony Optimization concept, and once an optimized path was found, they added some more features for more

convenience and less traffic jams [14]. Another paper used video surveillance for realizing the Real-Time scenario. The paper dealt with the decreasing response time of the emergency cars by establishing communication between emergency cars and traffic lights. Data was collected in Real-Time and was used to determine the traffic density using path optimization techniques [15]. Another paper used video surveillance and neural networks to reduce the traffic stress across the network [16].

2.3.4 Data Analysis System

2.3.4.1 Green Light Optimization

The paper gave a solution to minimize the waiting time of vehicle by testing the problems of traffic lights, paper used particle swarm optimization, ant colony optimization, and genetic algorithm which proved to be the best and the most important of them all [17]. Other paper proposed a model using infrared proximity sensor to collect data from the lane to fetch it to the microcontroller, the sensors are on each lane, if the sensor gave a low reading, then it meant that the traffic is low in that lane and gives the priority to the other lane instead [18]. Another paper presented a MATLAB simulation of a traffic controller for controlling the traffic flow in an intersection. The controller controls the traffic lights timing and sequence to ensure a smooth traffic flow [19]. Another paper proposed a monitoring system in addition to a traffic light system to determine the street case whether it was empty or crowded, with different weather conditions. Using associative memory depending on the stream of images extracted from the street video recorders [20]. This paper optimized traffic signals by focusing on three classes, ambulance, priority vehicles such as police vehicles, and normal traffic. Providing stoppage free path for the ambulances and preventing traffic congestion. Also managed the traffic density by increasing the duration of green light of the lane where the density was relatively high [21].

2.4 Artificial Intelligence Techniques

Artificial Intelligence is a branch in Computer Science that focuses on automating or creating machines that can perform some tasks that require a human presence such as speech recognition, visual observations, or decision making. Artificial Intelligence should be able to think like humans, and act like them too. Artificial Intelligence technologies such as Rule-Based Systems, Machine Learning, Deep Learning, Natural Language Processing, and Machine Vision.

Rule-Based Systems are systems that use a set of predefined rules to make decisions. These are more suited to small data applications, as they do not react well with big data and are useless outside of the set parameters defined, but they are highly effective for that one purpose application.

Machine Learning systems use algorithms that allow the machines to learn from the data given with no prior defined rules, instead the machine identifies the patterns in the data and sets its own rules and makes predictions or takes actions based on these patterns and rules. Machine Learning can have supervised learning or unsupervised learning. Machine Learning has plenty of algorithms such as Decision Trees, Logistic Regression, K-Nearest Neighbors, Artificial Neural Networks.

Deep Learning is a subset of Machine Learning that uses Artificial Neural Networks, it analyzes and learns data to be able to recognize patterns to perform human-like tasks such as image classification, and speech recognition. Deep Learning algorithms such as convolutional neural networks, CNN, and recurrent neural networks, RNN, Long Short-Term Memory Networks, LSTM, Generative Adversarial Networks, GAN.

Natural Language Processing focuses on giving the machine the ability to comprehend and understand human language, with applications including speech recognition, chatbots, and machine translation. NLP uses Deep Learning techniques such as RNNs, and Transformers.

Machine Vision enables the machine to understand visual data, making it able to analyze images and videos. Machine Vision can achieve tasks such as object detection, object recognition, and image captioning. Commonly used Deep Learning techniques such as CNNs, and GANs.

2.4.1 Rule-Based Systems

Rule-Based Systems are systems that use if-then conditions to reach a certain decision or conclusion, the if-then conditions are predefined rules that the system follows. These rules are modified by experts in the field, each rule consists of conditions “if” that when met, action “then” is applied. Rule-Based Systems are used in small data applications. As they are specifically conditioned systems, they cannot work on other systems with the same rules. Some applications on Rule-Based Systems such as XCON, which was used to configure their VAX computer systems, and MYCIN, which was developed to diagnose blood infections, using if-then decision rules to mimic the human ability of decision making.

2.4.2 Machine Learning

Machine Learning is a computer program that learns from experience to perform tasks, that then measures the performance of said task.

Deep Learning is under the umbrella of Machine Learning based on Artificial Neural Networks. Algorithm Varies when talking about Machine Learning and Deep Learning as there is supervised learning which assumes a function from the training data, consisting of an input, and a classified output, and there is unsupervised learning where the algorithm does not classify the information without guidance.

2.4.2.1 Learning Types

Supervised Learning and Unsupervised Learning are machine learning techniques which involve how the model trains on the data.

Supervised Learning is training the machine learning model on a labeled dataset, each datapoint is associated with a target variable, and the goal of the supervised learning is to find that connection between the input features and the target variable. It is evaluated based on how good it can predict the output for unseen data. Examples such as Logistic Regression, Decision Trees, and Neural Networks.

Unsupervised Learning is training the machine learning model on an unlabeled dataset, and the goal is to find that pattern, or relationship with no predefined target variable. It is evaluated based on how well the model discovered patterns in the data. Examples such as K-means clustering and autoencoders.

There are numerous types of Machine Learning techniques, such as Decision Trees, Logistic Regression, K-Nearest Neighbors, Artificial Neural Networks, convolutional neural networks, CNN, and recurrent neural networks, RNN.

2.4.2.2 Decision Tree

Decision Tree is a supervised learning algorithm, used in classification and regression. It uses a tree structure with a root node, decision nodes, branches, and leaf nodes.

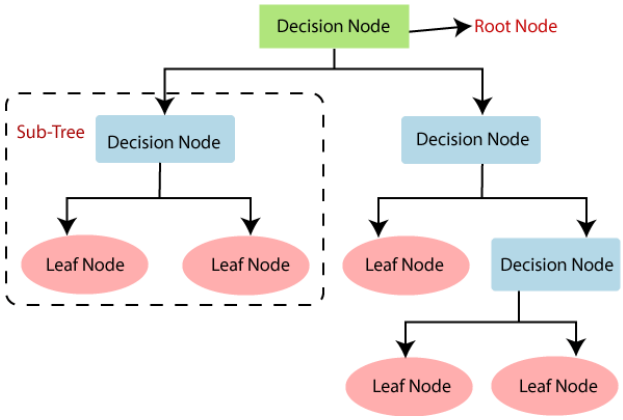


Figure 2, Decision Tree

It is one of the first learning algorithms to be developed. The advantages to it is that it is easy to interpret, it uses Boolean logic makes it easier to understand and to use, it can be very flexible for both classification, and regression tasks, with some insensitivity to the relationships between attributes which makes it easier as it needs no data preparation, can handle missing values by using splits or imputing the missing values, and can be used in ensemble methods such as random forests. The disadvantages of it are that when the decision trees get complex, they tend to have a major overfitting problem with a big high variance problem, but it can be made slightly better using pruning methods, and bagging for the high variance nonetheless, these solutions are very limited as it can affect the data and the predictions. Decision Trees are overly sensitive to small data making it inconsistent. Decision Trees are very costly as they take a greedy search method to construct the best tree possible, making it hard to train the algorithm as it takes way too much time. Also, Decision Trees cannot deal with complex relationships making it less effective with some of the bigger datasets with more relations between the variables.

2.4.2.3 Logistic Regression

Logistic Regression is used for classification and regression tasks, it models the probability of a binary dependent variable based on one or more independent variables, with an output of a probability value between 0 and 1. That differs from Linear regression which models the relationship between a continuous dependent variable and one or more independent variables to estimate the value of the dependent variable based on the values of the independent variables, with an output of a continuous numerical value representing the predicted value of the dependent variable.

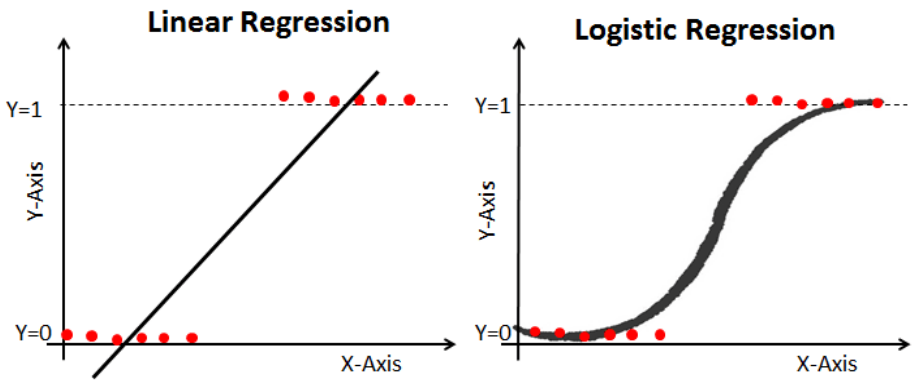


Figure 3, Difference between the output of Linear Regression and Logistic Regression

Its advantages are that it is fast as classifying records, it interprets the model coefficients as indicators of feature importance, can manage noise in the input data without affecting the performance of the model, stable with new and unseen data preventing it from overfitting. The disadvantage is that it cannot obtain complex relationships, it also cannot solve nonlinear problems as it assumes linearity and has a linear decision surface which makes it unable to handle non-linear relationships limiting its performance in some cases, requires labeled data, does not perform well with unbalanced data.

2.4.2.4 K-Nearest Neighbors

K-Nearest Neighbors is a supervised learning algorithm that is used for classification tasks, it uses the adjacency of the data point to cluster it with the nearest group. The algorithm uses distance metrics such as Manhattan distance and Euclidean distance to determine which data points are the closest to the given point. While K is the number of neighbors to determine the closest data point.

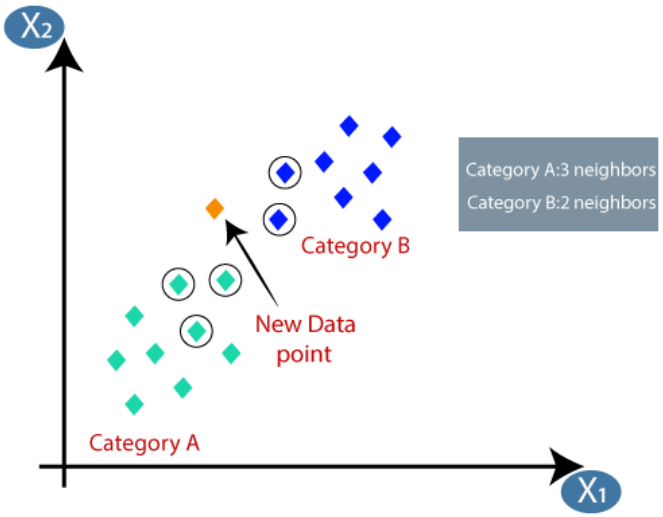


Figure 4, KNN Algorithm, with K=5

The advantages of the K-Nearest Neighbor are the ease of implementation and the lack of hyperparameters as the algorithm only has two required parameters, the K value, and the distance metric. It also adapts well with new training sets, as it has no training time. It is very flexible as it can work with linear, non-linear data, can be used with classification and regression, and can manage multiclass classifications. The disadvantages of it is that it is prone to overfitting, as the algorithm does not perform well with high dimensional data, a way to overcome this problem can be with some dimensionality reduction, the value of K can also overfit the data as if it is too small or underfit under greater values of K, sensitive to irrelevant data, calculating the distance between the points can be exhaustive and computationally expensive with large datasets.

2.4.2.5 Support Vector Machines

Support Vector Machines are used in classification and regression tasks, SVM goal is to find the best hyperplane which separates the data into different classes. The hyperplane maximizes the margin, the distance between the hyperplane and the closest data points from each class. The data points that are closest to the hyperplane are called support vectors.

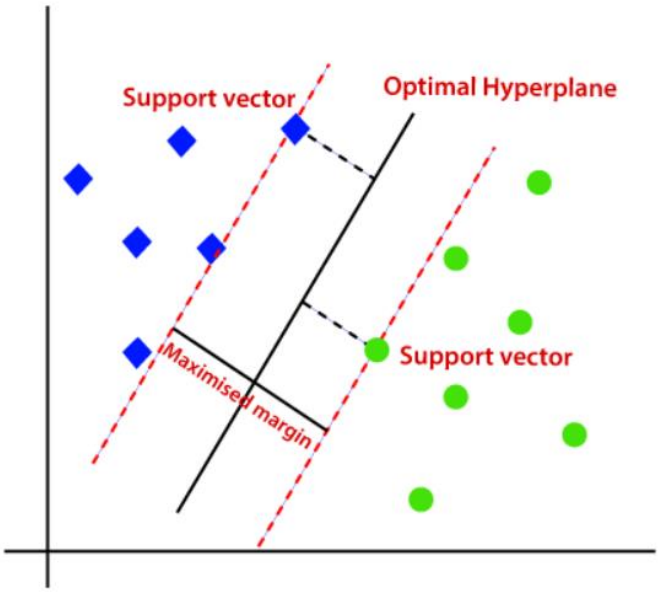


Figure 5, Support Vector Machines

Advantages of the Support Vector Machine are that it can manage high dimensional data effectively, making it a good option for tasks such as image classification, and it is less prone to overfitting compared to the other techniques such as Decision Trees. Disadvantages such that it is computationally expensive to train, also sensitive to the hyperparameters making it a difficult model to tune.

2.4.2.6 Artificial Neural Networks

Artificial Neural Networks are a subset of Machine Learning and the main foundation of deep learning. ANNs are composed of node layers, an input layer, hidden layers, and output layer. Each of the nodes are connected to one another with their weights and biases.

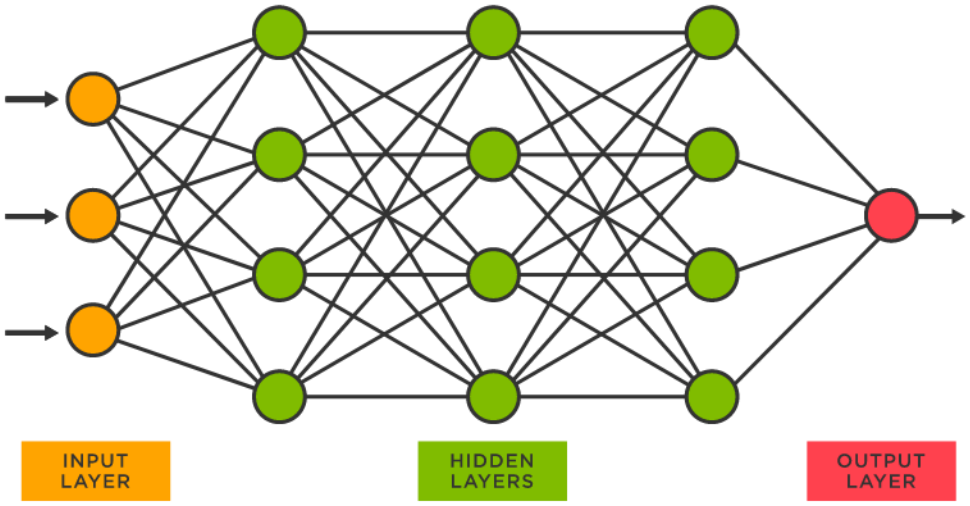


Figure 6, Artificial Neural Network layers

The network relies on training data and adjusting the weights and biases by doing forward propagation and backward propagation. A lot of neural network types are available, such as Multi-Layer Perceptrons, Restricted Boltzmann Machines, and Hopefield. Its advantages are that they have a distributed memory and that it stores the information across the entire network, which allows it to have a fault tolerance in case of unfortunate events occurring. It also can work greatly with incomplete and insufficient knowledge of the data, also has great adaptability with other data which makes it suitable for more than one application. While having disadvantages such as high hardware dependence, hyperparameter tuning such as number of nodes in every layer, number of layers making it a time-consuming process to achieve the best parameters for the best results, and some behavior might be unknown on why it happened exactly.

2.4.2.7 Convolutional Neural Networks

Convolutional Neural Networks is a feedforward network that is often utilized in computer vision tasks, and image processing. It has three main types of layers, Convolutional layer, pooling layer, and fully connected layer. With earlier layers focusing on simple features such as edges, and as the image progresses through the layers it starts to recognize the elements more until it identifies the intended object.

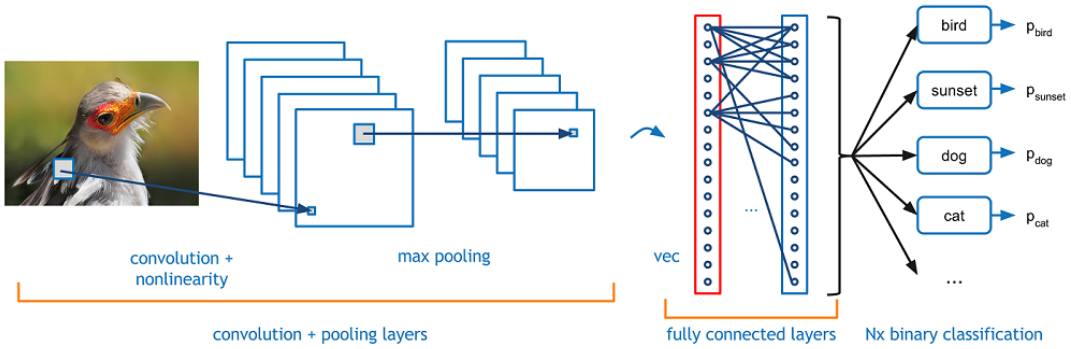


Figure 7, Convolutional Neural Networks

Convolutional Layer is where most of the computation occurs, it has some components such as input data, a filter, and feature map.

The filter moves across the fields of the image to check if the features are present. It is a two-dimensional array of weights representing the image. The filter is applied to an area of the image, and a dot product is fed into the output array, then the filter shifts to repeat the process until the kernel is swept across the entire image. The final product from the dot products is known as a feature map.

Pooling Layer performs some dimensionality reduction on the number of parameters in the input. Then the filter sweeps again across the input, but with no weight. Rather, the kernel applies an aggregation function to the values to the output array.

2.4.2.7.1 Types of Pooling

Max Pooling, as the filter moves, it selects the pixel with the maximum value to send to the output array. This approach is used more often than the average pooling.

Average Pooling, as the filter moves, it calculates the average values to send to the output array.

Fully Connected Layer, in the previous layers, the values of the input image are not connected directly to the output layer, but this layer is fully connected layer, each node in the output layer connects directly to a node in the previous layer.

The layer classifies based on the features extracted through the previous layers and their different filters.

CNN's advantages such as having a very high accuracy in image recognition problems, automatically extract the needed features from the input data, and weight sharing, and minimizing computation, its disadvantages such as requiring a lot of training data, much slower than the other

techniques, takes a lot of time to process, and it is only designed for machine vision tasks and does not suit other tasks such as Natural Language Processing.

2.4.2.8 Recurrent Neural Networks

Recurrent Neural Networks is a type of neural network that is often used in the natural language processing area, such as speech recognition. RNNs are special with their memories as it takes the information before the inputs to influence the current input and output, and it does not assume that the input and outputs are independent as the deep learning networks often assume. It takes the output and feeds it back into the model which is called Backpropagation.

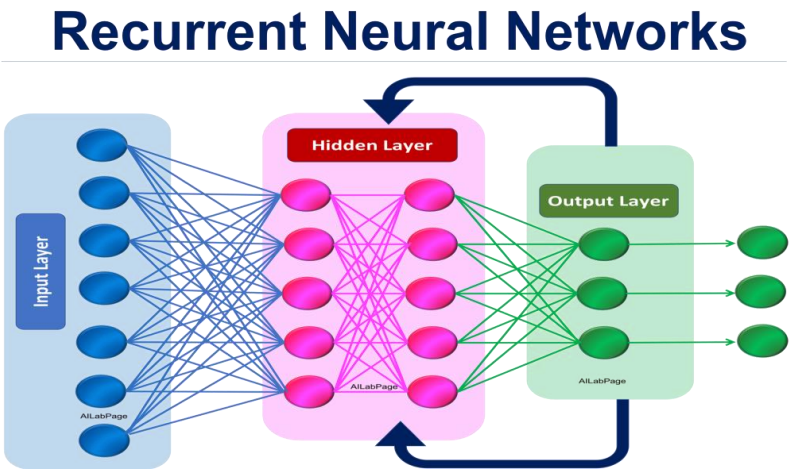


Figure 8, Recurrent Neural Network

Its advantages are that it remembers every information and previous inputs in its memory, it can understand the context of the input, and can generate text using language modeling. Its disadvantages are that it is hard to train RNN and has problems with the gradient vanishing and exploding when performing, RNNs can have some issues with the memory capacity which might be a problem for the model to remember earlier input.

2.4.3 Natural Language Processing

Natural Language Processing is a way in which the computer can comprehend and process the human language. It combines modeling of human language with statistical, machine learning, and deep learning models. NLP can be used in a wide range of tasks, such as text classification, speech recognition, machine translation, and text processing.

Natural Language Processing has many techniques, as it can be used with deep learning models to classify and label data whether it is voice data or text data, using techniques such as RNN, and Transformers, which allows the NLP to learn as it receives more data.

2.4.3.1 Transformers

Transformers are a deep learning algorithm that is being used in a variety of NLP tasks, it is a contrast between RNNs and CNNs. Transformers consist of an encoder and a decoder trained on a task, the encoder takes an input implements feedforward neural network to output a hidden representation of the input, the decoder generates the output regarding the hidden representation using feedforward neural network.

2.4.4 Machine Vision

Machine Vision can capture and analyze visual information, where the visual information is often captured with a camera, and then used on the computer as an input. Where image processing methods are used, there are many image processing methods such as pattern recognition, segmentation, edge detection, object detection, barcode reading, and said image processing can be implemented using Deep Learning methods such as CNNs, and GANs.

2.4.4.1 Generative Adversarial Networks

Generative Adversarial Network is a deep learning architecture consisting of two neural networks, a generator, and a discriminator. The generator has a goal to generate a realistic data sample that is somewhat comparable to the training data, the discriminator tries to categorize the real and the generated sample. The generator will get feedback from the discriminator on the quality of the generated samples, the generator uses said feedback to improve the performance.

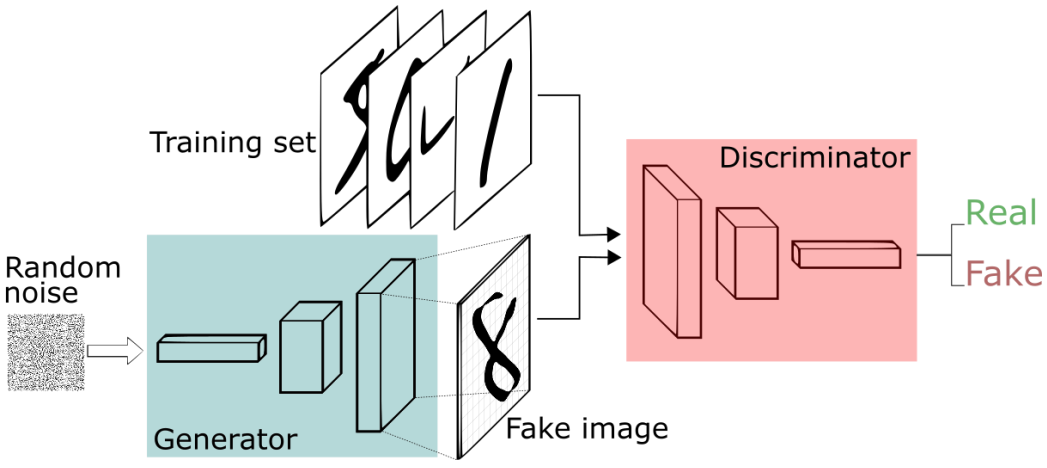


Figure 9, Generative Adversarial Network

GANs advantages are that it can generate high quality realistic data that can be used in generating all sorts of data such as images, videos, or audio. However, GAN is exceedingly difficult to train and uses a large amount of training data.

There exists a detector called YOLO (You Only Look Once) that uses CNNs with amazing results. CNN can be implemented in many languages such as C, Java, and python, but the preponderance of people implements it in python, as they can use frameworks such as Pytorch, TensorFlow, and Keras. Pytorch is used widely in the image processing field as it has better training times than the other frameworks.

Model	Advantages	Disadvantages
Decision Tree	a. Easy to understand. b. Can manage both classification and regression tasks. c. Can manage missing values. d. Can be used in ensemble models	a. Prone to overfitting b. Sensitive to small data c. Costly to construct the best decision tree. d. Cannot deal with complex relations
Logistic Regression	a. Simple, easy, and fast model to implement and train. b. Manages noises in the data. c. Standardized model preventing it from overfitting with new data	a. Cannot obtain complex relationships. b. Assumes Linearity c. Requires labeled data. d. Does not perform well with unbalanced data
K-Nearest Neighbor	a. Easy to implement. b. Has no training time. c. Manages multiclass classification	a. High dimensionality can lead to overfitting. b. Sensitive to irrelevant data c. Computationally expensive with large datasets
Support Vector Machines	a. Manages high dimensional data. b. Less prone to overfitting	a. Computationally expensive to train b. Hyperparameters sensitivity
Artificial Neural Networks	a. Distributed memory and processing b. Robust with incomplete data c. Great adaptability with data	a. High hardware dependance b. Hyperparameter tuning
Convolutional Neural Networks	a. High accuracy in image recognition b. Automated Feature Extraction c. Weight Sharing d. Reduced Parameters and minimizing computation	a. Requires a large amount of training dataset. b. Slower than the other techniques c. Domain Specific, only for machine vision applications
Recurrent Neural Networks	a. Maintaining the memory to remember inputs. b. Context understanding c. Generative text	a. Expensive training b. Gradient Vanishing c. Gradient Exploding d. Memory Capacity
Transformers	a. Manage long-range dependencies more effectively than RNN. b. More effective training than the RNN c. Does not suffer from Gradient Vanishing, or Gradient Exploding problems	a. Requires large amount of data to train. b. Expensive to train.

Generative Adversarial Network	a. Generates high quality, realistic data. b. Model is drastically improving with every iteration as both the generators and the discriminator's performances improve	a. Difficult to train. b. Uses large amount of data to train.

Table 1, comparing all the above techniques' advantages and disadvantages

2.5 Car Detection

As reported previously, Machine Vision techniques vary, as Support Vector Machines could be used, Convolutional Neural Networks, Neural Networks, YOLOv3, and YOLOv7. To determine which one is best used in the car detection area, this literature review will be comparing several papers that used different methodologies in car detection.

2.5.1 Paper using SVM-Classifier based HOG.

In paper [22], the training was done with Gradient Histogram to extract the characteristics of the images, this technique counts occurrences of gradient orientation in localized portions of an image. And after the training, to classify and process the image, Support Vector Machines technique was used. This method was classifying two classes which separate the positive and the negative examples, ensuring that the margin between the nearest positive and negative is maximal.

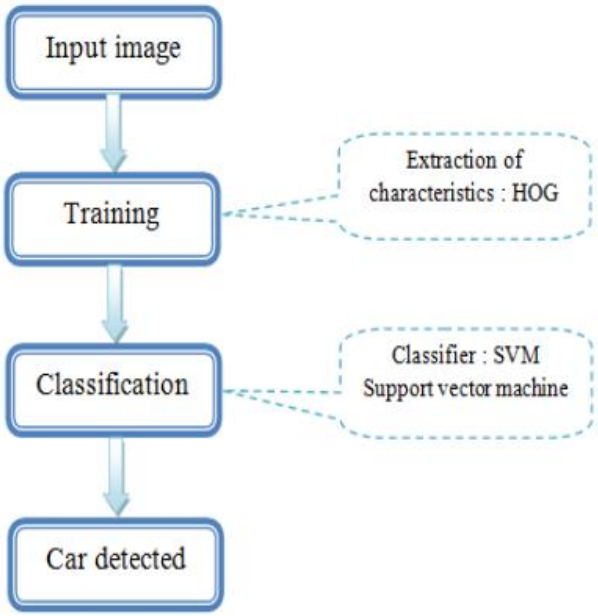


Figure 10, Vehicle Detection Architecture using HOG and SVM in paper [22]

The combination between the Gradient histogram and the Support Vector Machines was applied as Support Vector Machines use a large quantity of examples as inputs, which requires large matrix calculations. The experimental results showed that the vehicle detection detected with accuracy of 83%. Some false objects in the road were detected with a rate of 10%.

Images	Successful Vehicle Detection	False Vehicle Detection	False Objects Detected
122	83%	17%	10%

Table 2, Showing the results of the SVM model [22]

2.5.2 Paper using Deep Learning Approach

In paper [23], which used a combination between Convolutional Neural Networks, and Support Vector Machines, was based on four steps. Firstly, the image is over-segmented into a set of regions by means of the Mean-shift algorithm which is common practice in CNNs. The second step was the feature extraction process using pre-trained CNN. Third was the SVM classifier that is trained to classify “car” or “no-car” classes. The result is a binary map representing a segmentation of the UAV image into “car” and “no-car”. Finally, the binary map is tuned by the inspection of the isolated cars and the morphological operations.

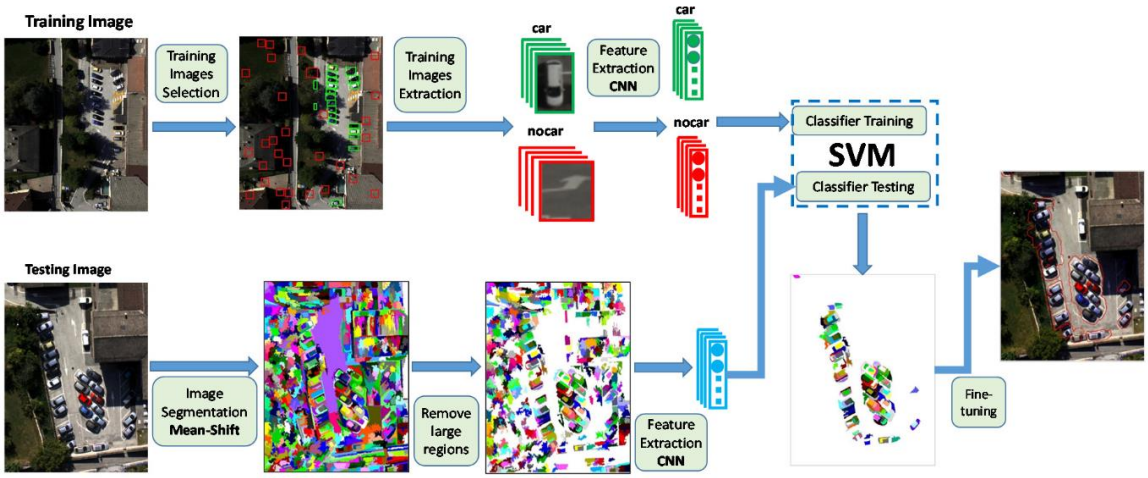


Figure 11, Vehicle Detection Architecture using CNN and SVM in paper [23]

How the fine-tuning works and why is it there, the paper had an observation on the results as the True Positives rate was high, the False Positive rate was relatively high as well due to small, isolated regions of the images where it had car-like signs.

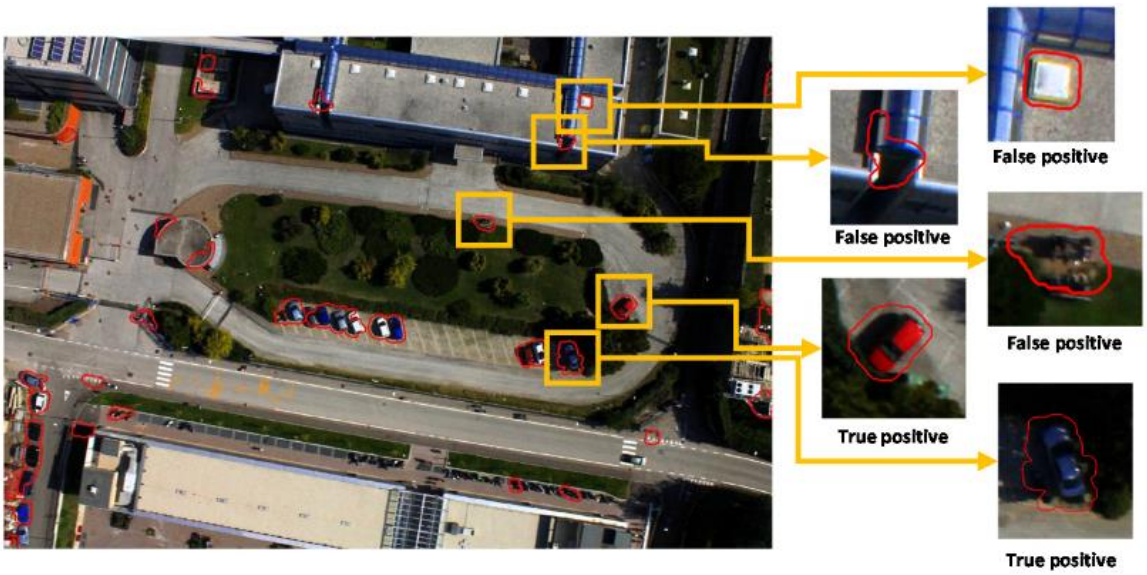


Figure 12, Sample of the True Positives, and the False Positives in the results [23]

.

Which then forced them to use horizontal and vertical rectangular windows that would further fine-tune the location of the car in isolated regions which do contain cars. For regions that do not contain a car, using such windows may break the car-like signature and remove the false positive.

The training was done on total of 136 positive training, and 1864 negative training with an average accuracy result of around 80%.

Test Image	Size	Regions	Within Range	Cars Present	TP	FP	Pacc	Uacc	Acc
Image 1	2626 × 4680	4666	4006	56	47	4	83.9%	92.2%	88.0%
Image 2	2424 × 3896	3114	2483	31	26	10	83.9%	72.2%	78.0%
Image 3	3456 × 5184	4623	1282	19	14	7	73.7%	66.7%	70.2%
Image 4	3456 × 5184	6473	2165	16	13	0	81.3%	100.0%	90.6%
Image 5	3456 × 5184	6666	1994	5	4	2	80.0%	66.7%	73.3%

Table 3, Showing a sample of the results from the CNN-SVM model [23]

2.5.3 Paper using R-CNN and YOLOv3

In paper [24], Faster R-CNN and YOLOv3 were selected for comparison, Faster R-CNN is a model divided into two modules. Region Proposal Network and a fast R-CNN detector, RPN is a fully convolutional network. The RPN and the Fast R-CNN detector share the same convolutional layers. YOLO contains 24 convolutional layers followed by 2 fully connected layers. YOLO divides the input image into an S × S grid, grid cell can only be associated to one object, and it can only predict a fixed number B of boundary boxes. Each box is associated with a one box confidence score.

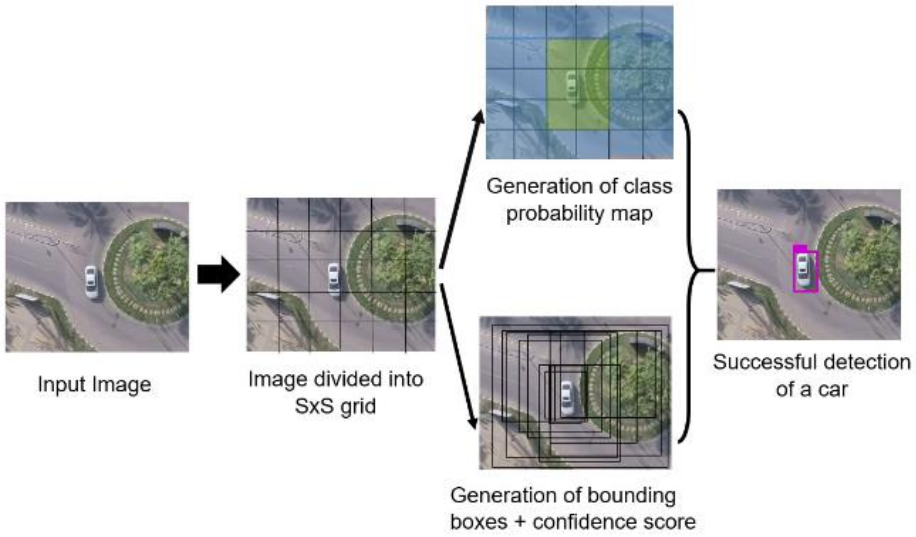


Figure 13, YOLO Architecture [24]

The training set contained 218 images and 3635 instances of labeled cars. The test set was 52 images and 737 instances of labeled cars. While Faster R-CNN had good accuracy, YOLOv3 had better scores, accuracy, quality, and faster processing time.

Measure	True Positives	False Positives	False Negatives	Precision	Sensitivity	F1 Score	Quality	Processing Time
Faster R-CNN	578	2	150	99.66%	79.40%	88.38%	79.17%	1.39s
YOLOv3	751	2	7	99.73%	99.07%	99.94%	98.81%	0.057ms

Table 4, Comparing between R-CNN and YOLOv3 [24]

2.5.4 Paper using YOLOv7

In paper [25], YOLOv7 was used in comparison with all the other YOLO frameworks,

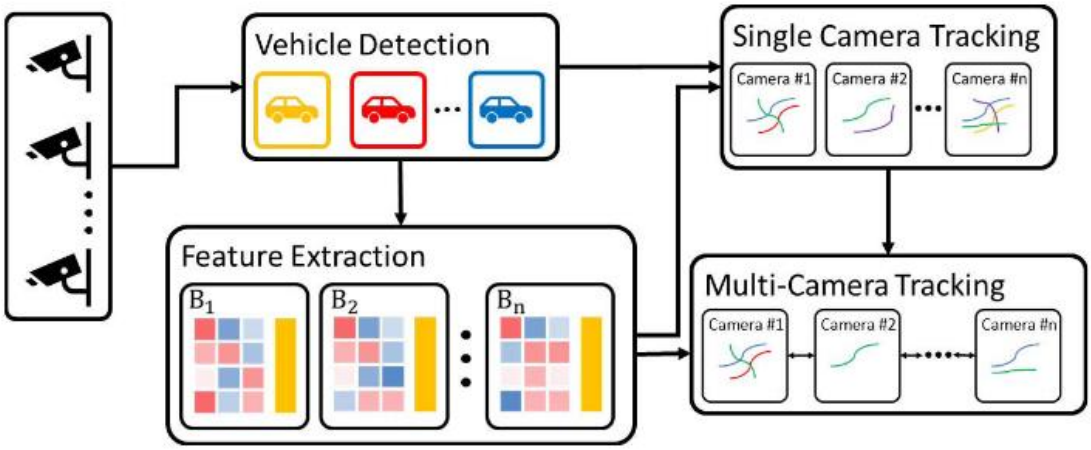


Figure 14, Pipeline of Paper [4] using YOLO [25]

YOLOv7 is the latest version of YOLO family with many upgrades and is considered the best object detection algorithm yet as it is 120% faster than the nearest YOLO algorithm.

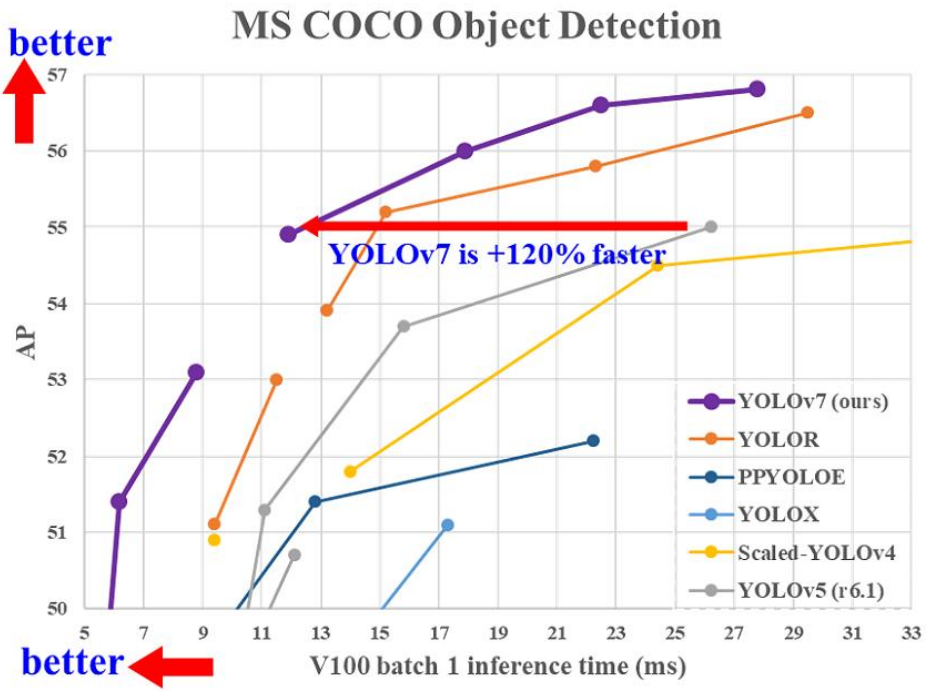


Figure 15, Comparison between the other YOLO algorithms [26]

2.5.4.1 How YOLOv7 Works

YOLOv7 is an object detection model which is trained from scratch on the MS COCO dataset [26], the model exceeded in both speed and accuracy models such as R-CNN by 509% in speed and 2% in accuracy, and outperformed models such as YOLOv5, and YOLOR as shown in figure 14.

YOLOv7 starts with a backbone network, which is a Convolutional Neural Network that processes the input image and extracts high level features. The backbone network uses repConv combining Convolutional Layers 3x3 convolution, Spatial Reduction Layer 1x1 convolution, and an identity connection in one convolutional layer, unlike YOLOv3 and YOLOv4 that used Darknet. YOLOv7 implements a Feature Pyramid Network to capture multi-scale features at different levels of the network, which enhances the ability of the model to detect various sizes and feature extraction. FPN creates a Top-Down Pathway to propagate information from higher resolution feature maps to lower resolution feature maps, with batch normalization in conv-bn-activation topology connecting batch normalization layer directly to convolutional layer, to integrate the mean and variance of batch normalization into the bias and weight of the convolutional layer. The pathway involves up-sampling the feature maps and is done by concatenating. The Detection Head is responsible for predicting the bounding boxes and class probabilities for object detection in the image. It consists of Convolutional Layers and Spatial Reduction Layer. YOLOv7 has an Objectness Loss which uses Loss Function Binary Cross-Entropy to evaluate how well the model predicts an object, the objectness loss measures the confidence score of the bounding boxes predictions. A Localization Loss which uses Mean Squared Error measuring the difference between the bounding box and the ground truth annotation (x, y, width, height).

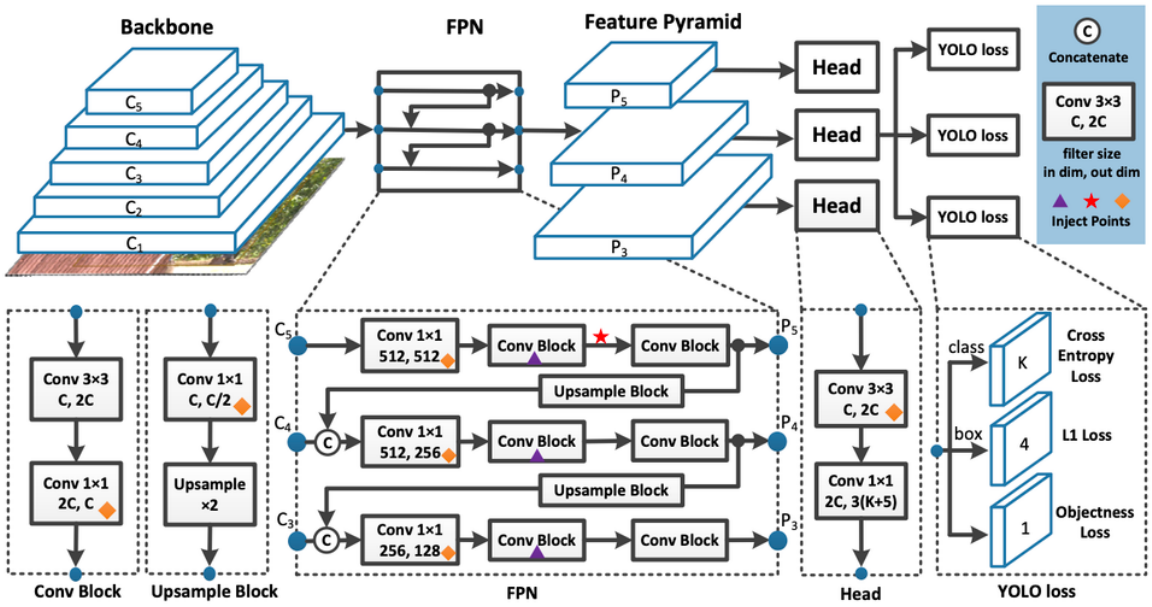


Figure 16, YOLOv7 network architecture

	Detector	IDF1	IDP	IDR	Precision	Recall
1	YOLOv3	0.8026	0.8375	0.7705	0.8628	0.7938
2	YOLOv4	0.8062	0.8529	0.7644	0.8809	0.8062
3	YOLOv5	0.8066	0.8555	0.763	0.8907	0.7944
4	ScaledYOLOv4	0.8102	0.8521	0.7723	0.8768	0.7947
5	YOLOR	0.8095	0.8569	0.767	0.8814	0.789
6	YOLOv7	**0.8115**	**0.8534**	**0.7735**	**0.8765**	**0.7945**

Table 5, Comparing between different YOLO versions [26]

As seen in Table 5, YOLOv7 got the best results out of the other YOLO algorithms, making it the best algorithm to apply with Vehicle Detection. As YOLOv7 outperforms the YOLOv3 algorithm, which outperformed the Faster R-CNN in accuracy and the performance time.

2.6 Traffic Flow Prediction

2.6.1 Papers using LSTM and GRU for Traffic Flow Prediction

In paper [27, 28], the authors compared Long Short-Term Memory, and Gated Recurrent Units. In paper [28], the authors compared LSTM and GRU on three different Optimizers, RMSprop, ADAM, and SGD.

LSTM and GRU are both types of Recurrent Neural Network, they are effective in handling long-term dependencies and capturing sequential patterns in data overcoming the limitations of normal RNNs, that struggle over long sequences. Although GRU has got fewer parameters and computational steps while also maintaining similar performance.

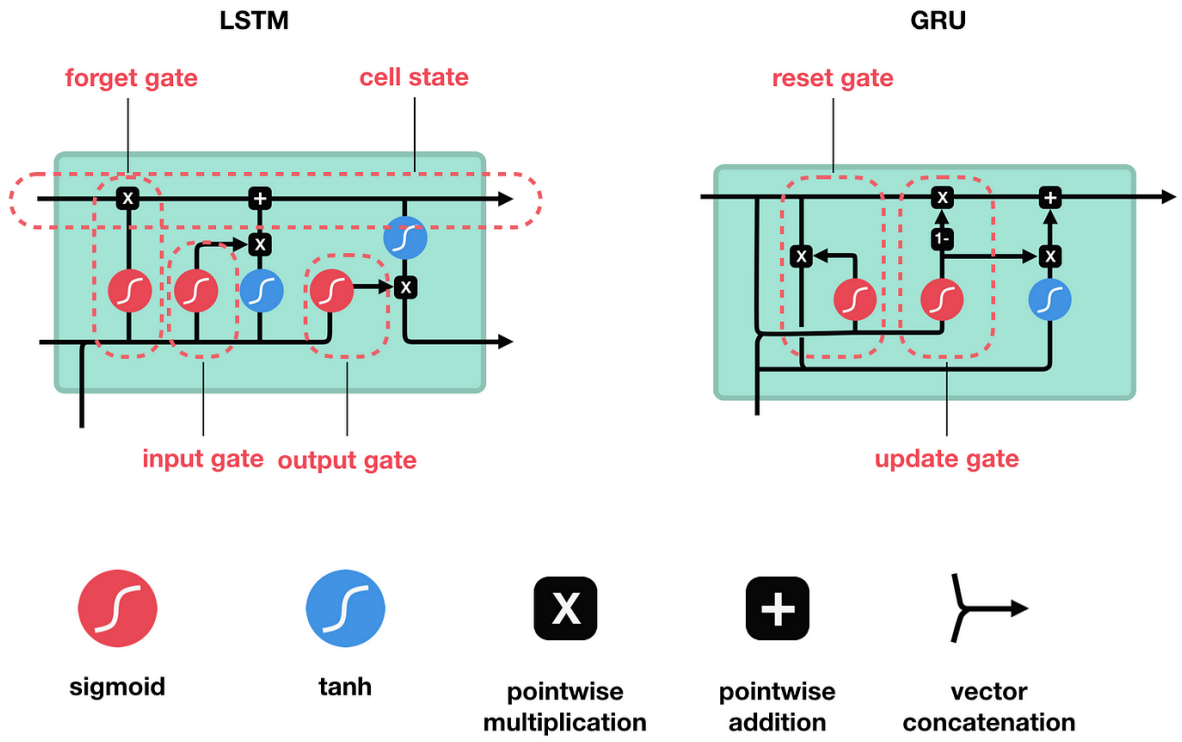


Figure 17, Comparison of LSTM and GRU

2.6.1.1 How LSTM works

LSTM consists of cell state, input gate, forget gate, hidden state, output gate, and candidate value.

Cell State is the memory component in the LSTM, it carries the information across time steps, allowing LSTM to maintain the relevant information and discard the irrelevant ones.

Input Gate is what determines how much information is added to the cell state, it passes through a sigmoid activation function. The output of the sigmoid function represents the amount of information to be let through.

Forget Gate is what dictates what information from the previous cell state is irrelevant and should be discarded. It also passes through a sigmoid activation function and the output represents the amount of information to forget.

Hidden State is the output of the LSTM cell. It is the Short-Term memory.

Output Gate is what determines how much information of the cell state should be revealed from the hidden state. It passes through a sigmoid activation function. The output of the sigmoid function represents the amount of information to be output.

Candidate Value represents the new information that can be stored in the cell state. It passes through a TANH activation function and generates a vector of new candidate values.

2.6.1.2 How GRU works

GRU consists of hidden state, update gate, reset gate, and candidate hidden state.

Hidden State represents the output of the GRU cell. It is the Short-Term memory.

Update Gate determines how much of the previous hidden state will be combined with the current input top compute the new hidden state. It passes on a sigmoid activation function, if the value is close to 0, the previous hidden is ignored, if the value is close to 1, the previous hidden state is mostly maintained to the next cell.

Reset Gate controls how much of the previous hidden state influences the calculation of the candidate hidden state. It passes through a sigmoid activation function. If the value is close to 0, the previously hidden has little influence, if the value is close to 1, the previous hidden state has a strong influence.

Candidate Hidden State represents the new information that could be added to the hidden state. It takes the information from the reset gate combined with TANH activation function and produces a vector of candidate values.

	LSTM	GRU
MSE	710.0502	668.9304
MAE	18.127758	17.2116

Table 6, Comparing LSTM and GRU performance [27]

As shown in Table 6, GRU outperforms the LSTM

Figure 18, Traffic Flow Prediction with RMSprop Optimizer [28]

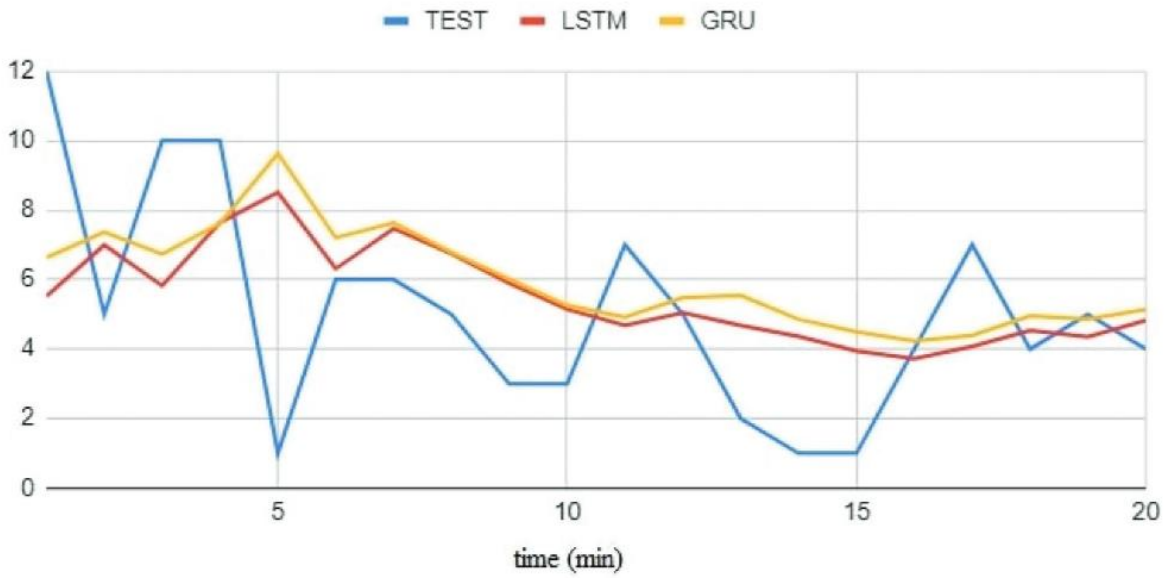


Figure 19, Traffic Flow Prediction with ADAM Optimizer [28]

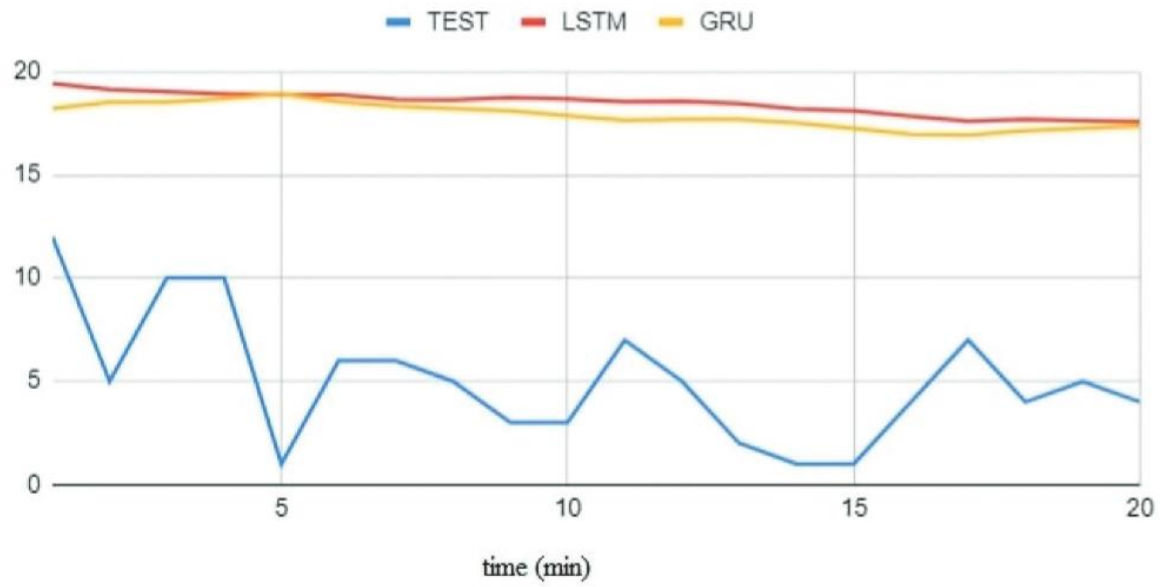


Figure 20, Traffic Flow Prediction with SGD Optimizer [28]

While shown at Figure 17 and 18, that GRU is close to LSTM that they are nearly mimicking and predicting the same traffic compared to the test data. In figure 19, using Stochastic Gradient Descent both models are nowhere near close to the actual test data, making SGD the worst of the three optimizers.

Index	Test	RMSprop	ADAM
1	12	LSTM =4.2 GRU = 6.2	LSTM = 5.5 GRU = 6.6
2	5	LSTM =6.1 GRU = 6.2	LSTM = 6.9 GRU = 7.3
3	10	LSTM =4.6 GRU =5.3	LSTM = 5.8 GRU = 6.7
4	10	LSTM =5.6 GRU = 5.6	LSTM = 7.6 GRU = 7.6
5	1	LSTM =6.2 GRU = 6.2	LSTM = 8.5 GRU = 9.6
6	6	LSTM =4.4 GRU = 5.1	LSTM = 6.3 GRU = 7.2
7	7	LSTM =5.1 GRU = 5.1	LSTM = 7.4 GRU = 7.6
8	5	LSTM =5.1 GRU = 5.1	LSTM = 6.7 GRU = 6.7
9	3	LSTM =4.3 GRU = 4.7	LSTM = 5.8 GRU = 6.0
10	3	LSTM =3.6 GRU = 4.7	LSTM = 5.1 GRU = 5.2
11	7	LSTM =3.3 GRU = 4.8	LSTM = 4.6 GRU = 4.9
12	5	LSTM =4.4 GRU = 5.3	LSTM = 5.1 GRU = 5.4
13	2	LSTM =3.5 GRU = 4.9	LSTM = 4.6 GRU = 5.5
14	1	LSTM = 3.1 GRU =4.5	LSTM = 4.3 GRU = 4.8

Table 7, Comparing RMSprop and ADAM optimizer [28]

As shown in Table 7 RMSprop is more accurate and closer to the test data than the ADAM optimizer. While LSTM is closer to the test data when the test data is low, GRU is closer to the test data when it is of great value, and considering the context that the model is better overestimating the traffic rather that underestimating it, the GRU is the better model between the two.

3. Intelligent Transportation System

3 Intelligent Transportation System

The proposed system architecture and methodology used for vehicle detection, tracking, data collection, and traffic flow prediction are presented in this chapter.

3.1 Solution Methodology

The proposed system is being implemented using the Software Development Life Cycle, which consists of Analysis, which was done in the background section, Design which is introduced in this section, Implementation, Testing and Evaluation. Video footage data is footage from traffic light cameras, the video is supposedly captured and sent over the internet as video stream. Then the vehicle detecting is done by machine vision models, such as R-CNN and YOLOv7. The tracking techniques are incorporating SORT which utilizes Kalman filtering and detection using YOLOv7. The data extracted from the model is the ID of the vehicle, its type, and its centroid coordinates in each frame. The traffic flow prediction uses the collected data and after pre-processing the data a traffic flow prediction model such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) was used for prediction. The training data was divided into 13 chunks of data to train the GRU model, MinMax scalar was used in the pre-processing. The traffic flow model predicts the traffic flow depending on the history of the traffic flow, and depending on the prediction the model either suggests using the route or not. The model decides the route usability according to a certain threshold, if it was exceeded the route won't be recommended.

3.2 Requirements

Requirements are crucial elements in any software development. They provide the foundation for designing, implementing, and evaluating the system.

3.2.1 Functional Requirements

The system shall

- Process video data and detect the type of vehicle.
- Track the vehicles.
- Collect traffic and road conditions information.
- Store the data coming from the video data.
- Predict traffic flow while minimizing the error.
- Recommend either taking this route or not based on the prediction.

3.2.2 Non-functional Requirements

- The System should be reliable even in the most intense scenarios, the system should have a strong and critical infrastructure.
- System security should be maximized against cyberattacks and leaks as the data going through the system is extremely confidential.
- The system should be highly scalable and adaptable as the system could be rolled out nationwide and should be able to adapt to such changes.
- The system should be efficient and should be able to handle many operations at once in the case of a crowded street.

3.3 Design.

The design section of the project report showcases the system's architecture and flow. It explains how the components are interconnected and the key design principles applied.

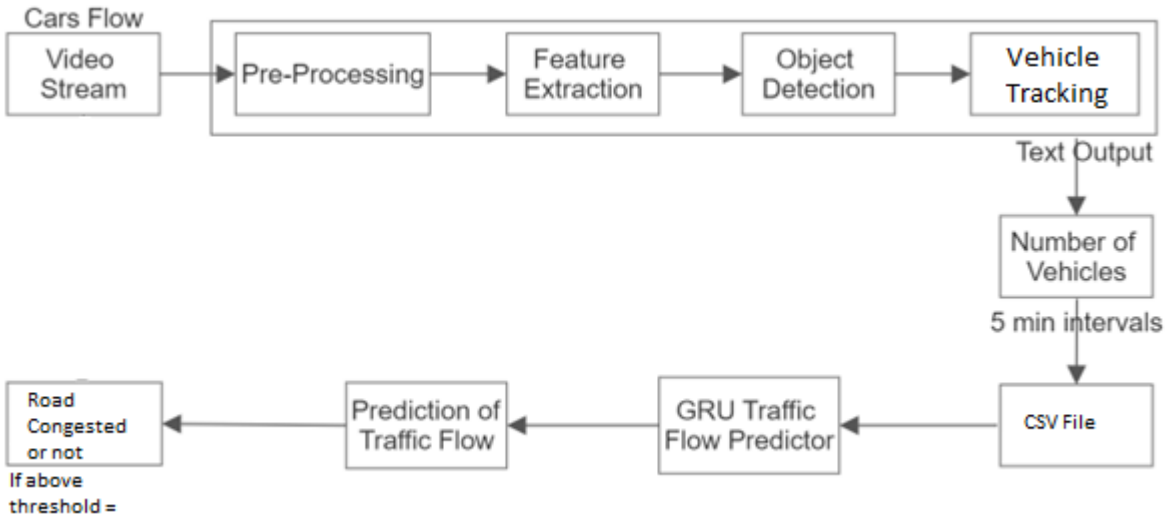


Figure 21, Block Diagram for how the System is designed

How the video stream is being input to the system is using traffic cameras which stream the video to the system, but since it is not possible to implement this hardware side, it is not in the scope of the proposed system.

4. Intelligent Transportation System Implementation

4 Implementation

This chapter details the implementation of the proposed system. It provides insights into the technologies and software frameworks used. The dataset collection process, the implementation and integration of vehicle detection and tracking algorithms are explained. Any challenges encountered during implementation and the strategies employed to overcome them are also discussed.

4.1 Implementing Machine Vision Model

After determining trying CNN, R-CNN, and YOLOv7. YOLO was the best model to implement since it is the fastest model and most accurate as shown in Tables 4, 5. It was decided to begin implementing the model by testing and getting the required dependencies and libraries such as, matplotlib, NumPy, OpenCV, Pillow, PyYaml, SciPy, Pytorch, TorchVision, and Cuda to be able to use the GPU while training and detecting using the model. Some libraries such as Cuda took more time to get it right with no errors, where Cuda insisted to download the wrong model which only accessed the CPU rather than the GPU, but after some days of troubleshooting it was finally working as intended.

4.1.1 Training the model using Transfer Learning

4.1.1.1 Transfer Learning

Transfer Learning is to take advantage of an existing model that is trained on large datasets such as ImageNet or COCO, called pre-trained model. The pre-trained model is used in feature extraction. The model usually is divided into layers, the early layers are the low-level features that are usually broad, and it is not recommended to overwrite those layers, so they are frozen, and extracted. The higher layers that are more specific to certain models to fine-tune them by training them on our custom data.

4.1.1.2 Collecting the training datasets

At first, the search was for a dataset that has got all the classes while also being balanced and relates to the scope of the project. But such specific requirements were hard to come by, so the decision was to choose multiple datasets and combine them all to achieve such requirements. 4 Datasets were collected at first, to combine these datasets with each other's, labels had to be integrated with each other's. So, a python script that accesses the dataset labels and changes the class number to be compatible with the other datasets was implemented.

After choosing the right datasets [29-30] to train and test on, pre-processing and resizing the data to 640x640. The training begun using a machine with AMD Ryzen 7 3750H CPU, 16GB DDR4 RAM, and a 6GB GDDR5 memory GTX 1660 TI GPU running Windows 11. Training was done 1600 images with the following parameters, Batch Size = 8, Epochs = 55, and Classes = 8. And after 25 hours

the training was done, but the yielded results were not sufficient for a stable model as the training dataset was not enough for the model to train on all the cases.

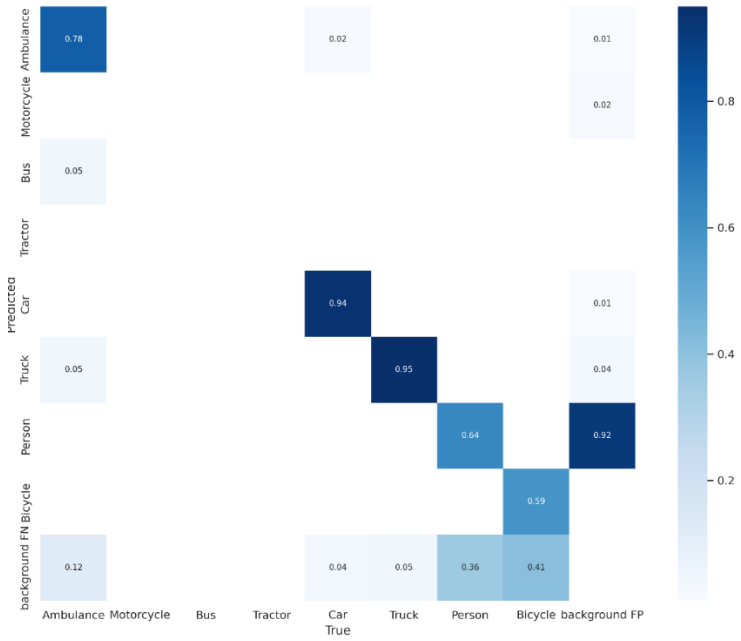


Figure 22,First YOLOv7 weights Confusion Matrix

As shown in Figure 22, some classes such as Bus and Motorcycle are nonexistent, and on the training data it always detected an ambulance is there, while there were no ambulances.

So, it was decided to add more images to the dataset [31], after adding 2010 images to the training dataset of total 3610 now and reconcile the new dataset with the already existing datasets. It was decided to use Google Colab instead for faster training time and bigger batch size. The GPU used in this training session was a 16GB GDDR6 Tesla T4. Now training is being done on 3610 images with the following parameters, Batch Size = 16, Epochs = 26, and Classes = 8. The batch size is doubled this time as the GPU Memory allows us to.

One of the limitations that was faced was that the epochs had to be nearly halved because Colab has got limited Computational Units that the runtime disconnects before exceeding the units, which is has happened when training before with 50 epochs and it got disconnected after 30 epochs. So, such things were taken into consideration when training for the second time.

Validation dataset contains 430 images but none of them are from the class tractor that is why it is empty on the confusion matrix in Figure 23.

Figure 23, YOLOv7 weights Confusion Matrix

As shown in Figure 23, compared to Figure 22 the model is way better and detects all the classes.

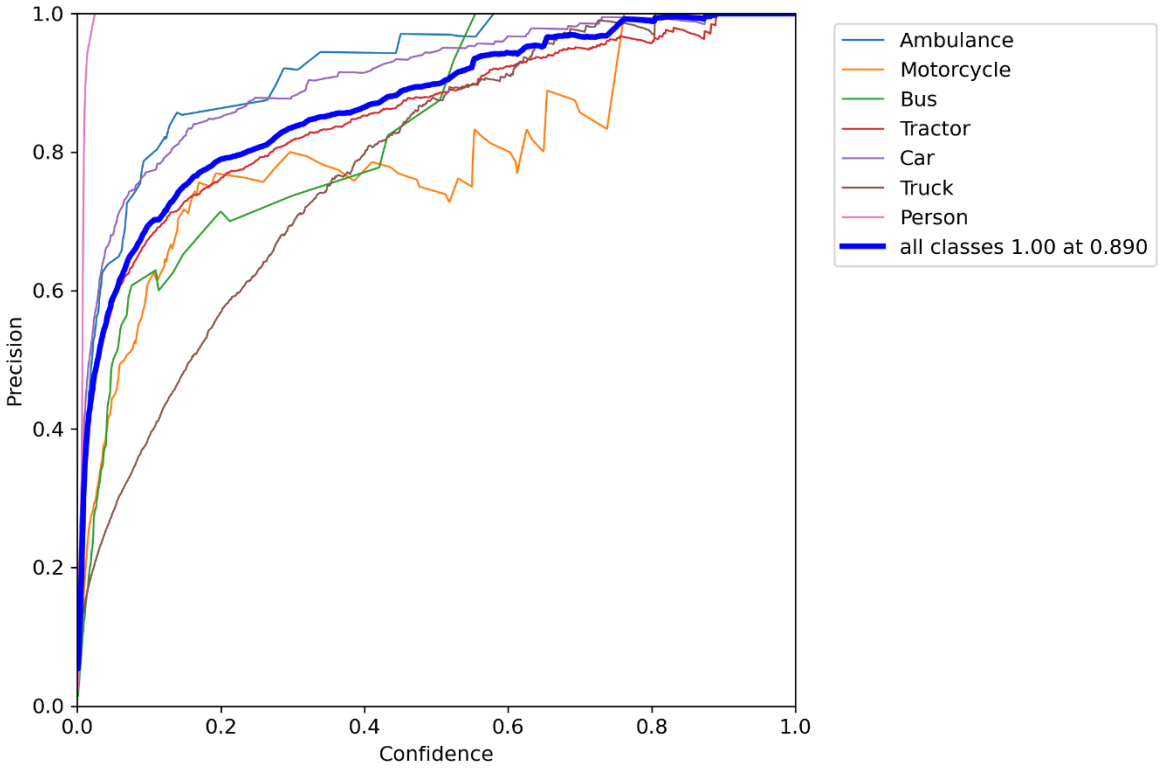


Figure 24, YOLOv7 weights Precision Curve

The model accuracy overall is better than the first training session, with a validation accuracy of 0.847.

The model contains 79 convolutional layers, 15 concat layers, 5 max pooling layers, 2 up-sampling layers, 3 RepConv, and 1 output layer.

Concat layer is what divides the input into a grid allowing the model to predict bounding boxes and class probabilities for each grid cell.

RepConv layer applies a convolutional layer with different kernel size so that it can get information at various scales, rather than like the traditional convolutional layers where a fixed kernel size is used, which limits the ability to get the overall context of the scene.

Non-maximum Suppression was implemented to prevent overlapping of the bounding boxes predictions. NMS takes several steps to get the highest relevant bounding box such as, confidence, IoU or Intersection over Union which calculates the degree of overlap with representing the ratio between the bounding boxes to their union area.

4.1.2 Object Detection Implementation

To detect vehicles in images and videos, detection was implemented with the following code.

This code calls the model to make prediction, then applies Non-Max Suppression. Then calls the classifier function.

```
with torch.no_grad():
        pred = model(img, augment=opt.augment)[0]
    t2 = time_synchronized()


    # Apply NMS
    pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes,
agnostic=opt.agnostic_nms)
    t3 = time_synchronized()


    # Apply Classifier
    if classify:
        pred = apply_classifier(pred, modelc, img, im0s)
```

The model which is called in the first loop is what calls the trained weights from earlier.

The following code processes the detection and extracts the information per image or frame.

```
# Process detections
    for i, det in enumerate(pred):  # detections per image
        if webcam:  # batch_size >= 1
            p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
        else:
            p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)


        p = Path(p)  # to path
        save_path = str(save_dir / p.name)  # img.jpg
        txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  #
img.txt
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain
```

4.1.3 Counting Vehicles in Images and Videos

After completing the implementation and successfully training the model, the image after detection was only showing the bounding boxes as shown in Figure 25, so a count function was implemented using OpenCV where the detected objects' classes are written over the image as shown in Figure 36.

Figure 25, Image output after detection without count

Figure 26, Image output after detection with adding the count function

This helps with getting a clearer view of the output.

The following code defines the found classes in the image and increments the classes and displays the text on the image using OpenCV.

```
def count(founded_classes,im0):
    font=cv2.FONT_HERSHEY_SIMPLEX
    model_values=[]
    aligns=im0.shape
    align_bottom=aligns[0]
    align_left=(aligns[1]/25)

    for i, (k, v) in enumerate(founded_classes.items()):
        a=f"{k} = {v}"
        model_values.append(v)
        align_bottom=align_bottom-50
        cv2.putText(im0, str(a), (int(align_left),align_bottom), font, 1, (153,0,153), 3, cv2.LINE_AA)
```

4.1.4 Tracking in Videos

To count the number of vehicles passing through the street, tracking each vehicle that the model detects was the answer. To track the vehicle and make sure that it does not get counted twice, the model should track it and give it an ID. SORT or Simple Online and Realtime Tracking [34] was used for that matter. SORT uses the object detection model, then applies motion estimation for objects to estimate, track, and predict where the object is going. It takes into consideration the aspect ratio and the velocity of the object. SORT uses Kalman filter in this association. Kalman filter assumes that objects can be modeled using linear equation with Gaussian noise. The filter has two steps, Prediction step, and Update step. Prediction step uses a mathematical model and the previously estimated state to predict the current state of the object. The update step compares the predicted state of the object with the measurements and adjusts the estimated state accordingly.

The bounding box is drawn, and SORT was initialized which then converts the box into aspect ratio and finds the center. Then passes it to the Kalman filter.

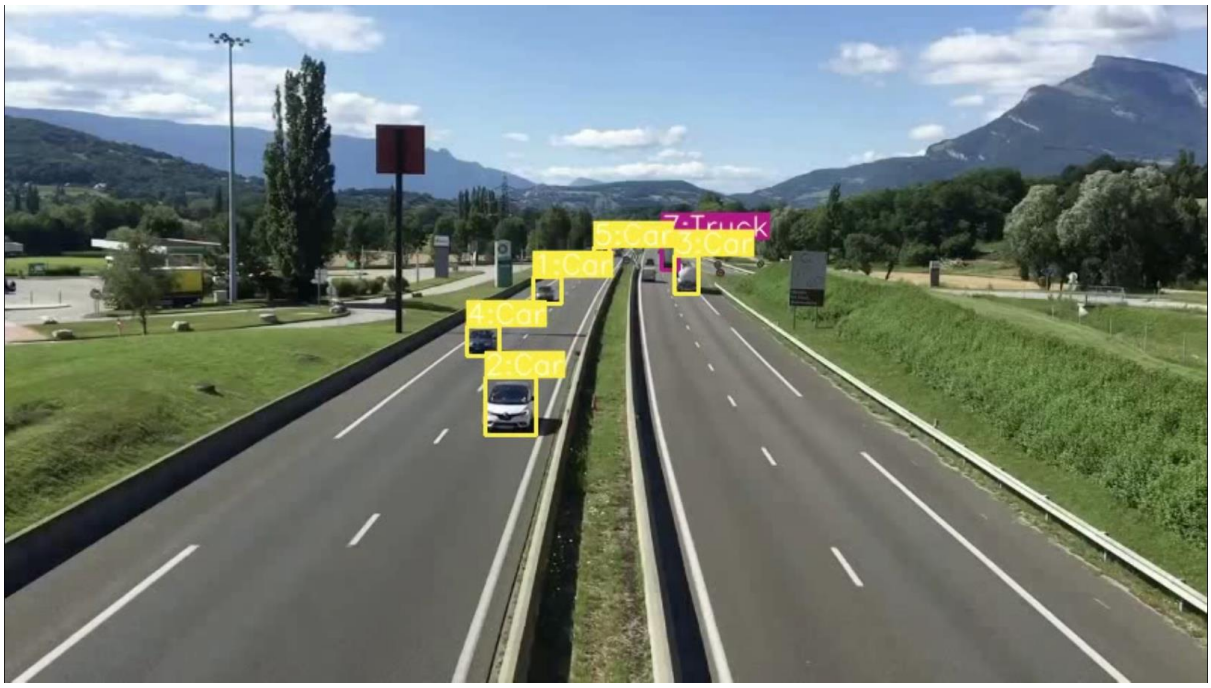


Figure 27, Example of Tracking with YOLOv7 and SORT

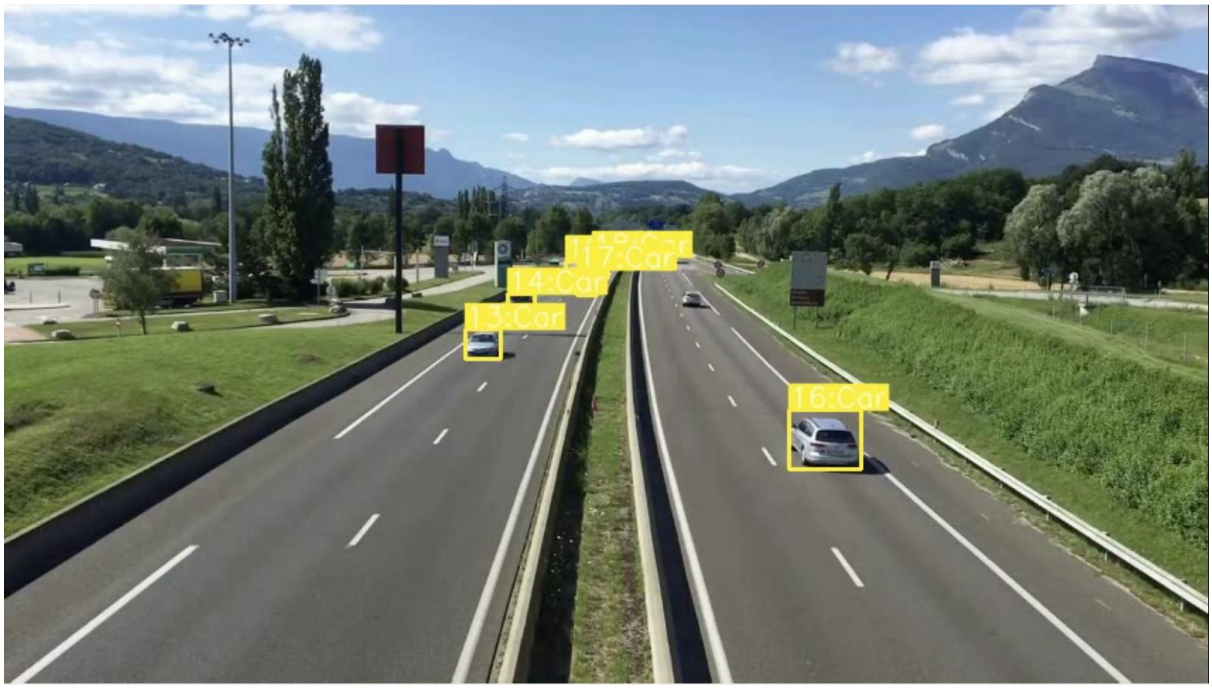


Figure 28, Another Example of Tracking with YOLOv7 and SORT

4.2 Saving the data into CSV file

The model outputs data that is going to be used in the Traffic Flow Predictor, but to have the ability to use it must be prepared. The model outputs a folder called labels, that folder contains .txt files, each .txt file equates to a frame.

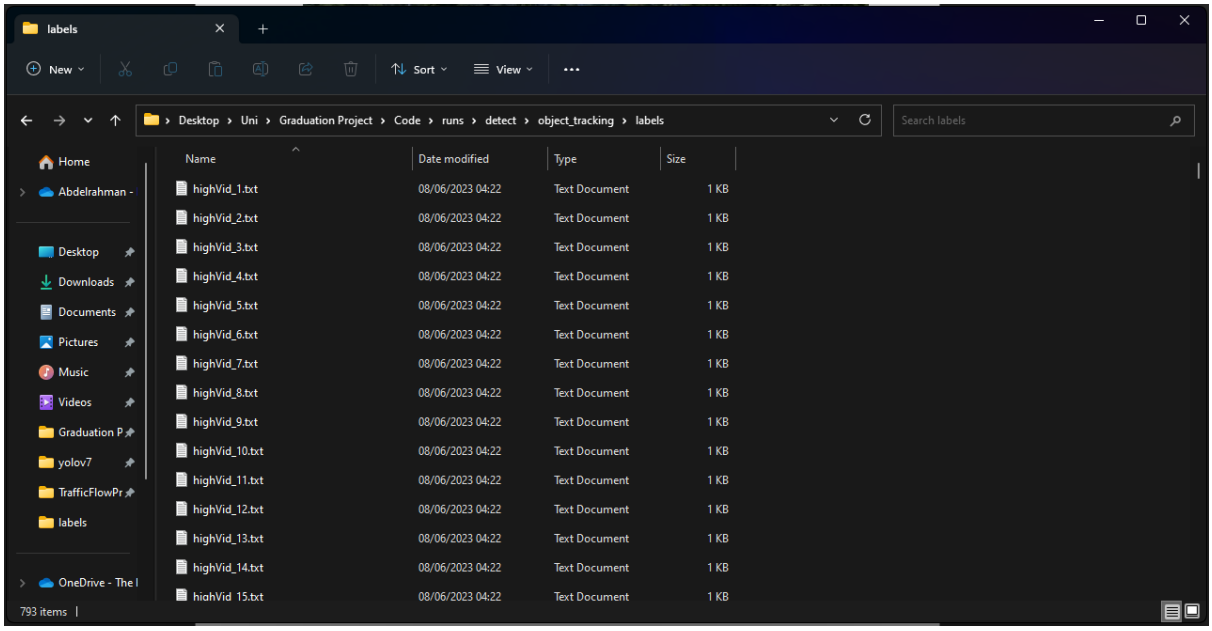


Figure 29, Labels folder where each txt file = frame

The .txt files are structured as follows, {vehicle ID, class, annotations for the centroid (x, y), annotations for the bounding box (x1, y1, x2, y2)}

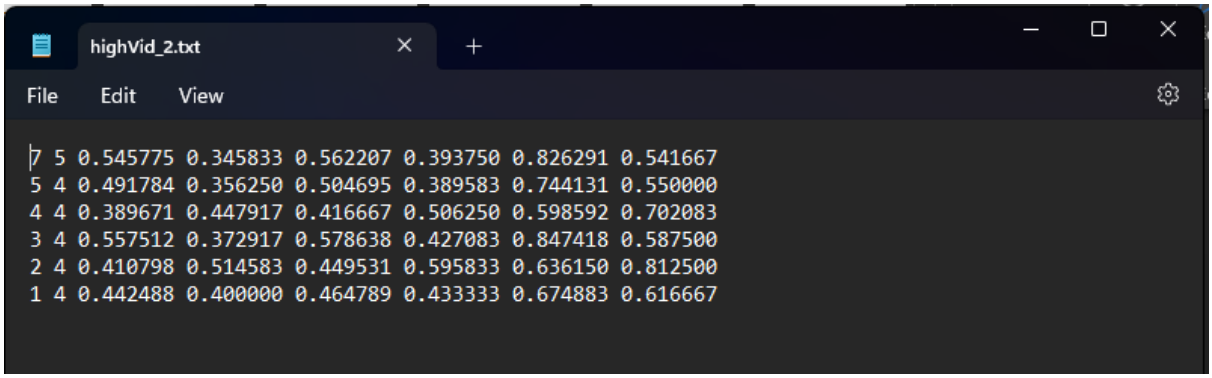


Figure 30, How the labels are structured out of the tracking model

4.2.1 Dividing the labels into batches

A limitation in dealing with the data was discovered, and that was that the data was not actually divided since all the video streams found was less than 5 minutes, so a function that divides the labels to batches was implemented for the traffic prediction model. This function takes the labels and divides them into 13 batches, each batch inserted to 13 different folders. The data is divided into 13 batches exactly because this is the minimum number of rows that the traffic prediction model takes as an input to make a prediction.

```
def files_into_batches(input_folder, output_folder):
    # Create the output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    files = os.listdir(input_folder)
    num_files = len(files)
    batch_size = (num_files + 13 - 1) // 13

    for i in range(0, num_files, batch_size):
        batch_files = files[i:i + batch_size]
        batch_folder = os.path.join(output_folder, f'batch_{i // batch_size + 1}')
        os.makedirs(batch_folder, exist_ok=True)
        for file in batch_files:
            file_path = os.path.join(input_folder, file)
            shutil.copy(file_path, batch_folder)
```

4.2.2 Collecting all the frames label files onto one

Since this information is currently useless, a function that loops on the labels folders was implemented, and it stores only the first two integers (vehicle ID, class) and stores these integers in 13 different .txt files for every folder.

A dictionary for duplicate IDs was created so that it only has the count of the vehicles that were tracked using the model. And since the class number is useless, another dictionary that contains the class number and its corresponding name was implemented.

```
def process_txt(folder_path, output_file):
    # Dictionary to store unique first integers and their corresponding replaced second integers
    unique_integers = {}

    # Mapping for replacing second integers
    second_int_mapping = {
        '0': 'Ambulance',
        '1': 'Motorcycle',
        '2': 'Bus',
        '3': 'Tractor',
        '4': 'Car',
        '5': 'Truck',
        '6': 'Person',
        '7': 'Bicycle'
    }
```

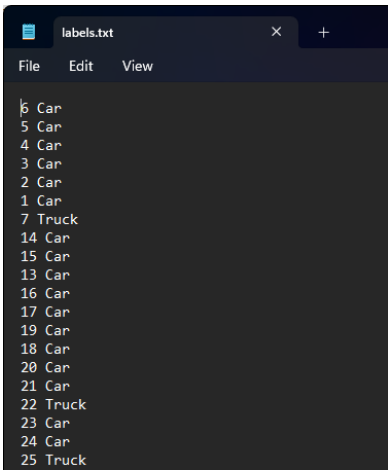


Figure 31, Collecting all the labels to one output

4.2.3 Counting and converting the .txt file to a CSV file

Since CSV are easier to deal with, a function that counts the IDs in the .txt files and insert it to a CSV file each file is inserted to a new row with the current time was implemented. That way the CSV file can be passed to the Traffic Flow Predictor.

	Time	Lane 1 Flow (Veh/5 Minutes)	# Lane Points	% Observed
0	2023-06-09 17:44:34	14	1	100
1	2023-06-09 17:44:34	17	1	100
2	2023-06-09 17:44:34	4	1	100
3	2023-06-09 17:44:34	6	1	100
4	2023-06-09 17:44:34	1	1	100
5	2023-06-09 17:44:34	2	1	100
6	2023-06-09 17:44:34	4	1	100
7	2023-06-09 17:44:34	4	1	100
8	2023-06-09 17:44:34	3	1	100
9	2023-06-09 17:44:34	3	1	100
10	2023-06-09 17:44:34	6	1	100
11	2023-06-09 17:44:34	7	1	100
12	2023-06-09 17:44:34	3	1	100

Figure 32, Sample of how the function should work

4.2.4 Converting the ID and type to CSV for future work

This way, the data output from tracking model can be used in our Traffic Flow Predictor model, but since data is so valuable, another function that converts the .txt files to a CSV file was implemented. This function takes the ID and type of the corresponding vehicle for future work that can incorporate some Data Analysis in it.

	A	B
1	Vehicle ID	Vehicle Type
2	6	Car
3	5	Car
4	4	Car
5	3	Car
6	2	Car
7	1	Car
8	7	Truck
9	14	Car
10	15	Car
11	13	Car
12	16	Car
13	17	Car
14	19	Car
15	18	Car
16	20	Car
17	21	Car
18	22	Truck
19	23	Car
20	24	Car

Figure 33, Sample of the data extracted for future work

4.3 Implementing Traffic Flow Predictor

After reviewing that GRU is a better model than LSTM, implementation to the model has begun. The model had 1 input layer, 6 GRU Layers, 6 dropout layers, 1 dense layer or a fully connected layer.

The dropout layer is used to prevent overfitting, the layer randomly sets a percentage of the Gated Units, that can be controlled through the hyperparameters, to zero. And the remaining units are scaled to compensate for the other units. This way the model is more trained against noise and is generalized for real-world data.

First, the data must be pre-processed, so a MinMax Scaler was applied to normalize the data between zero and 1. Then the data was converted into a 2-dimenstional array and applied transformation to the data using the MinMax Scaler. Then the training data was divided into 13 element snippets for the model to train on predicting the next value from these 13 values, which allows the model to catch on whether the traffic is getting higher, lower, or staying the same.

The first model was created using the Sigmoid activation function and the RMSprop optimizer since that was the best performing optimizer from the papers.

Sigmoid activation function ranges between 0 and 1, which makes it better for the binary classification but not so much in regression applications. While the tanh ranges between -1 and 1 making it more suitable and flexible than the Sigmoid.

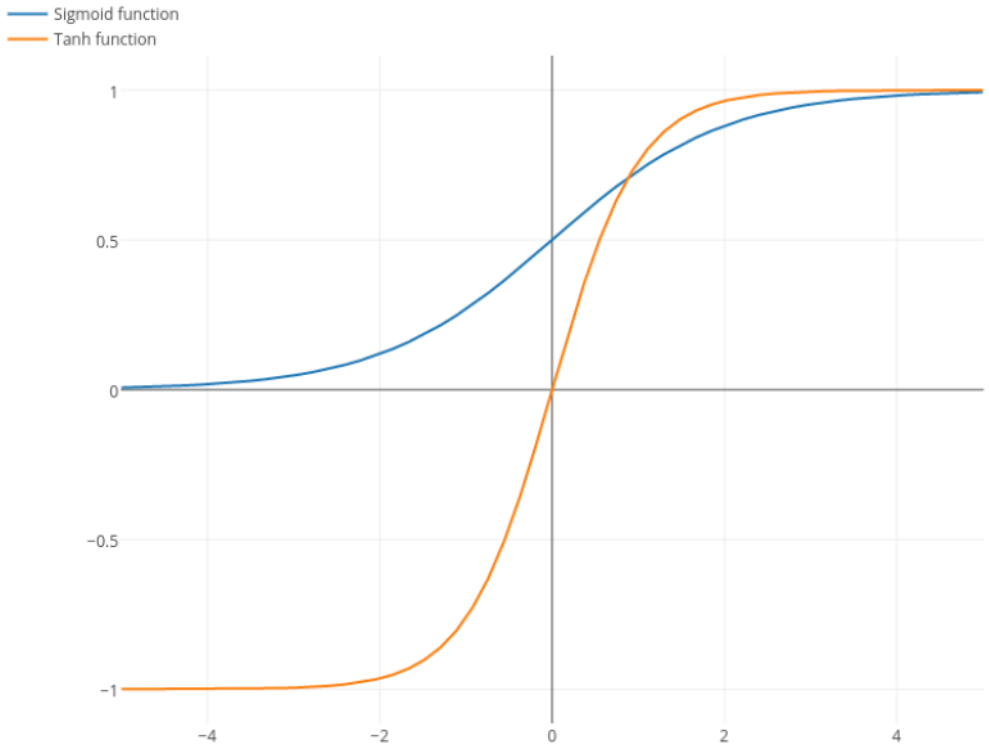


Figure 34, Difference between Sigmoid and TanH activation functions

RMSprop is an extension of gradient descent optimization which patches up some of the limitations. RMSprop adapts the learning rate and adjusts it for each parameter while training.

RMSprop main idea is that it normalizes the gradient using the Root Mean Square or RMS. So, it initializes the parameters such as learning rate and decay rate and adjusts the parameters by dividing the gradient by the RMS average of the squared gradients.

Adapting the learning rate allows RMSprop to handle different scales of parameters and overcome problems such as vanishing and exploding gradients.

ADAM is similar to RMSprop in the way that it adapts for different parameters. Except it maintains two averages, the mean, and the variance.

Similar to RMSprop, ADAM begins training with initialized values such as learning rate and the averages. And updates the mean and variance after the training iteration.

Stochastic Gradient Descent is also an extension of gradient descent, except that SGD trains on small batches and updates after, while gradient descent computes the loss function over the entire dataset. Making SGD more efficient computationally and time wise.

SGD initializes the model parameters, shuffles the data for randomization, and iterates over small batches of the training data.

4.3.1 GRU with Sigmoid activation and RMSprop optimizer

Training was done using a machine with AMD Ryzen 7 3750H CPU, 16GB DDR4 RAM, and a 6GB GDDR5 memory GTX 1660 TI GPU running Windows 11. Training was done 7776 records of vehicles that passed a highway under the California Department of Transportation with interval of 5 minutes, with the following parameters, Batch Size = 150, Epochs = 50, and early stopping to minimize overfitting. The first model did not yield good results at all, so it was decided to use tanh from now on.

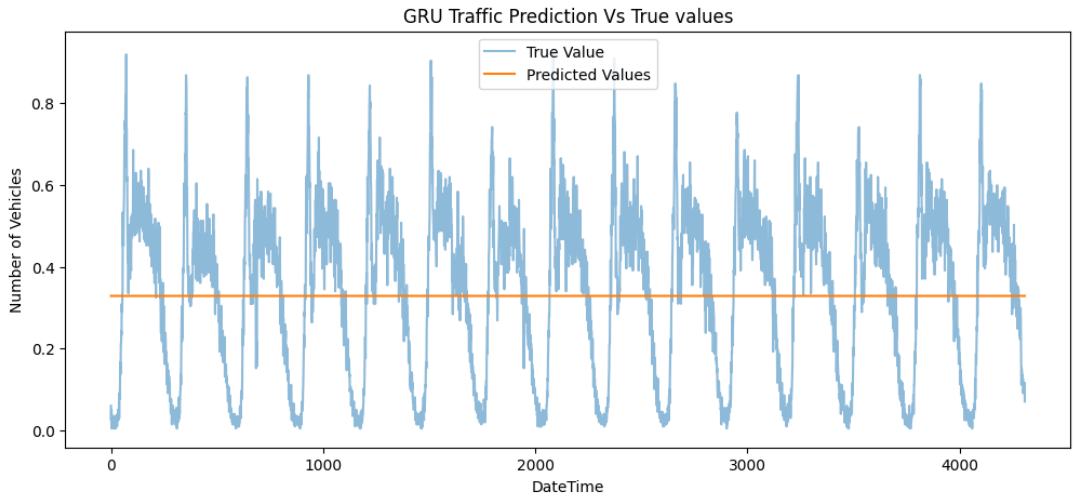


Figure 35, Plot comparing between true data and predicted data using sigmoid and RMSprop

4.3.1 GRU with TanH activation and RMSprop optimizer

The decision to use tanh quickly reflected on the model as soon as it was implemented a tanh model with RMSprop optimizer, and the results were surprising.

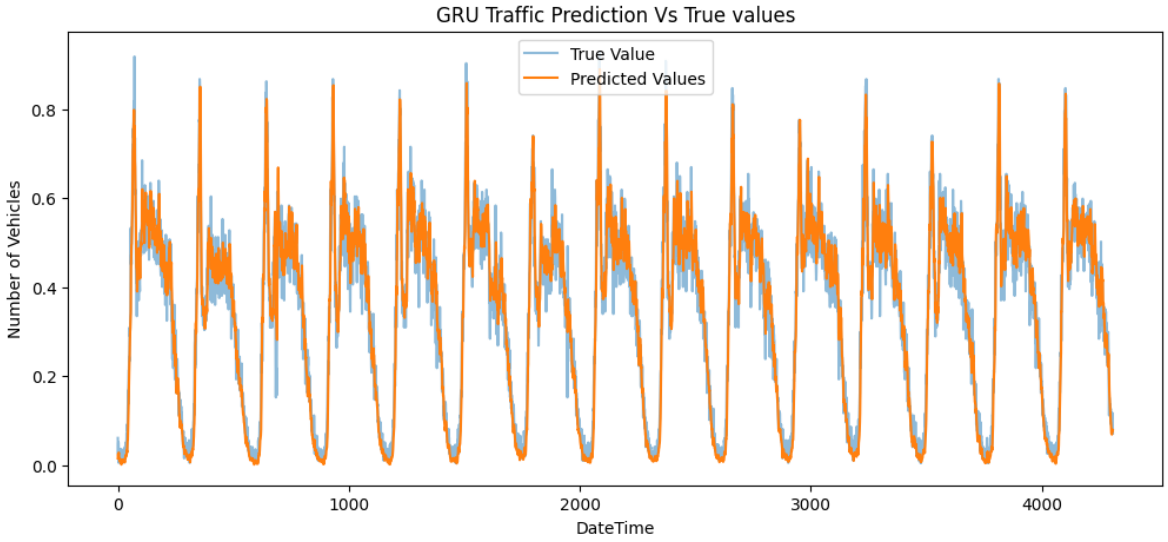


Figure 36, Plot comparing between true data and predicted data using tanh and RMSprop

4.3.1 GRU with TanH activation and ADAM optimizer

After that it was decided to experiment with the optimizer to find out who will yield better results, so two other models were implemented using tanh activation function, one was implemented with ADAM optimizer and the other was implemented with Stochastic Gradient Descent.

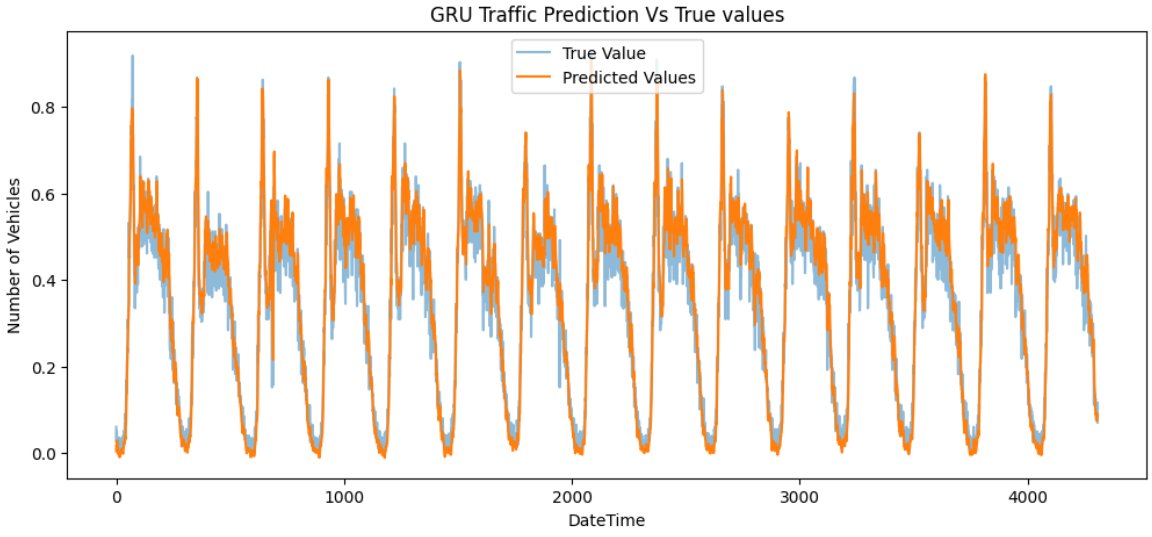


Figure 37, Plot comparing between true data and predicted data using tanh and ADAM

4.3.1 GRU with TanH activation and SGD optimizer

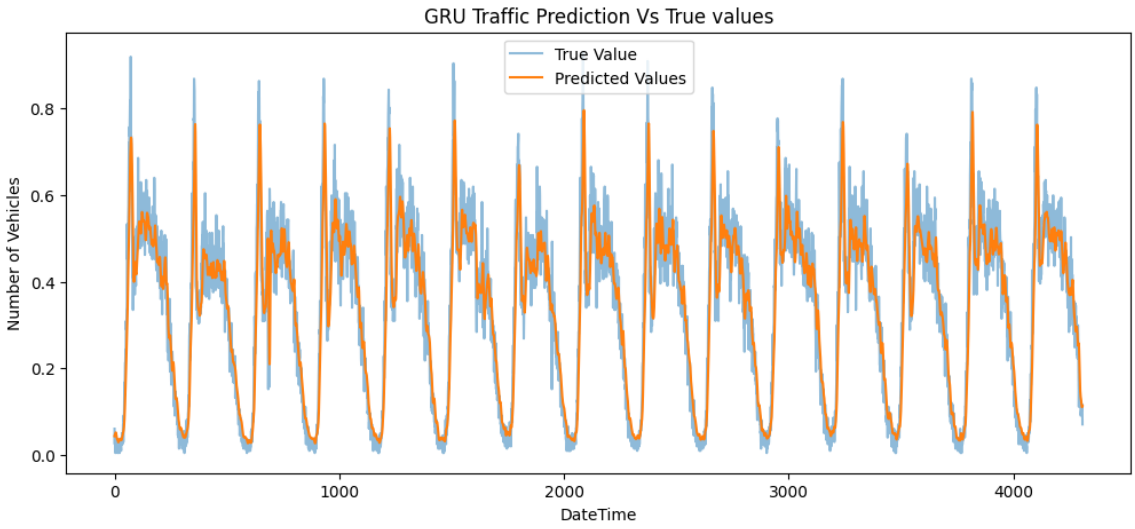


Figure 38, Plot comparing between true data and predicted data using tanh and SGD

	ADAM	SGD	RMSprop	RMSprop + Sigmoid
MSE	0.00297	0.00487	0.00268	0.0421
RMSE	0.0545	0.0698	0.0518	0.205
MAE	0.0411	0.0505	0.0389	0.177

Table 8, Comparison between the different GRU models

After the model predicts the traffic flow for the next period, the mean of the prediction is calculated and if the mean is higher than the threshold, the model recommends taking another road.

5. Testing and Evaluation

5 Testing and evaluation

In this chapter, the methodology used to evaluate the performance of the system is explained. It presents the results of vehicle detection accuracy, tracking efficiency, and the accuracy of traffic flow predictions. Additionally, any limitations or shortcomings observed during the testing process are discussed.

5.1 Testing

As shown in Figure 39 and 40, all the vehicles are correctly classified.

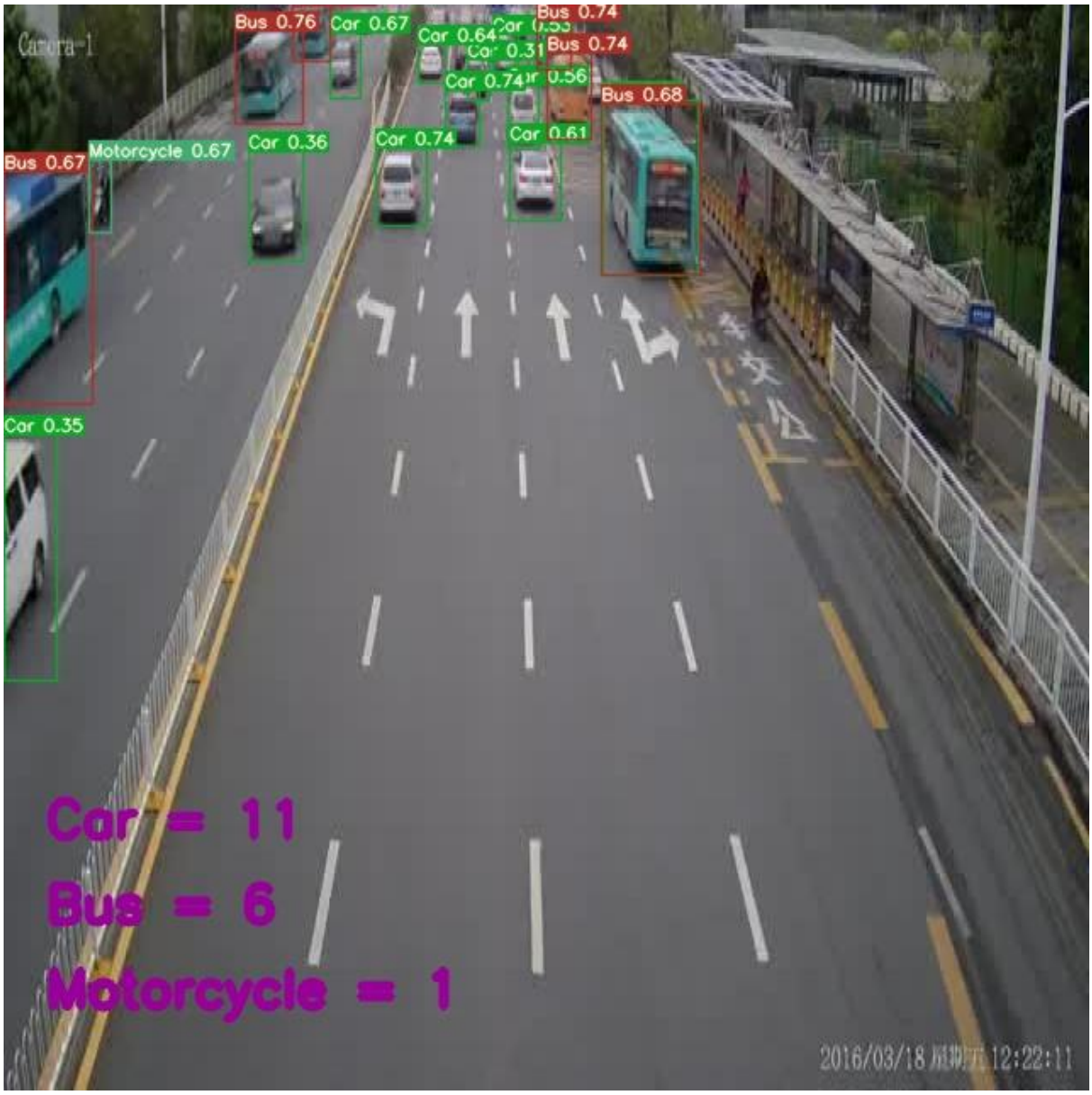


Figure 39, Example of the model detecting every vehicle right

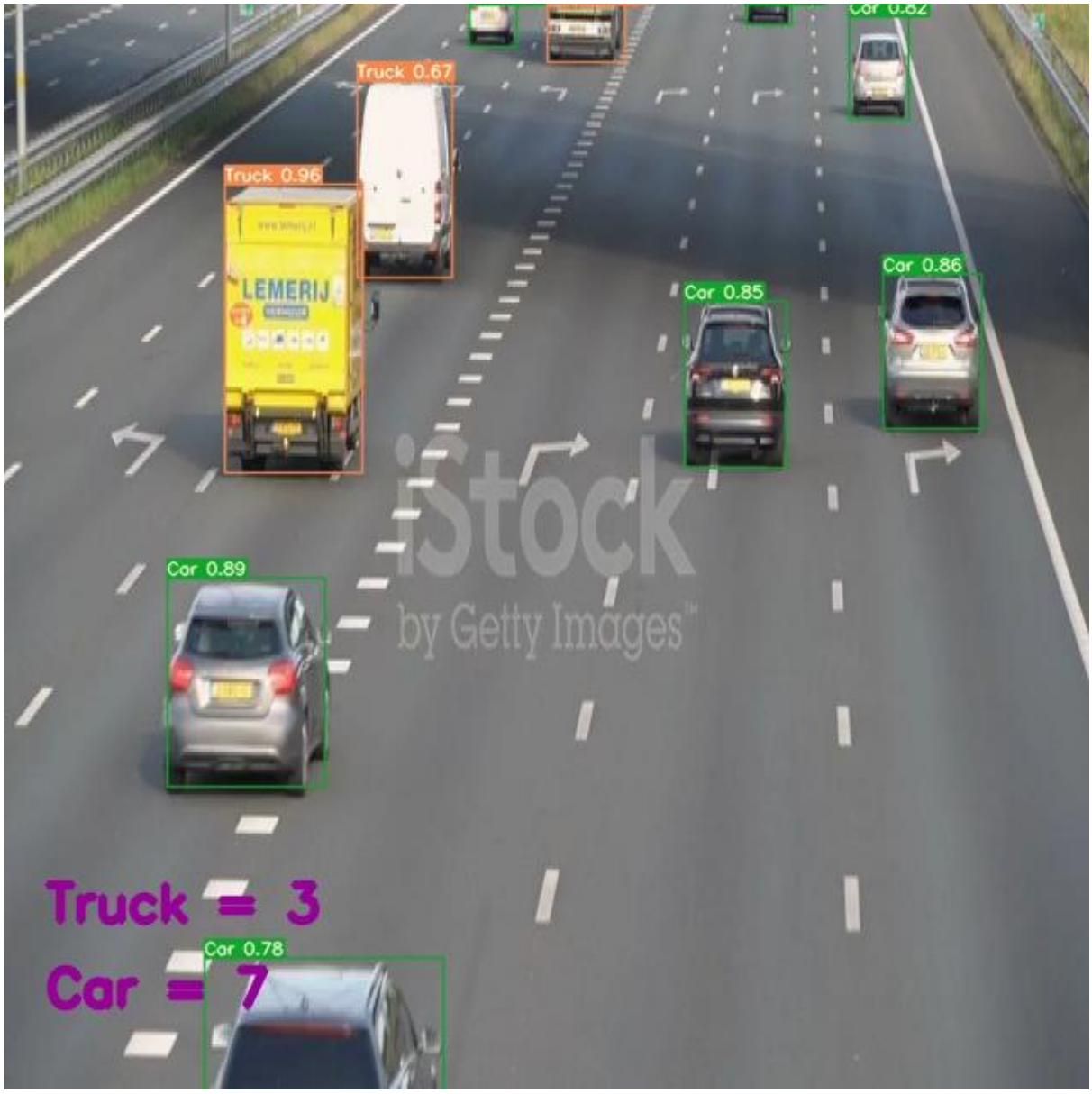


Figure 40, Another example of the model correctly detecting every vehicle

There are some cases where the model does not detect accurately, such as the following images,

Figure 41, Image where one truck gets detected, but the car does not

In Figure 41, the image is so pixelated that the model cannot identify the car in the back.

Figure 42, Truck not detected due to image pixilation

The same case as Figure 41, Figure 42 where the image is not very clear that it does not detect the truck.

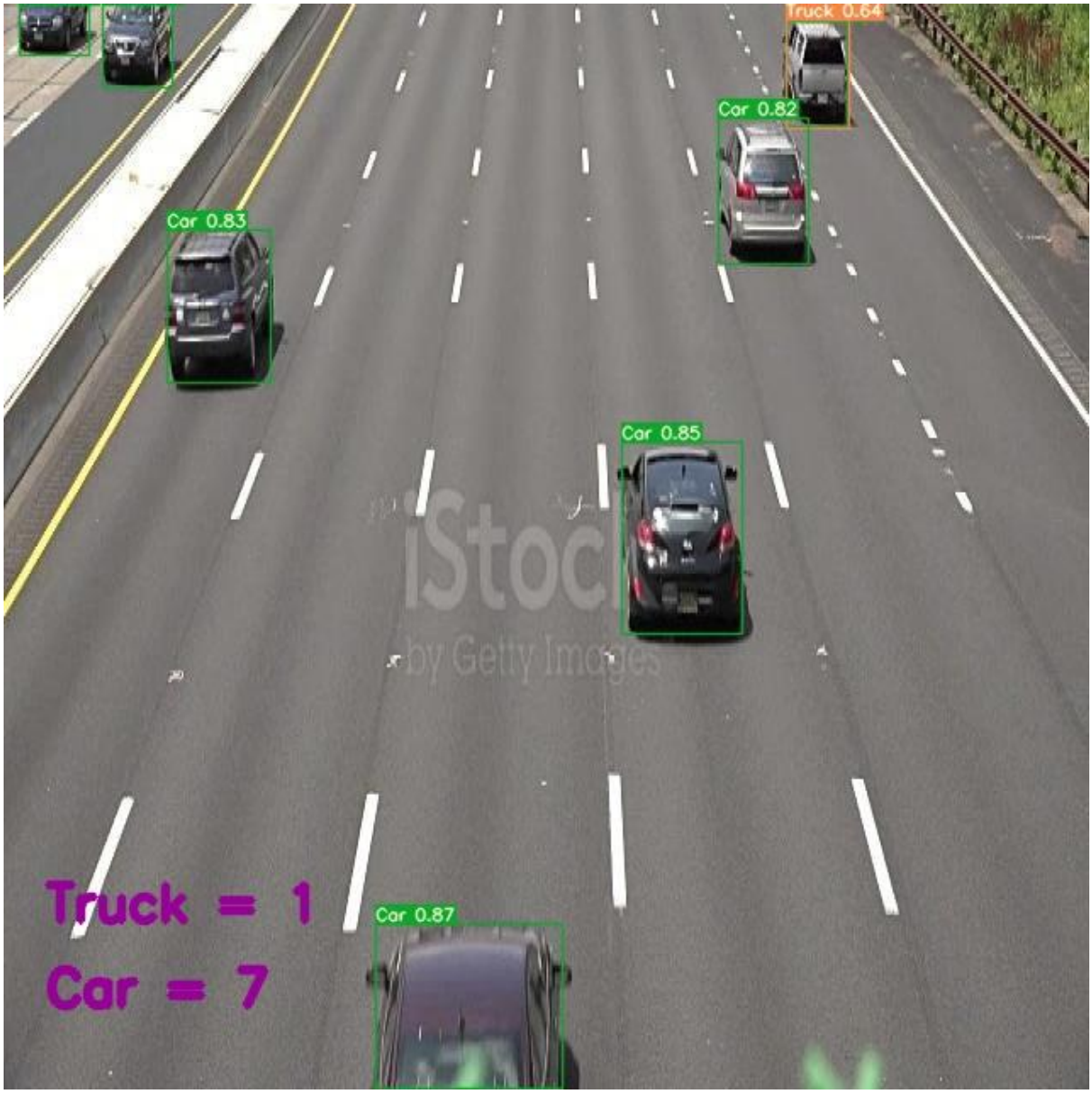


Figure 43, Image where all the cars are detected correctly except for a white SUV

Something that was confusing the model in detection was White SUVs where the model confuses the car for a truck because they are a bit bigger than normal cars.

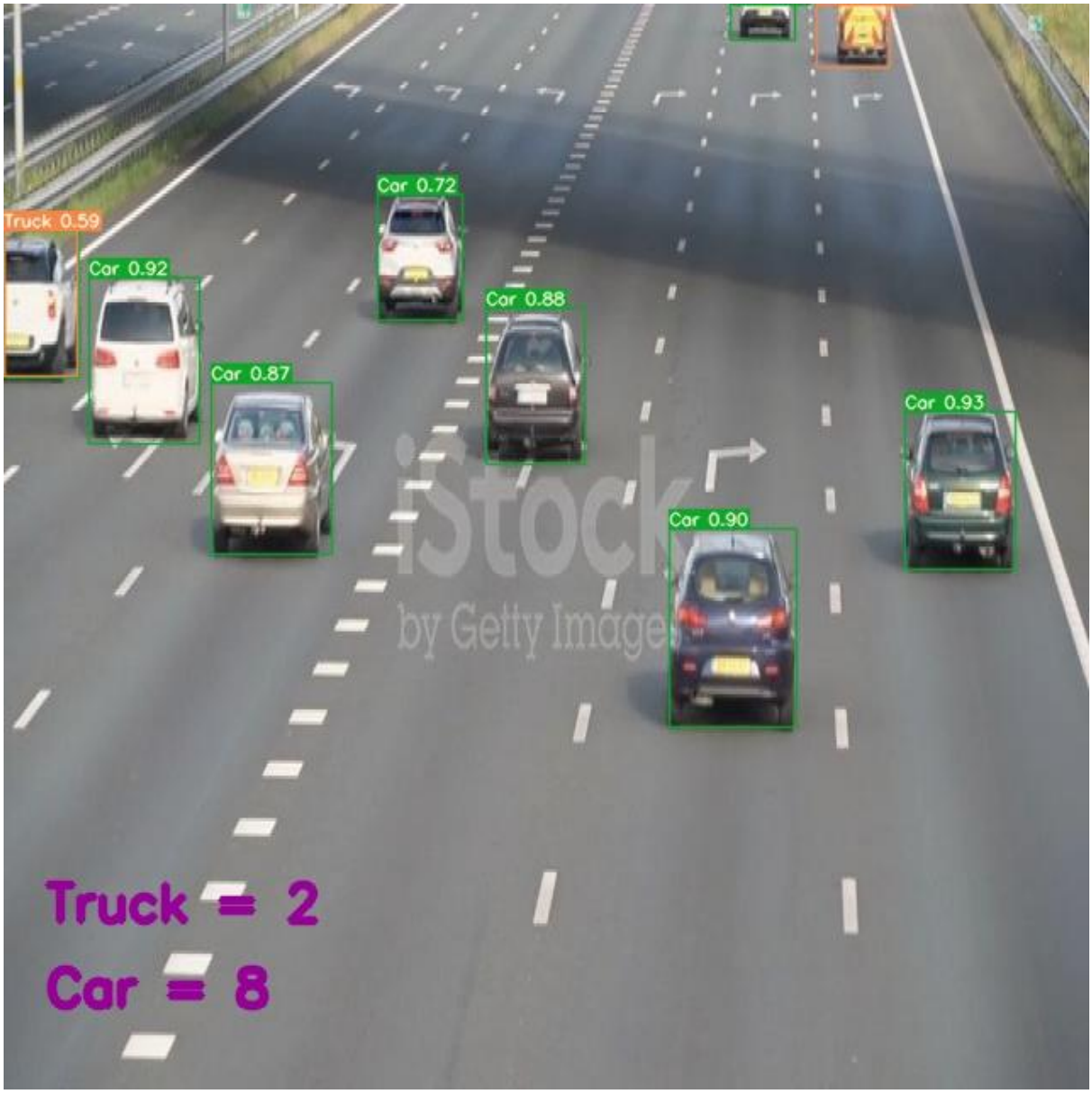


Figure 44, Another example for a Car (SUV) being detected incorrectly

Same as Figure 43, the white car (SUV) is being incorrectly detected for a truck.

The system shall process video data and detect the type of vehicle.

The system can successfully process video data and detect the type of vehicle, count them, and display the count on the screen. As shown in figure 45.

Figure 45, Example of the system detection

The system shall track the vehicles.

The system can successfully track the vehicles by assigning them IDs with the type of vehicle next to it. As shown in figure 46.

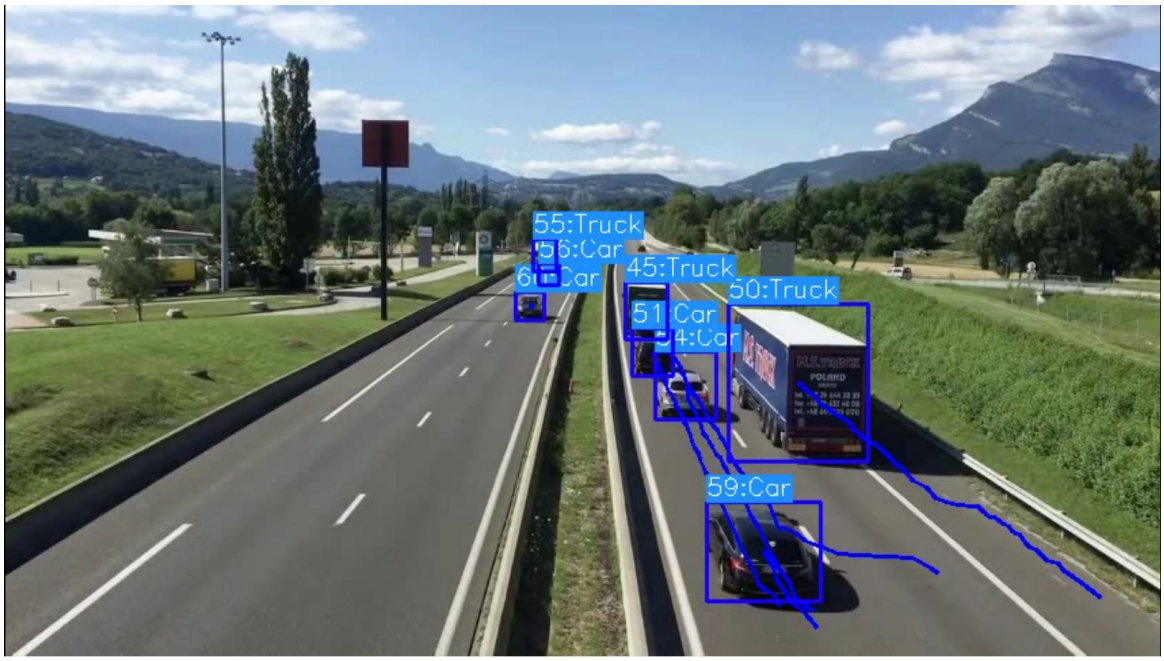


Figure 46, Example of the system tracking the vehicles

The system shall collect traffic and road conditions information.

The system can successfully collect and track the traffic and store the data in a .txt file from the tracking model. As shown in figure 47.

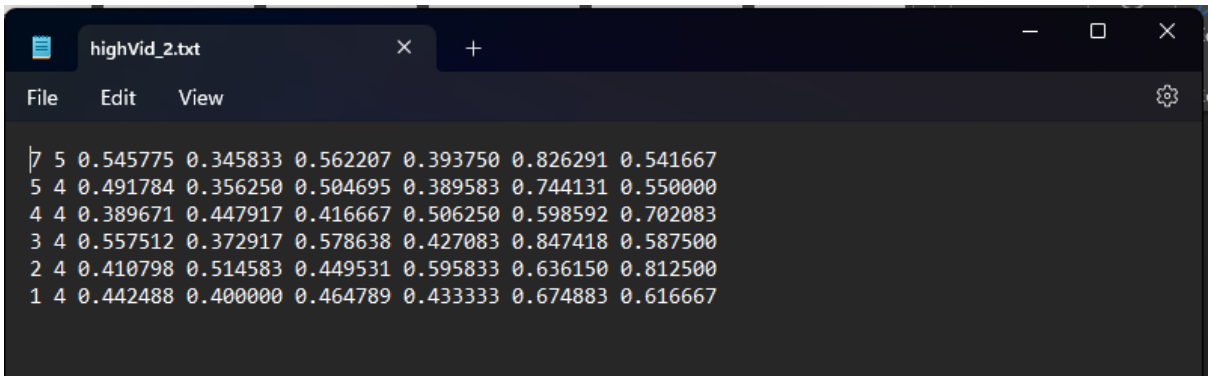


Figure 47, Example of the system collecting traffic data

The system shall store the data coming from the video data.

The system can successfully track the vehicles on YOLOv7 and store their data on CSV files with some data processing. As shown in figure 48, 49.

	Time	Lane 1 Flow (Veh/5 Minutes)	# Lane Points	% Observed
0	2023-06-08 04:22:58	74	1	100
1	2023-06-08 04:23:00	74	1	100
2	2023-06-08 04:23:01	74	1	100
3	2023-06-08 04:23:01	74	1	100
4	2023-06-08 04:23:02	74	1	100
5	2023-06-08 04:23:02	74	1	100
6	2023-06-08 04:23:03	74	1	100
7	2023-06-08 04:23:03	74	1	100

Figure 48, Example of the system storing the data to a CSV file from YOLOv7

	A	B
1	Vehicle ID	Vehicle Type
2	6	Car
3	5	Car
4	4	Car
5	3	Car
6	2	Car
7	1	Car
8	7	Truck
9	14	Car
10	15	Car
11	13	Car
12	16	Car
13	17	Car
14	19	Car
15	18	Car
16	20	Car
17	21	Car
18	22	Truck
19	23	Car
20	24	Car

Figure 49, Example of the system storing the data to a CSV file from YOLOv7

The system shall predict traffic flow while minimizing error.

The system can successfully predict traffic flow with Root Mean Square Error of 0.05. As shown in figure 50.

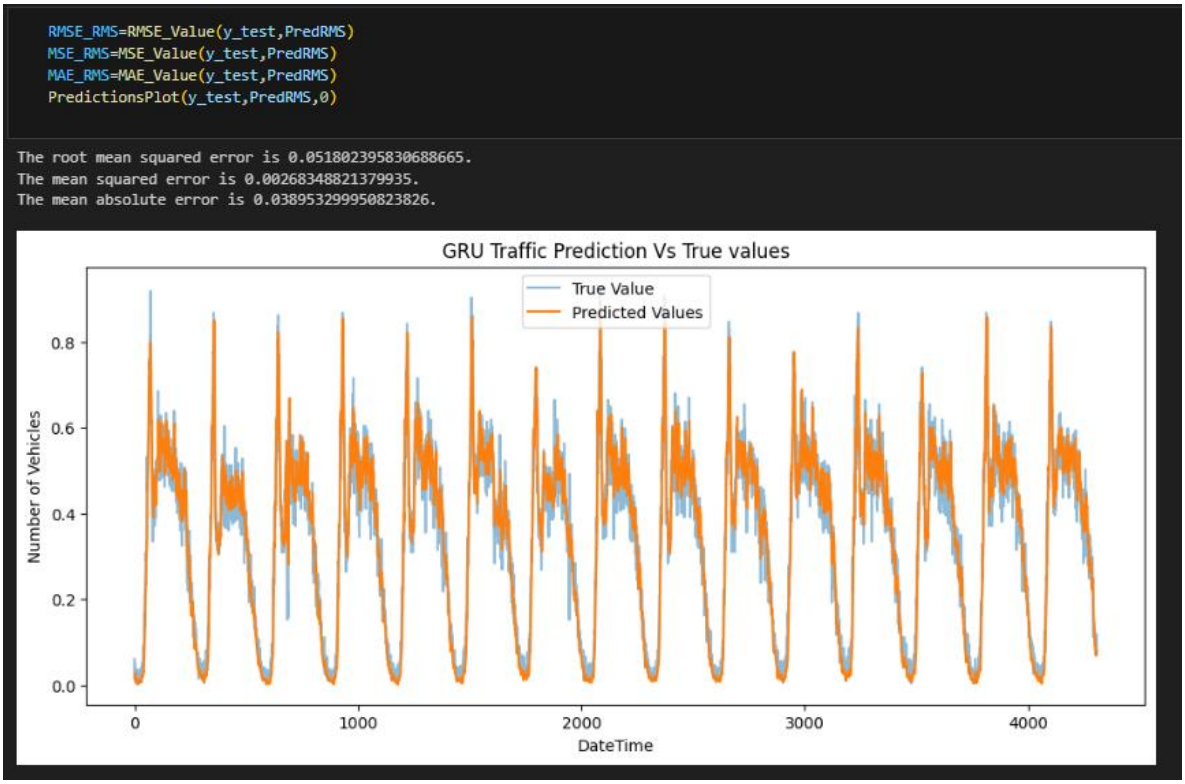


Figure 50, Example of the system training and predicting the traffic flow with minimal error

The system shall recommend either taking this route or not based on the prediction.

The system has a threshold that if the average prediction is exceeded, the system recommends finding another route to escape congestion. As shown in figure 51.

```
model=load_model('TrafficFlowPred\RMSmodel.h5')

scaler = MinMaxScaler(feature_range=(0, 1)).fit(data['Lane 1 Flow (Veh/5 Minutes)'].values.reshape(-1, 1))
flow = scaler.transform(data['Lane 1 Flow (Veh/5 Minutes)'].values.reshape(-1, 1)).reshape(1, -1)[0]

test = []
for i in range(12, len(flow)):
    test.append(flow[i - 12: i + 1])

test = np.array(test)

X_test = test[:, :-1]
y_test = test[:, -1]


pred = model.predict(X_test)
pred = scaler.inverse_transform(pred.reshape(-1, 1)).reshape(1, -1)[0]
y_preds = []
y_preds.append(pred)
print(y_preds)
average=mean(list(chain.from_iterable(y_preds)))
if((average)>=50):
    print("The Street is congested, We recommend taking another way")
else:
    print("The Street is not congested, We recommend using it more")
```

```
1/1 [==============================] - 2s 2s/step
[array([73.99534], dtype=float32)]
The Street is congested, We recommend taking another way
```

Figure 51, Example of the system recommending another route as the road is congested (over the threshold)

5.2 Evaluation

5.2.1 First Scenario

In this scenario, the user runs YOLOv7 model for tracking and detecting the vehicle on the user's custom video.

The user runs the command, and the user can see the video tracking instantly on an average FPS of 21 depending on the density of the traffic and things to process.

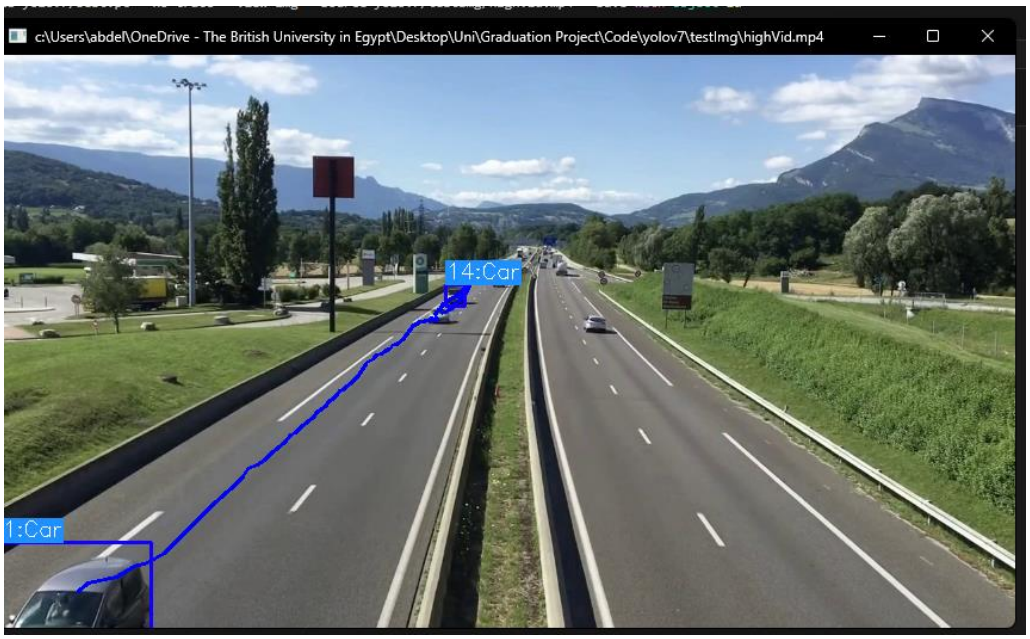


Figure 52, YOLOv7 running tracking on a custom video

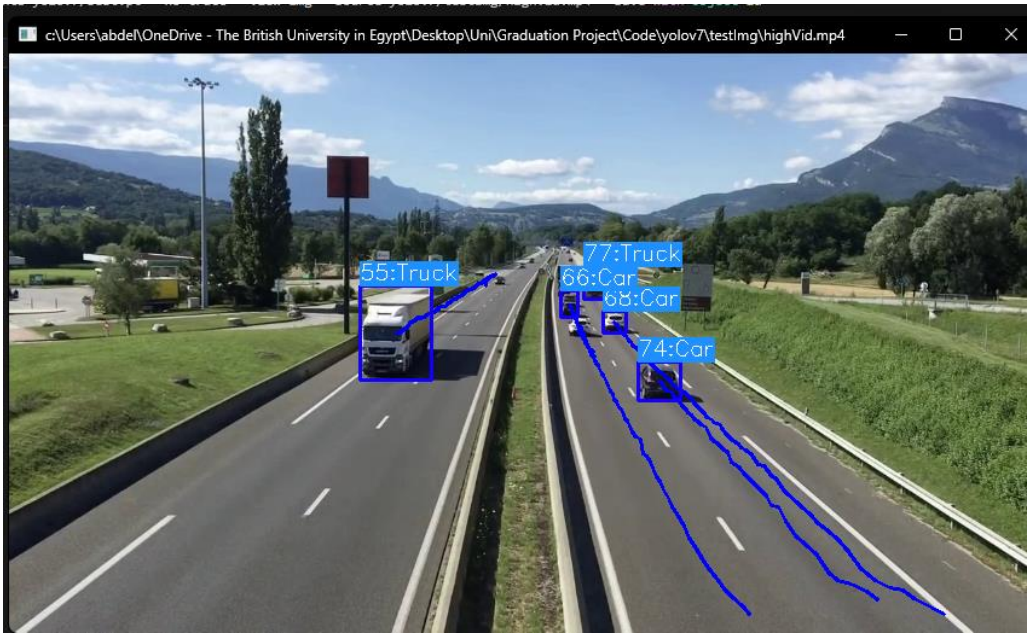


Figure 53, YOLOv7 running tracking on a custom video

After the model tracking and detection is done, the user has the data available to him, but it is still unprocessed, so it is hard to read on the model.

The user runs some data processing to get readable data from the model and use it.

```
folder_path='runs\detect\object_tracking\labels'
batch_folder='data/batches'
batchLabels_folder='data/batchLabels'
```

```
mainUtils.files_into_batches(folder_path, batch_folder)
```

```
mainUtils.process_txt(batch_folder, batchLabels_folder)
```

```
mainUtils.VCount_CSV(batchLabels_folder, 'data/count.csv')
```

```
mainUtils.VType_csv(batchLabels_folder, 'data/type.csv')
```

```
data=pd.read_csv('data/count.csv')
```

```
data
```

	Time	Lane 1 Flow (Veh/5 Minutes)	# Lane Points	% Observed
0	2023-06-09 17:44:34	14	1	100
1	2023-06-09 17:44:34	17	1	100
2	2023-06-09 17:44:34	4	1	100
3	2023-06-09 17:44:34	6	1	100
4	2023-06-09 17:44:34	1	1	100
5	2023-06-09 17:44:34	2	1	100
6	2023-06-09 17:44:34	4	1	100
7	2023-06-09 17:44:34	4	1	100
8	2023-06-09 17:44:34	3	1	100
9	2023-06-09 17:44:34	3	1	100
10	2023-06-09 17:44:34	6	1	100
11	2023-06-09 17:44:34	7	1	100
12	2023-06-09 17:44:34	3	1	100

Figure 54, CSV file after some processing to the data outputted from the model

After processing the data from the model, the user is ready to pass the data to the GRU model for traffic flow prediction. But for the model to be able to compute on said data, the user runs some pre-processing functions to scale the data for the GRU model.

After the pre-processing is done, the user runs the model on the data, the model predicts the traffic flow of how many cars are going to pass through that road, the model decides that the road will not be congested and that it is better to take this route because the average will is lower than the threshold which is 25 because of the nature of the street, but the threshold can be changed in case of a bigger street or a highway for example.

```
model=load_model('TrafficFlowPred\RMSmodel.h5')

scaler = MinMaxScaler(feature_range=(0, 1)).fit(data['Lane 1 Flow (Veh/5 Minutes)'].values.reshape(-1, 1))
flow = scaler.transform(data['Lane 1 Flow (Veh/5 Minutes)'].values.reshape(-1, 1)).reshape(1, -1)[0]

test = []
for i in range(12, len(flow)):
    test.append(flow[i - 12: i + 1])

test = np.array(test)

X_test = test[:, :-1]
y_test = test[:, -1]


pred = model.predict(X_test)
pred = scaler.inverse_transform(pred.reshape(-1, 1)).reshape(1, -1)[0]
y_preds = []
y_preds.append(pred)
print(y_preds)
average=mean(list(chain.from_iterable(y_preds)))
if((average)>=50):
    print("The Street is congested, We recommend taking another way")
else:
    print("The Street is not congested, We recommend using it more")
```

```
1/1 [==============================] - 2s 2s/step
[array([6.5938582], dtype=float32)]
The Street is not congested, We recommend using it more
```

Figure 55, Example of the GRU prediction model suggesting using the road because it is not congested

5.2.2 Second Scenario

In this scenario, the user runs YOLOv7 model for tracking and detecting the vehicle on the user's custom video.

The user runs the command, and the user can see the video tracking instantly on a less average FPS than in scenario one as the density of the traffic and things to process are higher than the first video.

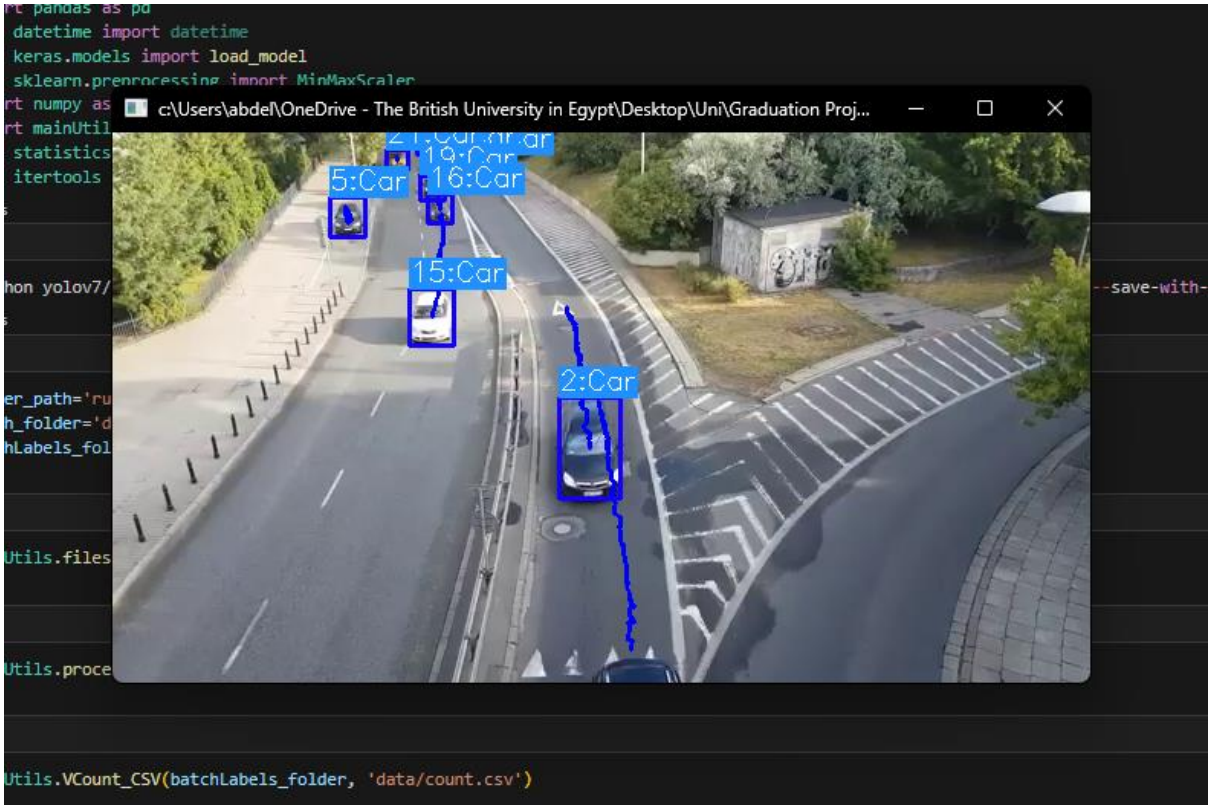


Figure 56, YOLOv7 running tracking on a custom video

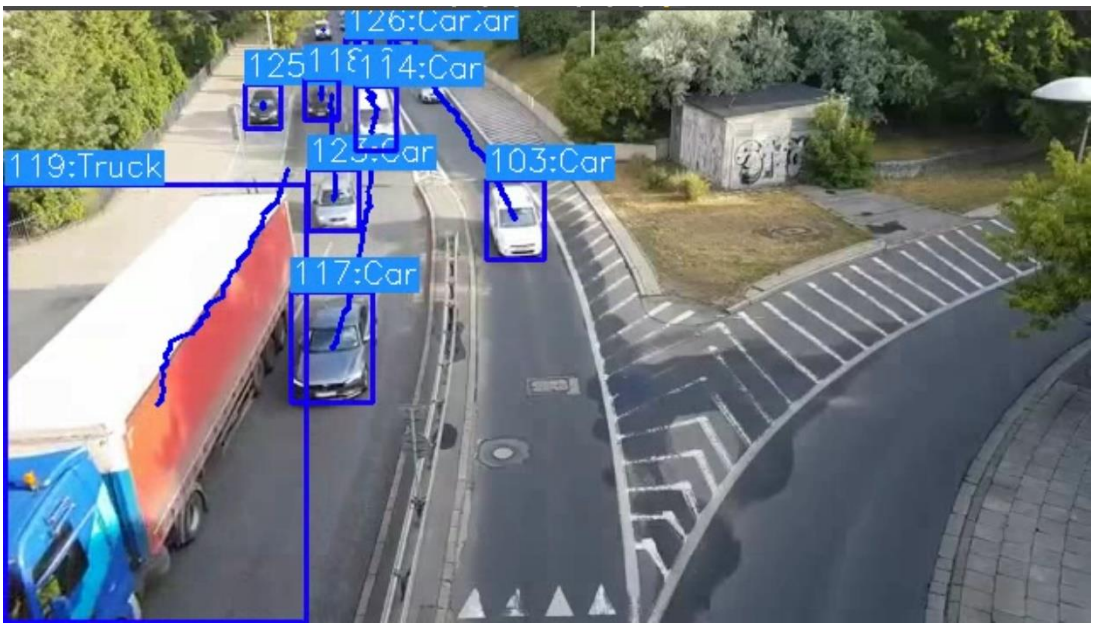


Figure 57, YOLOv7 running tracking on a custom video

After the model tracking and detection is done, the user has the data available to him, but it is still unprocessed, so it is hard to read on the model.

The user runs some data processing to get readable data from the model and use it.

```
folder_path='runs\detect\object_tracking2\labels'
batch_folder='data/batches2'
batchLabels_folder='data/batchLabels2'
```
✓ 0.0s

```
mainUtils.files_into_batches(folder_path, batch_folder)
```
✓ 23.6s

```
mainUtils.process_txt(batch_folder, batchLabels_folder)
```
✓ 17.1s

```
mainUtils.VCount_CSV(batchLabels_folder, 'data/count2.csv')
```
✓ 0.0s

```
mainUtils.VType_csv(batchLabels_folder, 'data/type2.csv')
```
✓ 0.0s

```
data=pd.read_csv('data/count2.csv')
```
✓ 0.0s

```
data
```
✓ 0.0s

	Time	Lane 1 Flow (Veh/5 Minutes)	# Lane Points	% Observed
0	2023-06-10 06:58:29	52	1	100
1	2023-06-10 06:58:29	25	1	100
2	2023-06-10 06:58:29	19	1	100
3	2023-06-10 06:58:29	18	1	100
4	2023-06-10 06:58:29	15	1	100
5	2023-06-10 06:58:29	52	1	100
6	2023-06-10 06:58:29	32	1	100
7	2023-06-10 06:58:29	30	1	100
8	2023-06-10 06:58:29	35	1	100
9	2023-06-10 06:58:29	38	1	100
10	2023-06-10 06:58:29	25	1	100
11	2023-06-10 06:58:29	22	1	100
12	2023-06-10 06:58:29	12	1	100

Figure 58, CSV file after some processing to the data outputted from the model

After processing the data from the model, the user is ready to pass the data to the GRU model for traffic flow prediction. But for the model to be able to compute on said data, the user runs some pre-processing functions to scale the data for the GRU model.

After the pre-processing is done, the user runs the model on the data, the model predicts the traffic flow of how many cars are going to pass through that road, the model decides that the road will be congested and that it is better to take another route because the average was higher than the threshold which is 25 because of the nature of the street, but the threshold can be changed in case of a bigger street or a highway for example.

```
model=load_model('TrafficFlowPred\RMSmodel.h5')

scaler = MinMaxScaler(feature_range=(0, 1)).fit(data['Lane 1 Flow (Veh/5 Minutes)'].values.reshape(-1, 1))
flow = scaler.transform(data['Lane 1 Flow (Veh/5 Minutes)'].values.reshape(-1, 1)).reshape(1, -1)[0]

test = []
for i in range(12, len(flow)):
    test.append(flow[i - 12: i + 1])

test = np.array(test)

X_test = test[:, :-1]
y_test = test[:, -1]


pred = model.predict(X_test)
pred = scaler.inverse_transform(pred.reshape(-1, 1)).reshape(1, -1)[0]
y_preds = []
y_preds.append(pred)
print(y_preds)
average=mean(list(chain.from_iterable(y_preds)))
if((average)>=25):
    print("The Street is congested, We recommend taking another way")
else:
    print("The Street is not congested, We recommend using it more")
```

✓ 4.6s

```
1/1 [==============================] - 2s 2s/step
[array([29.38342], dtype=float32)]
The Street is congested, We recommend taking another way
```

Figure 59, Example of the GRU prediction model suggesting avoiding the road because it is congested

6. Conclusion and Future Work

6 Conclusions and Future Work

This chapter summarizes the key findings and contributions of the research. It reflects on the effectiveness and potential impact of the proposed system. Future directions and areas for improvement.

6.1 Summary

In conclusion, the intelligent transportation system is designed to improve traffic management and provide real-time information about traffic conditions. For vehicle recognition and tracking, the system makes use of computer vision techniques, especially the YOLOv7 model which showed that there is a lot of potential, as the model was fast, accurate and can be used in real-time applications. It also incorporates a GRU model for traffic flow prediction which was discovered to be very accurate with RMSE of around 0.05. The YOLOv7 model is used in the implementation phase to process video data and precisely categorize cars with accuracy of 84.7% accuracy. The system successfully detects and tracks vehicles, assigns them distinctive IDs, and collects traffic and road conditions data. The collected data is stored in CSV and .txt files after some processing for further analysis. The system's traffic flow prediction component utilizing the GRU model uses the processed vehicle data to predict the number of cars that will pass through a particular road. To decide whether the road will be congested or not, the system applies a threshold of 25 that can be changed depending on the nature of the road. Based on the prediction and the threshold, the system recommends either taking the route or finding an alternative to avoid congestion. In the testing and evaluation phase, the system demonstrates satisfactory performance in vehicle detection accuracy, tracking efficiency, and traffic flow prediction. Nevertheless, significant drawbacks are noted, including challenges with precisely identifying automobiles in pixelated or hazy photos and confusion between white SUVs and trucks.

Overall, the implemented intelligent transportation system demonstrates promising abilities in vehicle detection, tracking, and traffic flow prediction, and providing recommendations based on the predictions. The system has the potential to enhance the effectiveness of transportation networks and offers useful real-time information for traffic management.

6.2 Future Work

While the implemented intelligent transportation system demonstrates promising capabilities, there are several avenues for future improvement and expansion. Enhancing the system's ability to detect vehicles in challenging scenarios, such as pixelated or unclear images, would contribute to more accurate and reliable results. Extending the system to incorporate data from multiple sources, including not only video but also GPS data, social media feeds using NLP, or weather information, would enable a more comprehensive analysis of transportation networks. Using a simulator or a real-life example might give more thorough and accurate results. Improving the user interface of the system to provide intuitive visualizations, interactive features, and incorporating the system with a more comprehensive route options to suggest to the user would improve user experience.

References

[1] Greenhouse Gas Emissions from a Typical Passenger Vehicle (2022) EPA. Environmental Protection Agency. Available at: https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle.

[2] Egypt number of registered vehicles (no date) CEIC. Available at: https://www.ceicdata.com/en/egypt/number-of-registered-vehicles-annual/no-of-registered-vehicles.

[3] P. Puvanachandra, C. Hoe, H. F. El-Sayed, R. Saad, N. Al-Gasseer, M. Bakr & A. A. Hyder (2012) Road Traffic Injuries and Data Systems in Egypt: Addressing the Challenges, Traffic Injury Prevention, 13:sup1, 44-56, DOI: 10.1080/15389588.2011.639417

[4] C. Creß, Z. Bing, and A. C. Knoll, Intelligent Transportation Systems Using External Infrastructure: A Literature Survey, 2022.

[5] M. Dotoli, M. P. Fanti, and C. Meloni, "Real time optimization of traffic signal control: application to coordinated intersections," SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme – System Security and Assurance (Cat. No.03CH37483), 2003, pp. 3288-3295 vol.4, DOI: 10.1109/ICSMC.2003.1244397.

[6] Chen Wenjie, Chen Lifeng, Chen Zhanglong and Tu Shiliang, "A Realtime dynamic traffic control system based on wireless sensor network," 2005 International Conference on Parallel Processing Workshops (ICPPW'05), 2005, pp. 258-264, DOI: 10.1109 ICPPW.2005.16.

[7] Queen, C.M., Albers, C.J., Forecasting traffic flows in road networks: a graphical dynamic model approach, 29 July 2008

[8] A. Sharma, R. Chaki, and U. Bhattacharya, "Applications of wireless sensor network in Intelligent Traffic System: A review," 2011 3rd International Conference on Electronics Computer Technology, 2011, pp. 53-57, DOI: 10.1109/ICECTECH.2011.5941955.

[9] D. Srinivasan, M. C. Choy, and R. L. Cheu, "Neural Networks for Real-Time Traffic Signal Control," in IEEE Transactions on Intelligent Transportation Systems, vol. 7, no. 3, pp. 261-272, Sept. 2006, DOI: 10.1109/TITS.2006.874716.

[10] G. S. Khekare and A. V. Sakhare, "A smart city framework for intelligent traffic system using VANET," 2013 International Mutli-Conference on Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013, pp. 302-305, DOI: 10.1109/iMac4s.2013.6526427.

[11] G. S. Khekare and A. V. Sakhare, "A smart city framework for intelligent traffic system using VANET," 2013 International Mutli-Conference on Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013, pp. 302-305, DOI: 10.1109/iMac4s.2013.6526427.

[12] Promila Sinhmar, A.: Intelligent traffic light and density control using IR sensors and microcontroller. Int. J. Adv. Technol. Eng. Res. 2(2), 30–35 (2012)

[13] Srivastava, M.D., Prerna, Sachin, S., Sharma, S., Tyagi, U.: Smart traffic control system using PLC and SCADA. Int. J. Innov. Sci. Eng. Technol. 1(2), Dec 2012

[14] Cyrille, B., Antoine, D., Sylvain, L., Damien, O.: Road traffic management based on ant system and regulation method (2006)

[15] Amine Kafi, M., Challal, Y., Djenouri, D., Bouabdallah, A., Khelladi, L., Badache, N.: A study of wireless sensor network architectures and projects for traffic light monitoring. In: International Conference on Ambient Systems, Networks and Technologies, pp. 543–552, 28 Aug 2012

[16] Ozkurt, C., Camci, F.: Automatic traffic density estimation and vehicle classification for traffic surveillance systems using neural network. Math. Comput. Appl. 14(3), 187–196 (2009)

[17] Shiuan-Wen, C., Chang-Biau, Y., Yung-Hsing, P.: Algorithms for the traffic light setting problem on the graph model (1996)

[18] S. P. Biswas, P. Roy, N. Patra, A. Mukherjee, and N. Dey, “Intelligent Traffic Monitoring System,” Advances in Intelligent Systems and Computing, pp. 535–545, 2015.

[19] Azura Che Soh/Lai Guan Rhung: MATLAB simulation of fuzzy traffic controller for multilane isolated intersection. Int. J. Comput. Sci. Eng. 02(04), 924–933 (2010)

[20] Emad, I., Kareem, A., Jantan, A.: An intelligent traffic light monitor system using an adaptive associative memory. Int. J. Inf. Process. Manag. 2, 2(2.4), 23–39 (2011)

[21] Jaiswal, S., Agarwal, T., Singh, A., Lakshita, Intelligent traffic control unit. Int. J. Electr. Electron. (ISSN NO. (O line): 27-2626 and Computer Engineering 2(2), 6–72 (2013)

[22] S. Bougharriou, F. Hamdaoui and A. Mtibaa, "Linear SVM classifier-based HOG car detection," 2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), Monastir, Tunisia, 2017, pp. 241-245, DOI: 10.1109/STA.2017.8314922

[23] N. Ammour, H. Alhichri, Y. Bazi, B. Benjdira, N. Alajlan, and M. Zuair, “Deep Learning Approach for Car Detection in UAV Imagery,” *Remote Sensing*, vol. 9, no. 4, p. 312, Mar. 2017, DOI: 10.3390/rs9040312.

[24] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, and K. Ouni, "Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3," *2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)*, Muscat, Oman, 2019, pp. 1-6, DOI: 10.1109/UVS.2019.8658300.

[25] D. N. -N. Tran, L. H. Pham, H. -H. Nguyen and J. W. Jeon, "City-Scale Multi-Camera Vehicle Tracking of Vehicles based on YOLOv7," *2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, Yeosu, Korea, Republic of, 2022, pp. 1-4, DOI: 10.1109/ICCE-Asia57006.2022.9954809.

[26] C. Wang, A. Bochkovskiy, and H. Mark Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," arXiv preprint arXiv:2207.02696, 2022.

[27] R. Fu, Z. Zhang, and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," in 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), pp. 324-328, 2016, DOI: 10.1109/YAC.2016.7804912.

[28] A. A. Kashyap, S. Raviraj, A. Devarakonda, S. R. N. K, S. K. V, and S. J. Bhat, "Traffic flow prediction models – A review of deep learning techniques," Cogent Engineering, vol. 9, no. 1, p. 2010510, 2022.

[29] "Ambulance detection Dataset," Open-Source Dataset, ell205, Roboflow Universe, Roboflow, Nov. 2022. Available: https://universe.roboflow.com/ell205/ambulance_detection-zfeve.

[30] "Thesis Dataset," Open-Source Dataset, Thesis, Roboflow Universe, Roboflow, Mar. 2023. Available: https://universe.roboflow.com/thesis-prblp/thesis-dataset-yvjdb.

[31] "Vehicle detection dataset," Open-Source Dataset, University of Siena, Roboflow Universe, Roboflow, Sep. 2022. Available: https://universe.roboflow.com/university-of-siena-dls9l/vehicle-detection-dataset.

[32] "Person dataset," Open-Source Dataset, SADI0v2, Roboflow Universe, Roboflow, Jan. 2023. Available: https://universe.roboflow.com/sadi0v2/person-dataset-burgs.

[33] "Vehicle Detection Dataset," Open-Source Dataset, L. T. Hau, Roboflow Universe, Roboflow, Feb. 2023. Available: https://universe.roboflow.com/le-trung-hau/vehicledetection-ik8bx.

[34] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," CoRR, vol. abs/1602.00763, 2016. Available: http://arxiv.org/abs/1602.00763.

[35] “PeMS 5min-interval traffic flow data,” Caltrans. Available: https://dot.ca.gov/programs/traffic-operations/mpr/pems-source.

Appendix I

The following script was for pre-processing the training data to make all the different training data compatible with the model

```
import os
input_folder = r"C:\Users\abdel\OneDrive - The British University in Egypt\Desktop\train/labels"
output_folder = r"C:\Users\abdel\OneDrive - The British University in Egypt\Desktop\train\labelsNew"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)
for filename in os.listdir(input_folder):
    if filename.endswith('.txt'):
        input_file = os.path.join(input_folder, filename)
        output_file = os.path.join(output_folder, filename)
        with open(input_file, 'r') as input_f, open(output_file, 'w') as output_f:
            for line in input_f:
                if line.strip():
                    first_char = line[0]
                    if first_char == '0':
                        modified_line = '2' + line[1:]
                    elif first_char == '1':
                        modified_line = '4' + line[1:]
                    elif first_char == '2':
                        modified_line = '1' + line[1:]
                    elif first_char == '3':
                        modified_line = '5' + line[1:]
                    else:
                        modified_line = line
                    output_f.write(modified_line)
```

The following code is for building the GRU model

```
def RMSGRU_model(X_Train, y_Train, X_Test):
    early_stopping = callbacks.EarlyStopping(min_delta=0.001,patience=10,
restore_best_weights=True)
    #The GRU model
    model = Sequential()
    model.add(GRU(units=150, return_sequences=True, input_shape=(X_Train.shape[1],1),
activation='tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(units=150, return_sequences=True, input_shape=(X_Train.shape[1],1),
activation='tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(units=50, return_sequences=True, input_shape=(X_Train.shape[1],1),
activation='tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(units=50, return_sequences=True, input_shape=(X_Train.shape[1],1),
activation='tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(units=50, return_sequences=True,
input_shape=(X_Train.shape[1],1),activation='tanh'))
    model.add(Dropout(0.2))
    model.add(GRU(units=50, input_shape=(X_Train.shape[1],1), activation='tanh'))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))
    #Compiling the model
    model.compile(optimizer=tf.keras.optimizers.legacy.RMSprop(decay=1e-7,
momentum=0.9),loss='mean_squared_error')
    model.fit(X_Train,y_Train, epochs=50, batch_size=150,callbacks=[early_stopping])
    model.save('RMSmodel.h5') # save the trained model to a file
    pred_GRU= model.predict(X_Test)
    return pred_GRU
```