# EEEE4008

# Final Year Individual Project Thesis

# Connected and Autonomous Vehicles


**AUTHOR:** **Mr Jonathan Adu-Sarkodie**

**ID NUMBER:** **14333262**

**SUPERVISOR:** **Dr. Steve Greedy**

**MODERATOR:** **Dr. Eric Larkins**


**DATE:** **5th May 2022**

Fourth year project report is submitted in part fulfilment of the requirements of the degree of Master of Engineering.

# Abstract

The constant evolvement of technology has reached a point where manufacturers have begun the development of fully autonomous vehicles. The implementation of autonomous vehicles in society aims to address several issues, an example of an issue relates to accidents caused as a result of human input. Or, it can be seen to provide convenience to the public, such as an autonomous taxi service or a delivery service. Regardless of the purpose autonomous vehicles are intended to serve, the most important factor of implementing such vehicles must be ensuring that the vehicles are safe and reliable enough to gain the trust of the public. Manufacturers of autonomous vehicles are using different forms of their own artificial intelligence to achieve a level of autonomy in their projects, involving the generation of a model with the use of Deep Learning and Convolutional Neural Networks (CNNs) to train their model from a dataset. Throughout this report, we delve into what it takes to produce a model capable of detecting objects, more specifically traffic control signs and vehicles. We explore the make-up of existing open source datasets to understand what it takes to produce a dataset that can be trained for optimal performance, as well state of the art methods of object detection currently used in industry for the production of Autonomous Vehicles. From the model produced; we also look at the information that can be extracted from the objects detected to reach a point where a decision can be made, imitating the decision making process of a human, but using Computer Vision. As well as extracting information from the objects detected, we also devise other methods of implementing desired features, such as Lane Detection and observing the speed of moving objects presented in frame. We compare the performance of the model in different lighting conditions and contexts in comparison to a model that was trained using one of the existing open source datasets and explore whether the performance limitations could be rectified with the additions of filters and different methods of data augmentation for images within the dataset used to train the model.

# Contents

# 1 Introduction

As a society, we are approaching a time where the need for human intervention in technology will soon be a thing of the past. With the implementation of Artificial Intelligence (AI) in autonomous technology; we are able to replicate, and even improve the execution of everyday tasks in our lives. To achieve this, we must give electronic devices a “conscience”. We must make these devices “aware” of their own surroundings, so when information is provided to the devices, they can make their own decisions. These decisions must emulate the decision a human would make when presented with the same information. The execution of the decision must also emulate human behaviour. In the context of Autonomous Vehicles; the information provided to them would be hazards related to traffic control, which is then further processed with a form of AI, known as Computer Vision.

The journey for the development of autonomous vehicles started as early as 1925 with the “American Wonder”, which was radio-controlled. The vehicle had to be operated by a human who had to be within a 50-foot radius of the vehicle. [1] Vision-based systems did not appear until the 1980’s, which was a result of Eureka’s PROMETHEUS project (Program for European Traffic with Highest Efficiency and Unprecedented Safety), where a number of car manufacturers, electronics producers and universities came together to develop these systems. [2] The most notable autonomous vehicle from this project was Mercedes-Benz’ S-Class W140, which managed to complete a 1043 mile trip from Munich to Copenhagen, whilst successfully overtaking vehicles with minimal human intervention purely using cameras and sensors. [2] Fast forward to the autonomous vehicle projects of today, there are developments being made on vehicles of Level 5 autonomy (no human assistance required), and members of the public already owning vehicles with a form of autonomy (typically Level 2), provided they are actively engaged with the vehicles activity behind the wheel in the case of the autonomous vehicle making a potentially fatal error.

Throughout this report, we look to investigate how vehicles controlled by Computer Vision are able to recognise their surroundings to make a decision depending on the objects it can recognise. This involves creating a dataset and training it on a convolutional neural network, from which we will critically analyse the performance of the models. We will then investigate how we can potentially improve the dataset to attain a better performing model; as well as analysing scenarios which can strengthen and weaken the performance of the model. The model comes in the form of a software solution that demonstrates object detection of traffic control signs and objects, with more focus on speed limit signs, traffic lights and vehicles. From the objects detected, the software must provide information to the user that demonstrates the software can understand the object it is looking at and thus, provide an instruction that must be taken to obey the rules of the object, i.e. if a red traffic light is detected, the software must overlay an instruction that instructs the user to stop. Using the model, a Heads-Up Display (HUD) can be created to provide a clear interface which indicates what exactly the model has detected, and the required actions that the user should follow. This will be achieved purely using computer vision techniques without the use of any sensors. Provided the performance of the object detection is strong, the software solution can be used in conjunction with a robot car to follow instructions provided by the software solution.

## 1.1 Thesis Map

In Section 2, we delve into a few examples of current autonomous vehicle projects, namely from Tesla, Nissan and Google. From these projects, we investigate the technology and artificial intelligence methods that they use to achieve autonomy in their vehicles. In Section 3, we investigate the technical aspects behind the AI used in the projects discussed in Section 2. Section 4 highlights the software packages to be installed and the installation process to commence the project work. Section 5 explores models that have been created using open source datasets on the object detection framework we use throughout our own project. The generation of our own model is also covered in this section. Section 6 covers the method behind the software solution produced as a result of this project. Section 7 then reviews the final software solution produced in comparison to aforementioned targets set in the Project Proposal.

# 2 Background

Innovate UK are overseeing a project with the vision to fully implement autonomous vehicles as the main form of transportation by 2050 under the name “UK Transport Vision 2050”. The use of autonomous vehicles will contribute to nullifying the carbon footprint of vehicles, as the vehicles will be powered by electricity and other sustainable fuel sources to achieve “net zero” (zero carbon dioxide emissions). [3] As well as the proposed environmental benefits of autonomous vehicles, there is also the proposition of reduced traffic congestion and better traffic flow, which will contribute to the reduction of road accidents. However, these propositions cannot be met unless every vehicle on the road is an autonomous vehicle where no assistance is required by a human behind the wheel. The Society of Automotive Engineers (SAE) categorise the autonomous capabilities of autonomous vehicles using various “Levels”, where Level 0 indicates there is no driving automation, whilst Level 5 indicates full driving automation capabilities with no human assistance. [4] There are several projects currently underway that are taking different approaches to reaching Level 5 automation, including Tesla’s “Autopilot” and “Tesla Vision”, Nissan’s “HumanDrive” and Google’s “Waymo”.

## 2.1 Tesla Autopilot/Tesla Vision


*Figure 1 - Tesla Autopilot Object Detection Model [5]*

An example of an autonomous vehicle that utilises object detection software is Tesla’s Autopilot. Tesla developed their own deep neural network that utilises a network of cameras surrounding the car to get a video feed of the cars surroundings. Using this video feed, the car can detect different objects and distinguish hazards surrounding the car. The use of the deep neural network helps the car to “learn” about its surroundings for millions of different scenarios on the roads from the video feeds, using object detection, semantic segmentation & monocular depth estimation. [6] In

conjunction with this camera system, Tesla's Autopilot used RADAR and LIDAR sensors to assist the car in difficult weather conditions and navigation on the road. However, as of May 2021, Tesla announced that they will stop producing self-driving vehicles that utilise RADAR and LIDAR sensors, meaning that they are shifting to a system that only utilises cameras and neural net processing. Their own implementation of this new system is called Tesla Vision. [7] This change may be in relation to the 24 reported accidents involving their car with the Autopilot system getting into accidents due to the car not being able to recognise semi-trucks, police cars and fire trucks. In one of these accidents, the driver was killed. [8]

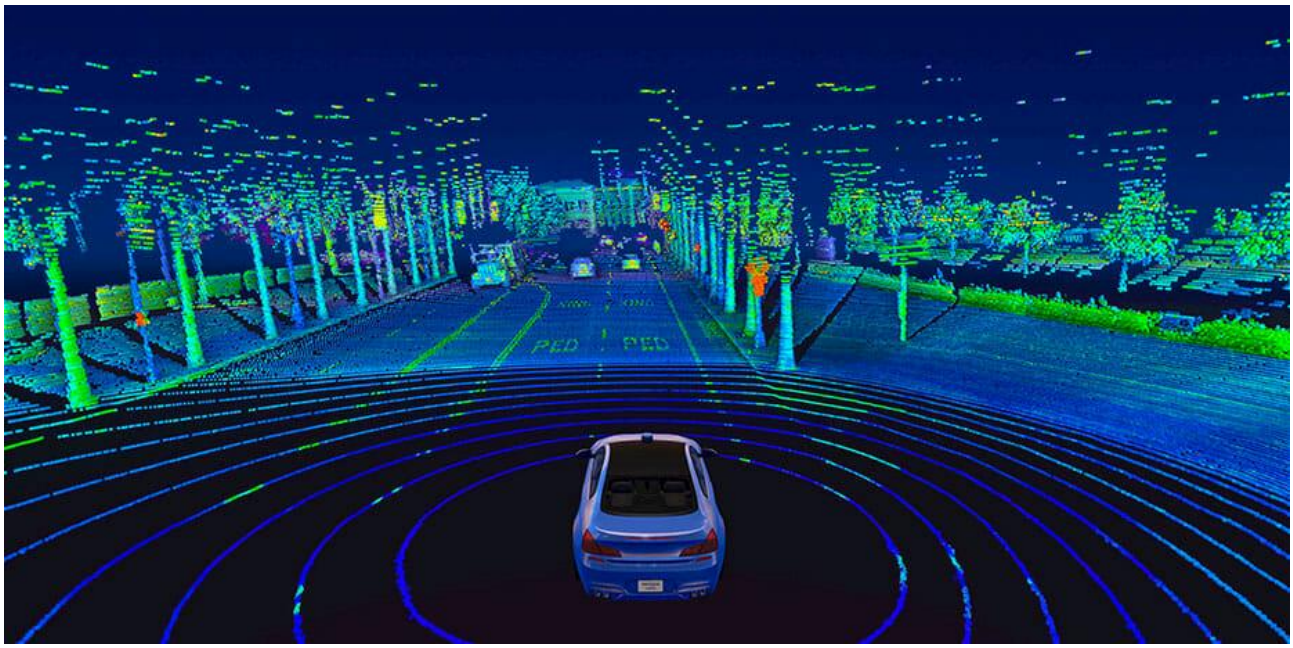


*Figure 2 - LIDAR Point Cloud [9]*

LIDAR (Light detection and ranging) sensors are extremely accurate sensors that provide distance measurements from the source of the sensor. LIDAR uses a laser which is pulsed into the environment, and where it comes into contact with objects in the environment, the laser pulses are reflected back to the sensor. Calculations are then made using the speed of which the light was emitted and the time taken for the pulse to travel to the object and back to the sensor, to obtain the distance of the object from the sensor. [10] Using the speed of light, this process can be repeated millions of times a second to produce a 3D environment for autonomous vehicles.

It is common for autonomous vehicle projects to utilise LIDAR and RADAR simultaneously, however it has been mentioned that LIDAR is complicated to implement in Tesla's desired system, as a crucial aspect of their system involves creating a 3D vector space of the vehicles surroundings, which helps the vehicle have a true understanding of the environment. With computer vision, the vehicle can be trained to fully identify any object, whether it is stationary in the environment or not, whereas LIDAR is said to struggle in these scenarios, as Andrej Karpathy (Senior Director of Artificial Intelligence at Tesla) stated at Tesla's Autonomy Day in 2019, "**LIDAR cannot differentiate a road bump from a plastic bag"**. [11] As a result, software solutions have been developed to replicate the complex functionality of a LIDAR sensor, such as accurately establishing the distance of an object such as a curb in the distance only with information provided by pixels on the cars camera network.

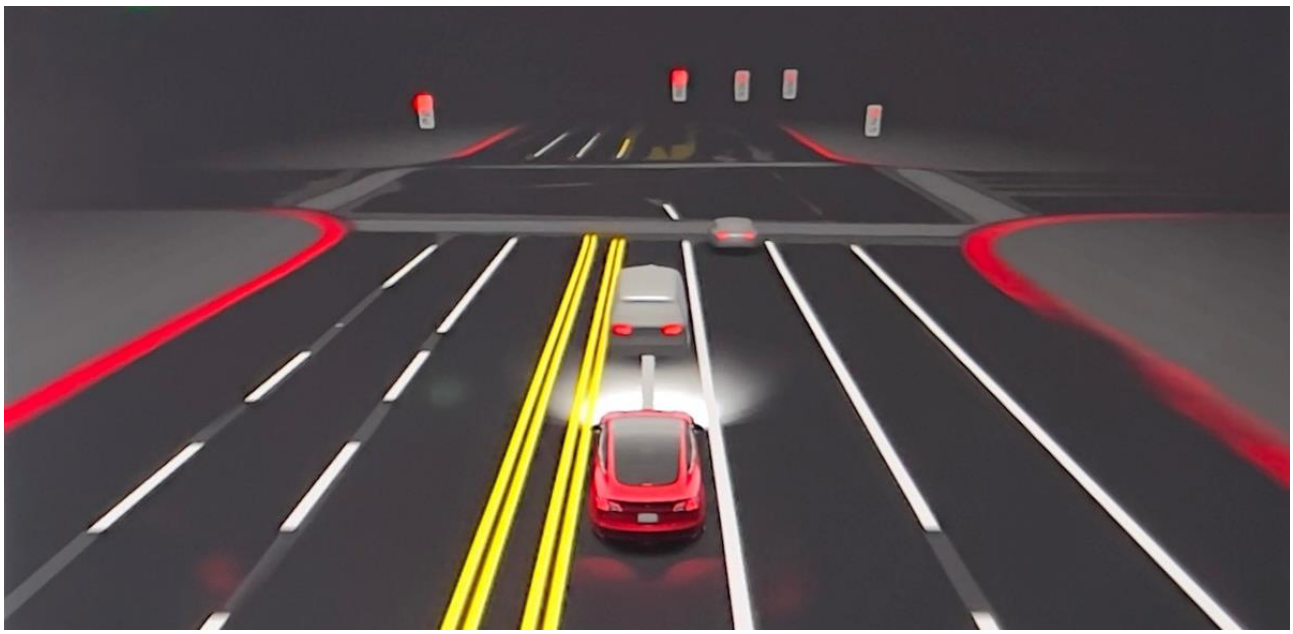

*Figure 3 - Tesla's 3D Vector Space for Autopilot [12]*

Another reason for Tesla to ditch LIDAR is due to the cost of utilising the sensor in its detection network. It has been quoted that each sensor costs approximately $10,000 to implement on each vehicle [11], which does not help with making Tesla's self-driving vehicles more affordable.

## 2.2 HumanDrive

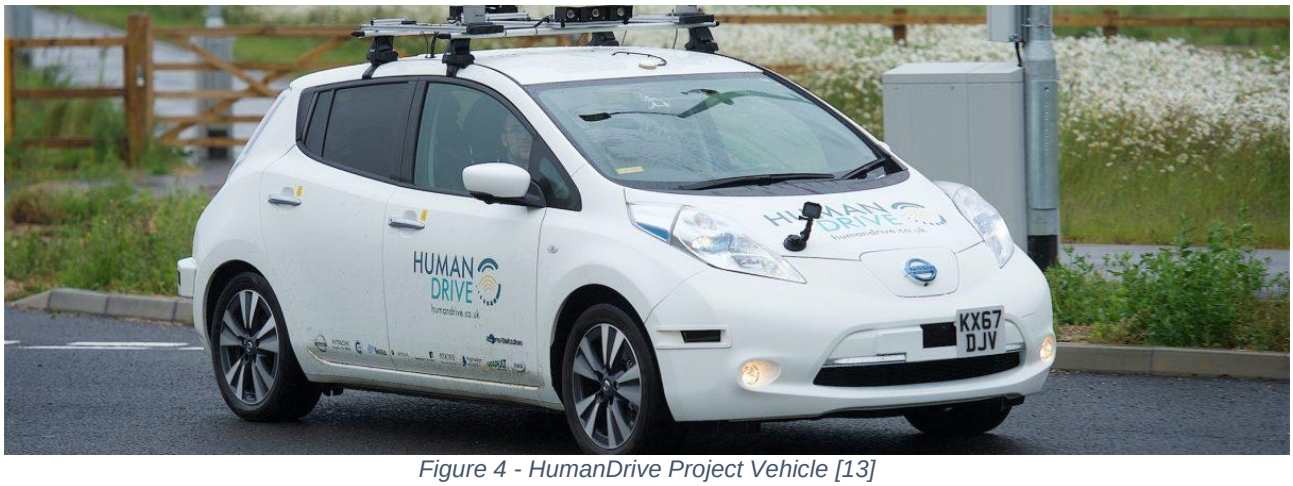


*Figure 4 - HumanDrive Project Vehicle [13]*

An AV project that demonstrates that is HumanDrive, led by Nissan and sponsored by Innovate UK. The vehicle completed a 230 mile drive across the UK without human intervention in November 2019. [14] This would not have been possible without the contribution of Hitachi, who built on a Deep Learning system which was used in this project. Hitachi utilised a "Fast R-CNN" framework (Region-based Convolutional Neural Network) along with stereo cameras to obtain high quality RGB images with a good level of depth that would help the system identify objects in low lighting and poor conditions. The depth information would help generate good region proposals for the RCNN framework, which indicate where a certain object may be in an image. This depth information in conjunction with the camera's "RGB" image formed an "RGB-D" image, which is passed through the deep neural network to achieve real time object detection and classification. [15] The main area of improvement with this system would be performance based. Hitachi intend to experiment with different deep learning architectures such as faster RCNN, retina-nets & YOLO-V3 with the aim to improve object detection and classification speed, as well as improving an images depth information for even more difficult scenarios. [15]

## 2.3 Google Waymo

Unlike Tesla, Google's "Waymo" project does utilise 3D LIDAR and RADAR sensors to produce 3D "point clouds" [16], which is just data points in space, similar to Tesla's 3D vector map. LiDAR by itself isn't useful to determine what object it is detecting the presence of; as shown on the right of Figure 5 below, but it does give very good depth information from the RGB videos provided from Waymo's camera network that is mounted on their vehicles. The depth/distance of the objects surrounding the car is obtained by utilising the calculations discussed earlier in section 2.1.

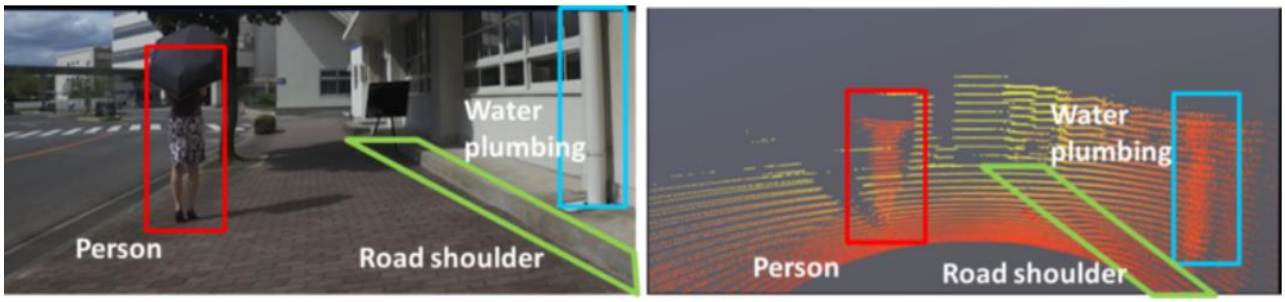


*Figure 5 - RGB image (left) vs LIDAR image (right) [17]*

To combine both LiDAR and RGB videos together, Waymo developed a neural network which they call "4D-Net", which learns to take RGB videos and "Point Clouds in Time" (PCiT) to perform 3D object detection [16] , whilst accurately perceiving the depth of the object from the Waymo vehicle. Both PCiT and RGB video are perceived as 4-dimensional inputs, as previous frames are taken into consideration so the vehicle can fully grasp the context it finds itself in, and thus predictions can be made regarding potential future scenarios that could possibly occur, and decisions can be made in advance, imitating human behaviour. [17] The output of the "4D-Net" architecture shown in Figure 6 are 3D boxes of the objects that the network successfully detected, shown in Figure 7. Waymo vehicles are described to be Level 4 autonomous vehicles [18], where the vehicle can navigate the roads within a limited geographical area without assistance but humans can override the autonomous operation if desired.

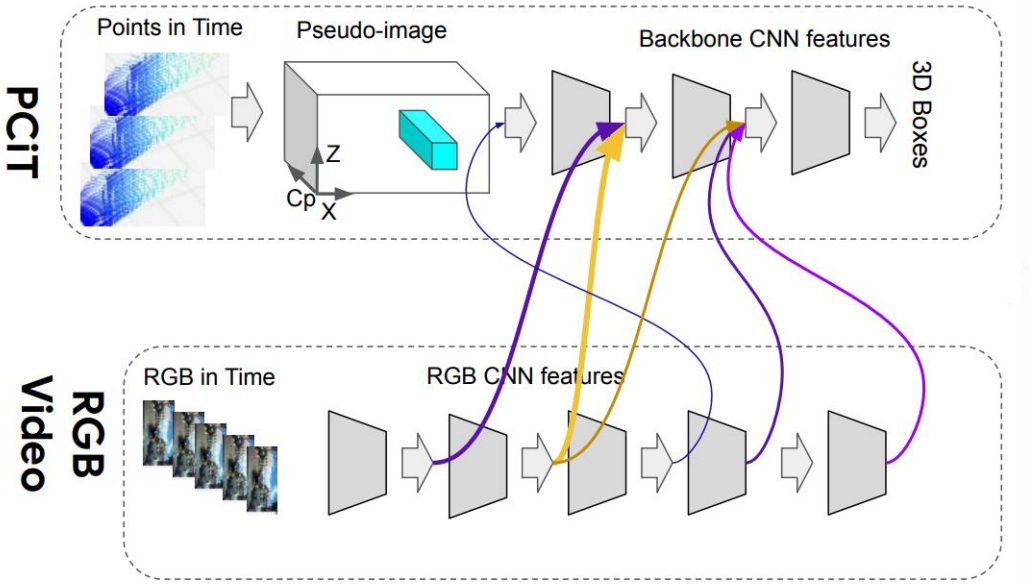


*Figure 6 - "4D-Net" Network Architecture [16]*


*Figure 7 - 3D Boxes Output from 4D-Net architecture [16]*

Google are planning to use their Waymo vehicles to create their own Uber-like service called Waymo One. As of April 2022, Waymo One is already in operation for the public to travel in Phoenix, Arizona and San Francisco, California. [19]

## 2.4 Ethics

The need for autonomous vehicles (AV's) in particular is not seen as a necessity by members of the UK public. A national survey was undertaken between May 2021 – August 2021 by Project Endeavour; a project that are looking to accelerate the deployment of AV's in the UK. [20] They gathered approximately 2500 responses, gaining an understanding of their perception of AV's regarding how much they trust the technology, as well as whether they believe they are safer than the vehicle technology we currently use. The outcome of this survey showed that 55% of these responses would not feel comfortable with the introduction of AV's, whilst 18% are undecided. [20] Regarding the perception of safety of AV's, 44% of these responses believe that AV's will not be safer than vehicles controlled by humans, whereas 30% do. [20] Despite the response of the public, there are some damning statistics which otherwise prove that change must occur to make our roads safer. Data collected by the UK Government's Department for Transport found that in 2020, the top 5 causes of road accidents in the UK are related to decision making as well as speed perception of oncoming vehicles. The top 5 causes are as follows: (1) driver/rider failed to look properly; (2) driver/rider failed to judge other person's path or speed; (3) driver/rider careless or in a hurry; (4) poor turn or manoeuvre; (5) loss of control. [21] Whilst humans continue to remain in control of the vehicle, there is always a great risk of accidents occurring due to natural human error. However, should AV's be the common method of transport in the future, who remains liable for accidents? No official laws have been put in place that answer this query, however a few companies working on fully autonomous vehicles such as Google, Volvo and Mercedes have stated that the companies themselves will hold accountability for any accidents involving their self-driving vehicles. [22] Despite this, is it enough to sway customers into trusting self-driving vehicles? During the development of the AI behind the self-driving cars, can the software ever be 100% reliable? Can the software ever be tested enough to ensure that it is safe for the public to trust and use? Current owners of Tesla's self-driving vehicles have had access to Tesla's public beta software for their "Full Self-Driving" software, which is geared towards vehicles of autonomy level 2 (partial automation). Whilst the drivers consent to using this software, Tesla had to withdraw the software due to issues found in this public testing exercise. [23] Although it is normal for beta software to have issues, it does not give the public confidence that they can invest in a self-driving vehicle at this moment in time. In addition to this, many people own vehicles for the joy that driving gives them, would the public be willing to sacrifice this feeling in favour of being driven?

# 3 Literature Review

To be able to train a model capable of detecting objects in a similar fashion to the projects discussed in Section 2, we must process the environment surrounding the robot using image processing methods. Object detection is achieved by training the "brain" of the robot, which entails training a Convolutional Neural Network (CNN) to detect desired objects with a large volume of images. More artificial intelligence is used to carry out the image processing element of object detection, in this case, Computer Vision. In this section, we will glance over the importance of Computer Vision and the technical aspects of how a Convolutional Neural Network works. The convolutional neural network is a key feature of this project, as it is used to train a model geared to identify the desired object classes via the YOLOv4 algorithm, which is an instance of a CNN that is to be used throughout this project.

## 3.1 Computer Vision

Computer Vision is a form of AI that gives computers the ability to make sense of the world we live in by processing images/videos. Computer Vision is utilised in many applications, including autonomous vehicles to help them to identify everything that is necessary on the road, whether it be traffic signs, road markings and hazards such as pedestrians and oncoming traffic. Autonomous vehicles have built in cameras to help them "see" their surroundings, from which the video feed is analysed and objects are detected and classified. For the autonomous vehicles to "learn" how to distinguish between different objects or the same object which may have a slightly different appearance, a form of Deep Learning, the Convolutional Neural Network (CNN) is used. [24]

### 3.1.1 Deep Learning

Deep Learning is a subset of Machine Learning which tries to replicate how the brain perceives and learns data. Thanks to the advances in modern day technology with regards to improved GPU performance, cheaper cost of high end performing hardware and the improved ability of chip performance, deep learning has become much more popular when it comes to performing various computer vision tasks. [25] Deep Learning mainly utilises two processes; nonlinear processing and supervised/unsupervised learning depending on the application. Nonlinear processing is the event where the algorithm takes the output of a previous layer to use as an input to generate an output for the next layer. [26] Supervised/Unsupervised learning is the parameter that states how the algorithm trains on a set of data.

Deep Learning algorithms can be utilised via Convolutional Neural Networks (CNN's), which we will be exploring throughout this project. CNNs are proven to be highly effective and can be used in a wide variety of computer vision applications.

#### 3.1.1.1 Supervised Learning

With Deep Learning, models are trained to detect specific objects dependent on the classes (objects) within the dataset. For the algorithm to train "supervised", the input dataset (X) must be labelled with class information. The algorithm would "learn" what an object by mapping (f) the input to the output (Y), which can be described in equation 1 below: [27]

$$Y = f(X) \tag{1}$$

The input dataset would contain information alongside the input images. An example of this are usually labelled pictures, with the labels consisting of information like the class name as well as coordinates of a manually annotated bounding box of where that class is in the input image.

During training, what should be the correct outcome is already known, and any errors during training can be corrected until the algorithm reaches a good enough performance when identifying a particular class.

*Figure 8 - Manually annotated bounding box*

Supervised Learning are usually categorised into two types of problems: [27]

- Classification – The output is usually a categorical variable, such as Yes or No, or Red/Blue/Green.
- Regression – The output variable is a real value.

For an algorithm learning from labelled data, the key to having a good performing algorithm is by having a huge dataset, ideally on a scale of millions of images and even more class instances within those images. The more data presented to the algorithm to learn from, the better the performance. The more diverse the data is, even better performance of the algorithm can be achieved as it would be prepared to detect the object from a wider range of scenarios. There are methods that can be used to generate a wider variety of images via data augmentation, which will be covered in a later segment of this report.

#### 3.1.1.2 Unsupervised Learning

Unsupervised Learning has a similar concept to supervised learning, however there is no labels to the data which the algorithm learns from, it is up to the algorithm to determine ways of identifying a class, and what makes that class unique to other classes. Unsupervised Learning problems are categorised in the following ways: [27]

- Association: This gives a prediction of what may occur based on other parameters within the input data, i.e. what customers may purchase online dependent on their age group, ethnicity and where they live.
- Clustering: This gives an idea of similarities in your data which can be grouped together.

#### 3.1.1.3 Semi Supervised Learning

Semi Supervised Learning takes a combination of both supervised & unsupervised learning to train the model. It splits the training data to have a small proportion of labelled data and a larger proportion of unlabelled data to train the model. It is said that using a combination of both labelled and unlabelled data can provide a more reliable estimate of the “decision boundary”, which is defined as the boundary separating decision regions where points are given the same label in a dimensional space. [28]

### 3.1.2 Convolutional Neural Network

The CNN architecture consists of an input layer, an output layer, and several layers in between consisting of “neurons" that are used to help “learn” the distinct features of an object in an image. [29] More details regarding how these layers operate can be found below.

### 3.1.2.1 Input Layer

The input layer of a CNN holds the image which the CNN will be "learning" distinct features of. It is important that the input image matches the desired attributes for what the CNN can work with, which is the size of the image (height (H) & width (W)) and the depth of the image (D), which refers to its "activation volume". [30] This gives the network a platform from which it can derive "weights" (HxWxD).

### 3.1.2.2 Convolution

Convolutions are the "building blocks" to the Convolutional Neural Network (CNN). A convolution is the idea of expressing the amount of "overlap" a function has as it is overlapping another function. [31] In this application, a convolution could be described to act as a filter to an input that results in "activations". The input of a CNN is typically an image of a set size, so an activation in this case could be a distinct feature of an object which is presented in the input image. The convolutional layer looks at regions of the image and makes connections based off a distinct feature of the object in the input image, which is passed onto a neuron within the next hidden layer. This is essentially applying several filters to the image, with each filter aiming to isolate a new distinct feature which belongs to the object in question. Each filter is "convolved" with the input, activating the distinct feature from the input image. This process takes place across the entire input image to create a "2D Activation Map". [30] This is efficient as it means that these features can be identified no matter where it is in the image.

### 3.1.2.3 Rectified linear unit (ReLU)

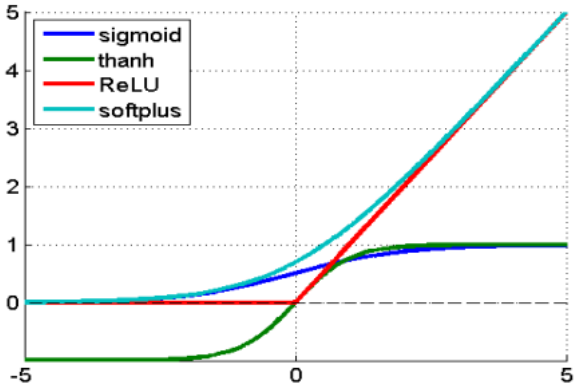


*Figure 9 - Common Non-Linearity Functions [32]*

$$\mathrm{ReLU(x)} = \max(0, z) \tag{2}$$

$$\frac{d}{dx} ReLU(x) = \{1 \; if \; x > 0; 0 \; if \; otherwise\} \tag{3}$$

Within this layer, the rectified linear activation functions are applied to the output of the previous layer (functions shown in equations 2 & 3 above), as it improves training performance by making the data more non-linear. In the past, the sigmoid and tan hyperbolic (tanh) functions were used, however there was a problem where the gradient signal began to vanish with more layers of the network, commonly known as the "vanishing gradient". [32] ReLU has "sparser representation", meaning that true zero values can be better obtained, in comparison to tanh and sigmoid. ReLU also has much more linear behaviour as opposed to sigmoid and tanh, as can be seen in Figure 9 with the gradient of the curve after 0. The "activated" feature of the object is carried into the next layer.

### 3.1.2.4 Pooling

The output from the ReLU is simplified in the Pooling layer via down-sampling, which reduces the resolution of the output so the network has less parameters to work with. This helps the CNN for the

latter stages of this process in future layers. This process is called “Max Pooling”, where a “depth slice” (as described in the Input Layer) is taken and split into regions, depending on the “stride” value set in the CNN. The network strides from block to block to take the largest value and store it in a depth slice of reduced size, containing the number of regions from the original input slice. [32]

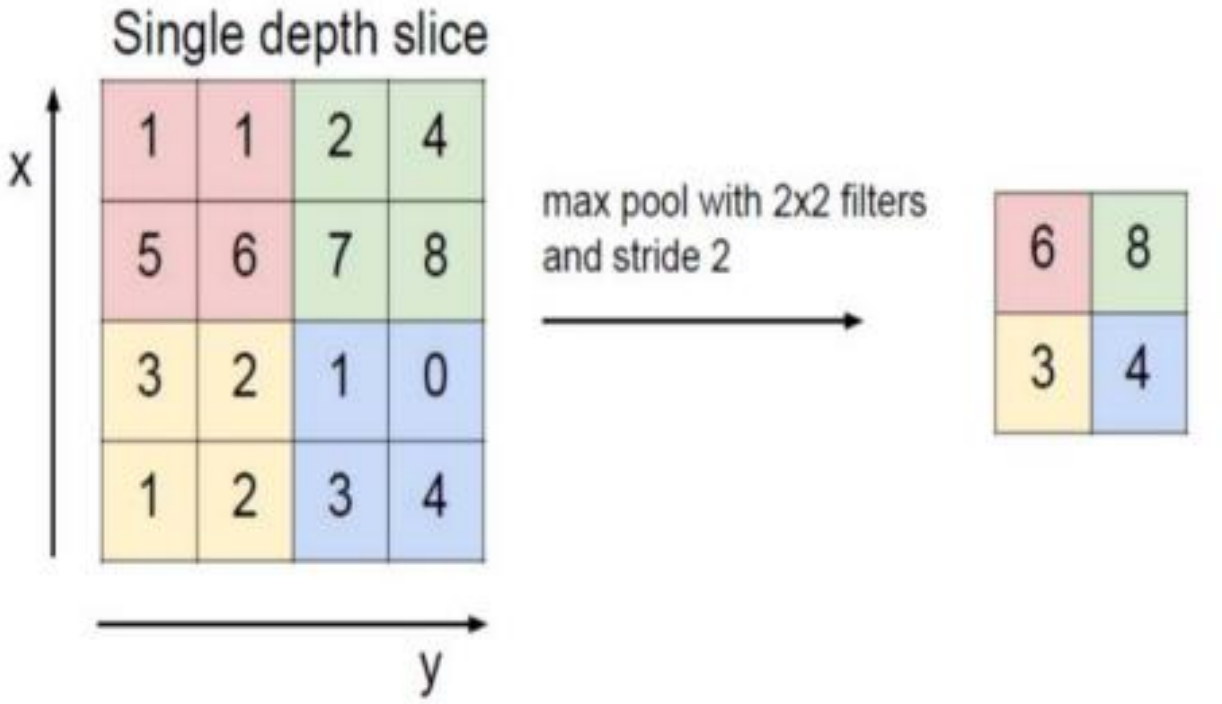


*Figure 10 - Max Pooling Representation [32]*

### 3.1.2.5 Fully-connected Layers

The neurons after these processes would have been trained to identify a particular feature about the object that the network had been trained to detect. After these processes, the CNN has the Classification Layers, consisting of the fully connected layer & the final classification layer, where the classification output of the object is provided. [33] This is achieved by connecting every node from the previous and next layer together.

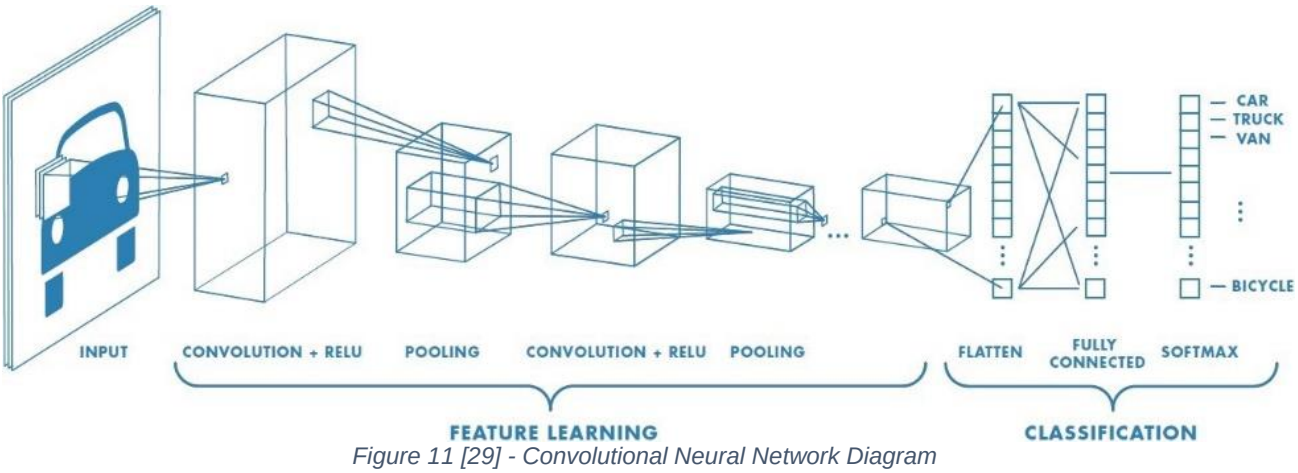


*Figure 11 [29] - Convolutional Neural Network Diagram*

### 3.1.2.6 Summary of overall detection process

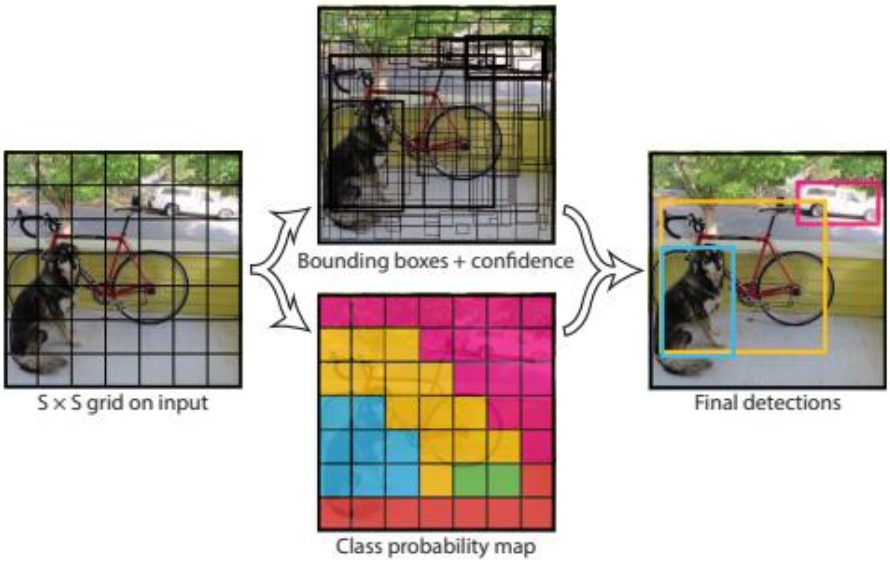


*Figure 12 - Detection Process [34]*

The input image is split into S x S grids. Each region is responsible for predicting what class is within that region and providing a confidence score. The grid cell where the centre of the object falls within is responsible for predicting the class and providing a confidence score from it, which is defined by equation 4 [34]:

$$\text{Confidence} = \Pr(\text{Object}) * IOU_{pred}^{truth} \quad (4)$$

Where Pr(Object) is the probability of a particular object in that cell and IOU is the Intersection Over Union between the predicted box and the ground truth box (if any). If there is no object in that cell, the confidence score is zero. [34]
At test time, class specific confidence scores and generated using equation 5 [34] in each grid cell:

$$\text{Class Specific Confidence: } \Pr(Class_i|\text{Object}) * \Pr(\text{Object}) * IOU_{pred}^{truth} = \Pr(Class_i) * IOU_{pred}^{truth} \quad (5)$$

## 3.2 YOLO

### 3.2.1 What is YOLO?

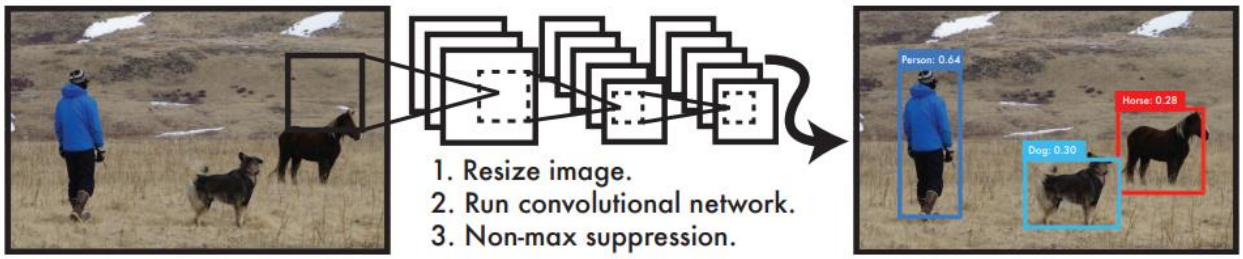


*Figure 13 – The YOLO System [34]*

The convolutional neural network that will play an integral part throughout this project is the "You Only Look Once" (YOLO) algorithm. YOLO is an open source CNN that utilises a single convolutional network, achieving object detection by processing the image in its entirety in one evaluation by taking the input image as a single regression problem, predicting where the bounding boxes would be as well as the probability of the class that the bounding box is encapsulating simultaneously during the training and test time. This makes it significantly faster than its competitors in object detection, such

as R-CNN (Regional CNN) or the Deformable Parts Model (DPM), where their classification methods include an extra stage for their method of localisation of the object before predicting the object class. [34] The benefit of incredible speed however is held back by its performance with localisation of the objects in the image as a result. Another benefit to YOLO processing the entire image is that it takes note of the contextual information around the object, so when these objects are to be detected in another environment, it performs better than its competitors. [34]

A simple explanation to how YOLO works is shown in Figure 13. Depending on the set parameters, the input image dimensions are scaled by a multiple of 32 pixels (i.e. 416x416) before running its convolutional network on it. Non max suppression is then applied to the image to select the best bounding box to encapsulate the object. An example of this is depicted in Figure 14 below.

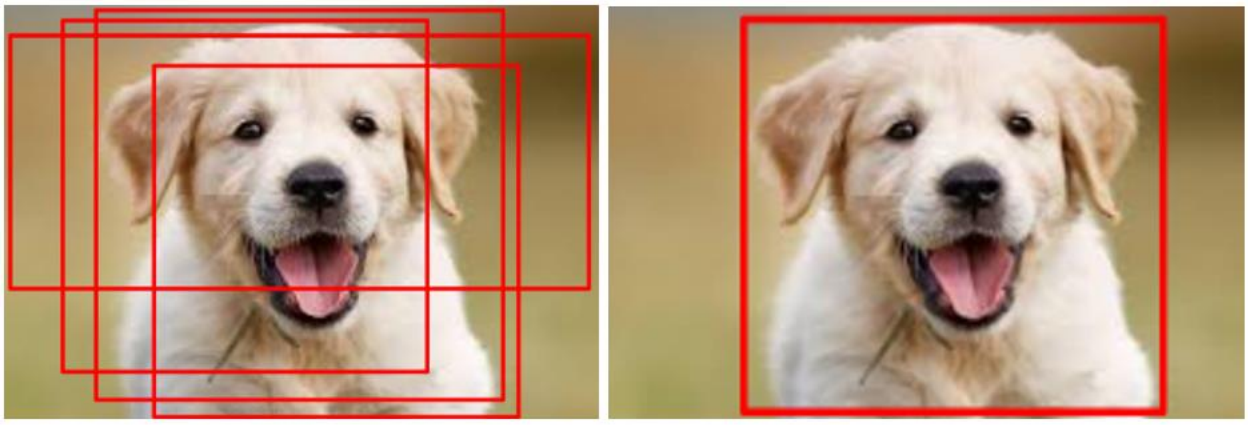

*Figure 14 - Before & After Non Max Suppression [35]*

### 3.2.2 Object Detectors

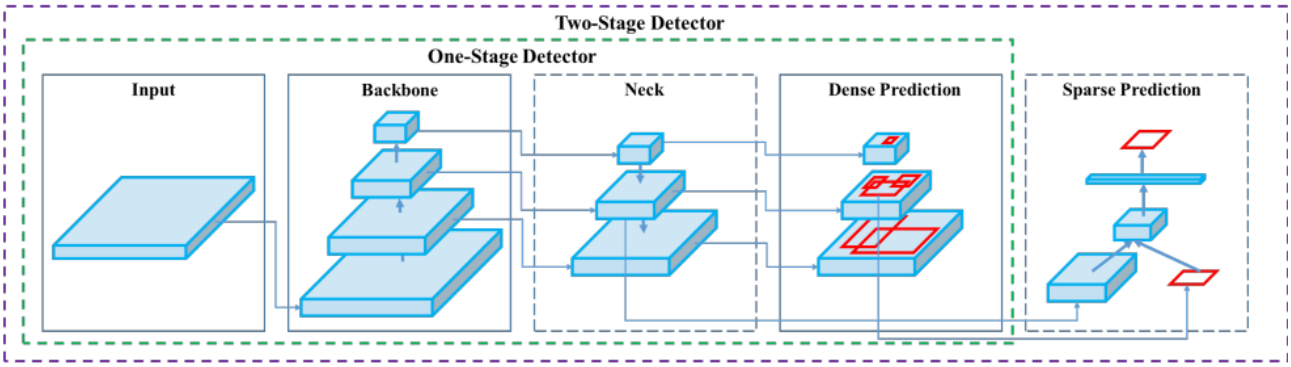


*Figure 15 - Generic Object Detector Overview [36]*

The general object detector consists of the following features:

- Input
- Backbone
- Neck
- Head

The backbone is pre-trained to identify desired objects/classes. The best object detectors are trained on ImageNet, which is a large dataset consisting of more than 14 million images of more than 20,000 classes. [37] The “Neck” stage consists of the layers to collect feature maps generated from prior stages. The YOLO algorithm consists of 24 convolutional layers as well as 2 fully connected layers, as depicted in Figure 16. As discussed in the section 3.1.2, convolutional neural networks generally have two stages for object detection, where the “Head” stage of the object detector consists of two parts for prediction (Dense & Sparse prediction as shown in Figure 15). Where YOLO differs from

the usual CNN is that it only consists of one head as opposed to two. [36] The Head stage is what carries out the prediction of classes and bounding boxes.

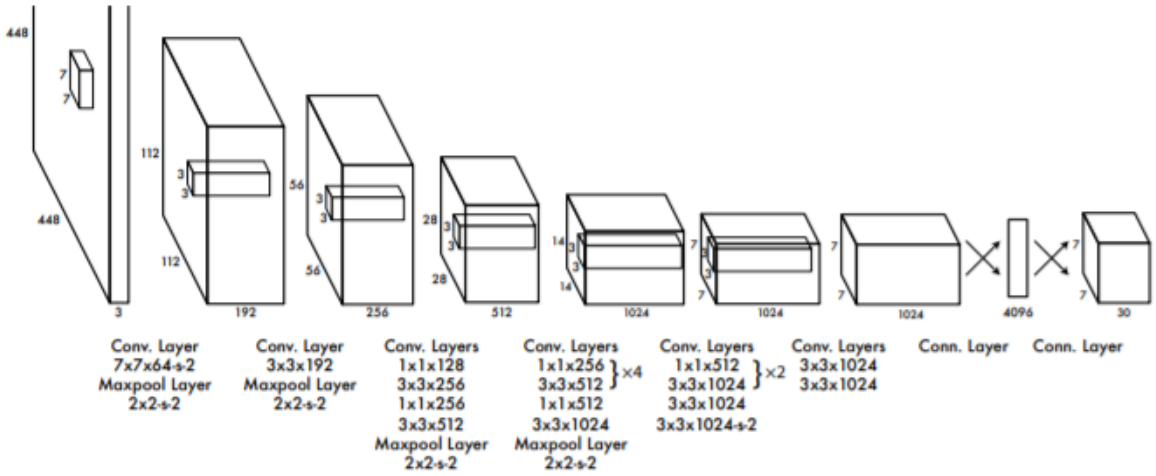


*Figure 16 [34] – Architecture of YOLO's convolutional network*

### 3.2.2.1 YOLOv4

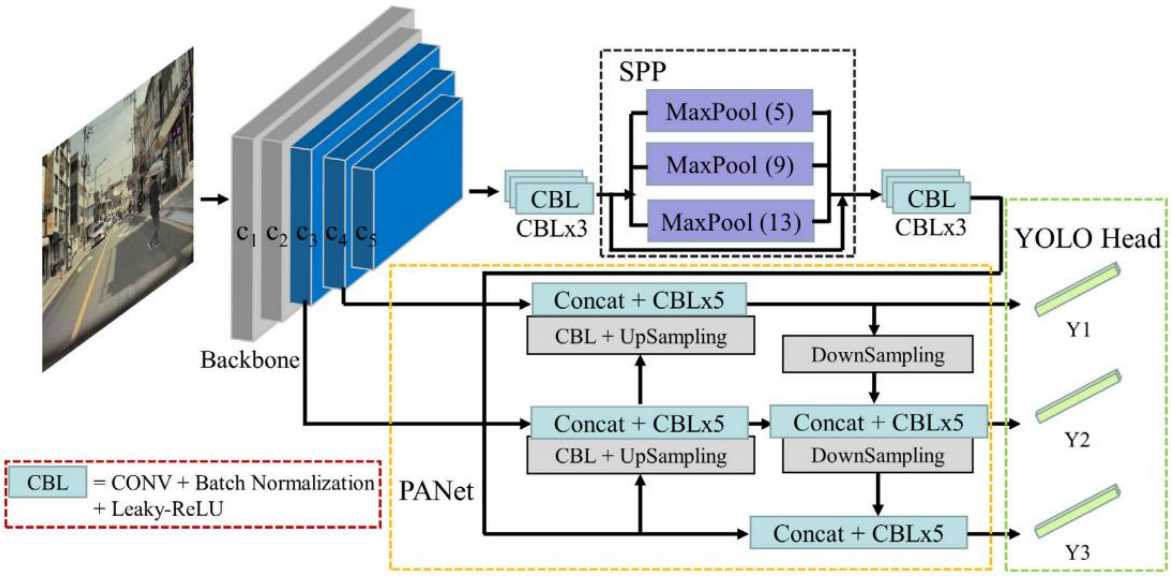


*Figure 17 - YOLOv4 [38]*

The YOLO system is structured as the following: [36]

- Backbone: CSPDarknet53
- Neck: Spatial Pyramid Pooling (SPP), Path Aggregation Network (PAN)
- Head: YOLOv3 (x3)

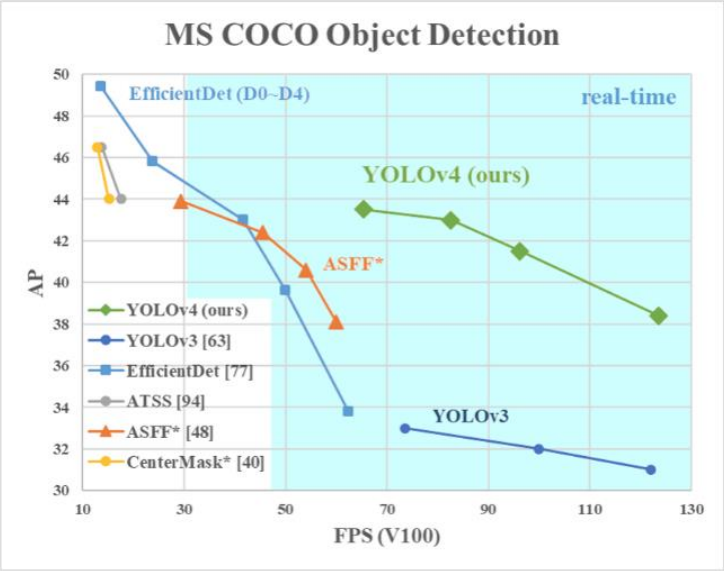


*Figure 18 - Detection Accuracy of Object Detectors [36]*

YOLOv4 utilises features such as Mish Activation to achieve results comparable to state-of-the-art implementations of object detectors. When tested on the Microsoft Common Objects in Context Dataset (MS COCO), as shown in Figure 18, YOLOv4 detects at a faster frame rate, whilst having a detection accuracy (AP) of approximately 44%, whilst the industry standard EfficientDet detects at a much slower frame rate whilst achieving a detection accuracy of approximately 50%.

#### 3.2.2.2 YOLOv4-tiny

YOLOv4-tiny essentially is a toned-down version of YOLOv4, which prioritises inference and detection speed whilst sacrificing accuracy. It is designed to be used on more mobile devices where the required specification to be able to use it isn't as heavy as it is it for YOLOv4.

The YOLOv4-tiny system consists of the following: [39] [40]

- Backbone: CSPDarknet53-tiny
- Neck: Feature Pyramid Network (FPN)
- Head: YOLOv3 (x2)

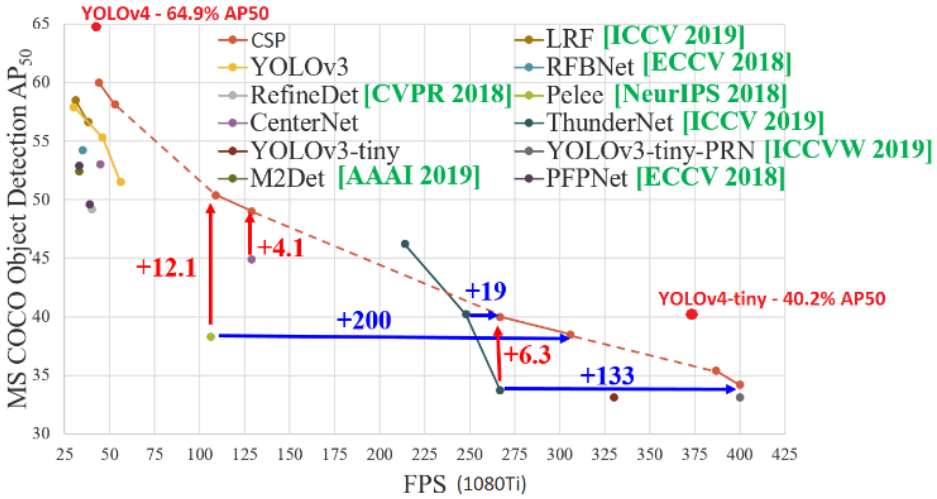


*Figure 19 - YOLOv4 vs YOLOv4-tiny [41]*

YOLOv4-tiny trained on the same MS COCO dataset produces a detection accuracy of about 22%, a significant drop off compared to YOLOv4. However, where YOLOv4-tiny shines is the FPS at which it detects objects, achieving an FPS of approximately 400. [41] A like for like comparison is also shown in Figure 19 based on AP50, where YOLOv4 performs at 64% on the dataset whilst YOLOv4-tiny performs at 40.2% accuracy.

# 4 Setting up Development Environments

In this section, we will cover the software packages that are to be installed for the completion of this project, as well as the installation procedures and methods of verifying that the software has been installed correctly.

| Software Package to Install | Version Number | Vendor | Reason |
|---|---|---|---|
| Microsoft Visual Studio Community | 2019 | Microsoft | Used to build OpenCV with CUDA capabilities |
| MinGW | GCC: 8.1.0<br>MinGW: 6.0.0 | SourceForge | Provides the required C compiler when compiling programs in command line |
| Git | 2.35 | Git | Helps to manage source code |
| Powershell | 5.1.22 | Microsoft | Command Line application |
| CUDA Computing Toolkit | 11.6 | NVIDIA | Features libraries to be utilised by the GPU |
| cuDNN | 8.3.2 | NVIDIA | Deep Neural Network library to be utilised by the GPU |
| CMake | 3.22.2 | CMake | Compile projects with desired CUDA capabilities |
| Anaconda | 4.1.2 (for Python 3.9) | Anaconda | Create different python environments |
| OpenCV | 4.5.2 | OpenCV | OpenCV library with base modules |
| OpenCV-contrib | 4.5.2 | OpenCV | OpenCV library with additional modules |
| LabelImg | 1.8.1 | Tzutalin (GitHub) | YOLO file annotation |
| OIDv4-Toolkit | V4 | theAIGuysCode (GitHub) | Download and change annotation format from Google's Open Source Image Dataset |

*Table 1 - Software Packages to be installed for the project*

Instructions to install these programs in Table 1 can be found on their websites. Ensure that the Path variable in the system variables are updated, directing the Path variable to the binary folders of these programs, and restart the device once this process is complete.

## 4.1 Darknet on Windows 11

To compile Darknet on Windows 11, the programs mentioned above were installed. Next, we will install darknet using Microsofts "vcpkg" which comes with OpenCV & CUDA capabilities during the installation process. Using Powershell, enter the following commands:

```
Set-ExecutionPolicy unrestricted -Scope CurrentUser -Force
git clone https://github.com/AlexeyAB/darknet

cd darknet
```

```
./build.ps1 -UseVCPKG -EnableOPENCV -EnableCUDA –EnableCUDNN -EnableOPENCV_CUDA
```

Once these commands are executed and the packages are installed, the next step is to download YOLOv4 weights based on the MS COCO (Microsoft Common Objects in Context) dataset, which can be found on the Darknet Github page (the same website which was cloned in the previous step). We will use these weights to determine whether we can run object detection based on preset images provided during the Darknet installation process.
Copy the yolov4.weights file to the Darknet directory. To run a test object detection on one of the preset images, enter this command in Powershell:

darknet.exe detect cfg/yolov4.cfg yolov4.weights data/dog.jpg

This should produce the preset dog image with bounding boxes around each object in the image, along with the confidence the algorithm has of the object it has detected, as shown in Figure 20, indicating that the algorithm is working on our device.

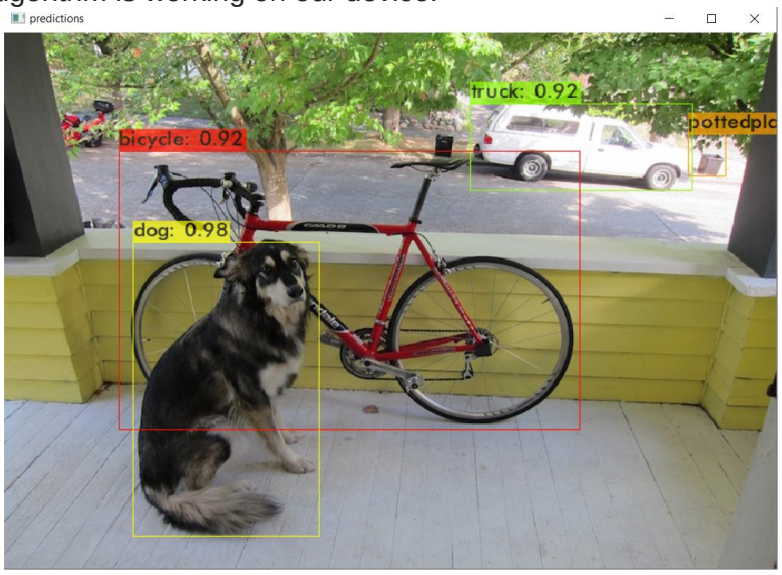


*Figure 20 - Preset Dog Image after Object Detection*

## 4.2 OpenCV with CUDA

This process will allow us to utilise OpenCV's DNN module with GPU capability. This will give us the best possible frames per second (FPS) for video playback during test time. Before installation, ensure the correct CUDA architecture version of your system is noted. This project will be conducted using an Nvidia GeForce RTX 3050Ti GPU 16GB RAM (4GB VRAM) with a CUDA Compute Capability of 8.6. This information can be found on NVIDIA's website. [42]

Using an anaconda prompt, the latest version of numpy is installed using the following command: "`–upgrade numpy`".
In the darknet directory, a folder dedicated to OpenCV was made to extract the download files to. A build folder was also made within this directory. Then using the CMake GUI, the source code is set to the folder renamed after the OpenCV version in the OpenCV directory (i.e. opencv-4.5.2), and the libraries are built in the newly created build folder. The project is then configured using Visual Studio 16 with the x64 platform for generation. The "PYTHON3" section is then filled out by highlighting the paths to the library and include folders, which can be found in the "C:/Users/..../anaconda3" folder. The project is then configured again before enabling/setting the following options:

- `WITH_CUDA`
- `BUILD_opencv_dnn`
- `OPENCV_DNN_CUDA`
- `ENABLE_FAST_MATH`

- BUILD_opencv_world
- OPENCV_EXTRA_MODULES_PATH: “C:/Users/.../opencv_contrib-4.5.2/modules

The project is then configured once more so the following options are available to enable:

- CUDA_FAST_MATH
- CUDA_ARCH_BIN: Set to your machines CUDA architecture version
- CMAKE_CONFIGURATION_TYPES: Release

The project is then generated and ready to be built using the opencv build folder by entering the following command in anaconda prompt:

```
cmake --build "C:\*path to build folder*\build" --target INSTALL --config Release
```

# 5 Creating a Weighted Model for Traffic Sign Detection

Throughout this section we will investigate a popular open source dataset regarding the number of images and instances of objects it has to provide a good idea of what it takes to produce a good dataset of our own. The dataset created to train an object detection model is just as important as the convolutional neural network that you train it on, as flaws in the model stem from the volume and variety of the images you train the neural network with. From the model created in this section as well as the open source dataset discussed, we critically analyse the performance and investigate the limitations in performance of the models for the YOLOv4 and YOLOv4-tiny architectures.

## 5.1 Existing Datasets and Models

One of the reasons for creating a dataset is so that it can be trained on a neural network to identify desired classes (objects). A good industry standard dataset consists of hundreds of thousands to millions of images with several instances of the desired classes that are to be trained for detection. A popular dataset which we will be comparing our dataset and model to is the Microsoft Common Objects in Context (MS COCO) dataset.

### 5.1.1 Microsoft COCO (MS COCO) Dataset

The Microsoft Common Objects in Context (MS COCO) dataset exists to detect the most common objects in our day to day lives, with the dataset providing labels for 91 different classes. A total of 328k images are within this dataset, from which over 2.5 million instances of the classes are labelled from these images. [43] According to this paper [43] on the MS COCO dataset, the dataset was created with the intention of trying to ensuring that any neural network using the dataset is not subjected to images only trained on an “iconic view” of the object, “iconic view” meaning that the object is placed in the very centre of the image with no obstructions or background context provided. This is detrimental for a neural network as we know in realistic scenarios, more often than not the object would be subject to occlusion and various lighting conditions. A brief technical overview of the COCO dataset is detailed below, which can be comparable to the dataset to be created further along this report:

- At least 82 of these classes have at least 5000 labelled instances, [43] thus we can expect a much weaker performance from the remaining 9 classes should the dataset be trained on a neural network.
- MS COCO dataset has an average of 7.7 object instances per image. This is significantly more than other commonly used datasets such as ImageNet (3 instances per image) and PASCAL VOC (2.3 instances per image). [43]

#### MS COCO YOLO Model

Following the installation steps as described in section 4.1 above, we have access to a model that created based on this dataset trained on YOLO’s neural network. A total of 80 classes were taken from the original dataset into this model, including objects such as vehicles, traffic lights and pedestrians. Models based on YOLOv4 and YOLOv4-tiny are also provided and we can observe clear differences in accuracy of detection and inference between both frameworks, as illustrated in section 5.1.2.

### 5.1.2 YOLOv4 vs YOLOv4-tiny

Despite being trained on the same dataset, we can see the clear increase in object detection and confidence scores produced by YOLOv4 compared to YOLOv4-tiny shown in Figure 21 below. This would be due to the extra convolutional layer (YOLO-head) in YOLOv4 providing the capability to detect objects even more thoroughly. We can see an improvement in object inference which appear smaller and further away in the frame, which YOLOv4 is said to struggle with typically due to processing each frame as a single regression problem, however the strength of the COCO dataset evidently aids the performance of YOLO in these scenarios.


*Figure 21 - MS COCO YOLOv4 vs YOLOv4-tiny*

Despite these improvements, we can also see in Figure 21 that YOLOv4 still struggles to detect the traffic light in darker conditions. This could be a result of the lack of colour disparity between the traffic light and the night sky, as it would make it difficult to pick up the edges of the traffic light in this scenario. It could also be a result of the frame in question being of a lower resolution than what would be considered high definition. One would assume that a lower resolution would negatively affect the neural networks ability to detect objects; as what the network would define as a "distinct" feature of an object such as the edge of a traffic light, could be lost as more information would have to be represented across less pixels in a frame. This loss of information would not be accounted for in training unless the dataset consisted of blurry, low resolution pictures. Python scripts were developed to apply filters to these images so we can see if there is a change in performance if we affect the noise levels within the images as well as if we alter the colour information of the pictures via contrast.

#### 5.1.2.1 CLAHE Filter

Contrast Limited Adaptive Histogram Equalization (CLAHE) is a form of Adaptive Histogram Equalization (AHE), a filter that is normally used to improve the contrast of images. However it is well known that AHE typically over-amplifies the noise when applying its contrast amplification to the image. [44] CLAHE counters this by operating in small regions of the image (referred to as Tiles), and then combining neighbouring tiles together via bilinear interpolation the remove the artificial boundaries between the tiles, whilst also ensuring that the contrast amplification is limited during this process. [44] This produces an image where pixels within the "tiles" appear slightly more pronounced, making the image look a bit more pixelated. This is great for images where the quality is not very clear; however if applied to an already high quality image, the tiles utilised in the filter can be visible, and it can ruin the quality of the original image.


*Figure 22 - YOLOv4 vs YOLOv4-tiny before and after application of CLAHE Filter*

As shown in Figure 22 above, the application of CLAHE has made the image look a lot pixelated, seemingly affecting the confidence scores of the objects detected compared to the confidence before the filter was applied. However there are more instances of traffic lights being detected in YOLOv4, along with one false detection of a traffic light (detection furthest to the right). This indicates that potentially for YOLOv4, CLAHE can encourage better inferencing of objects, as somehow there is more information being presented to the network for it to detect the presence of an object. This is not the case for YOLOv4-tiny however, as the only object to be detected before the filter has now disappeared. Considering YOLOv4-tiny is a weaker network than YOLOv4 just from the overall structure of the network, these results indicate that it may be safer to keep pixels within an image as clear as possible for YOLOv4-tiny, or otherwise train YOLOv4-tiny on much lower resolution images to aid its detection of objects in these scenarios.

#### 5.1.2.2 Contrast

As shown in Figure 23 below, after decreasing the contrast of our image, we can see for YOLOv4 that the traffic light further down the road in the image was picked up, however a traffic sign towards the front of the image was picked up as a truck. In addition to this, the confidence scores did not really change significantly, although for the car closest to the source of the frame decreased by a confidence score of 0.09.

YOLOv4-tiny in comparison is expected to perform much worse, and it does. Only the car at the start of the frame is detected with a confidence of only 0.5. Once the filter is applied, the algorithm detects the car as a train with a reduced confidence of 0.28.

This may suggest that the confidence YOLOv4 has in detecting objects with a decreased contrast is reduced, considering the algorithm actually performs worse on what it was more confident on prior

to the application of the filter. As shown in YOLOv4, the ability to inference objects may increase, however it does not necessarily mean that the accuracy of the inference is good.

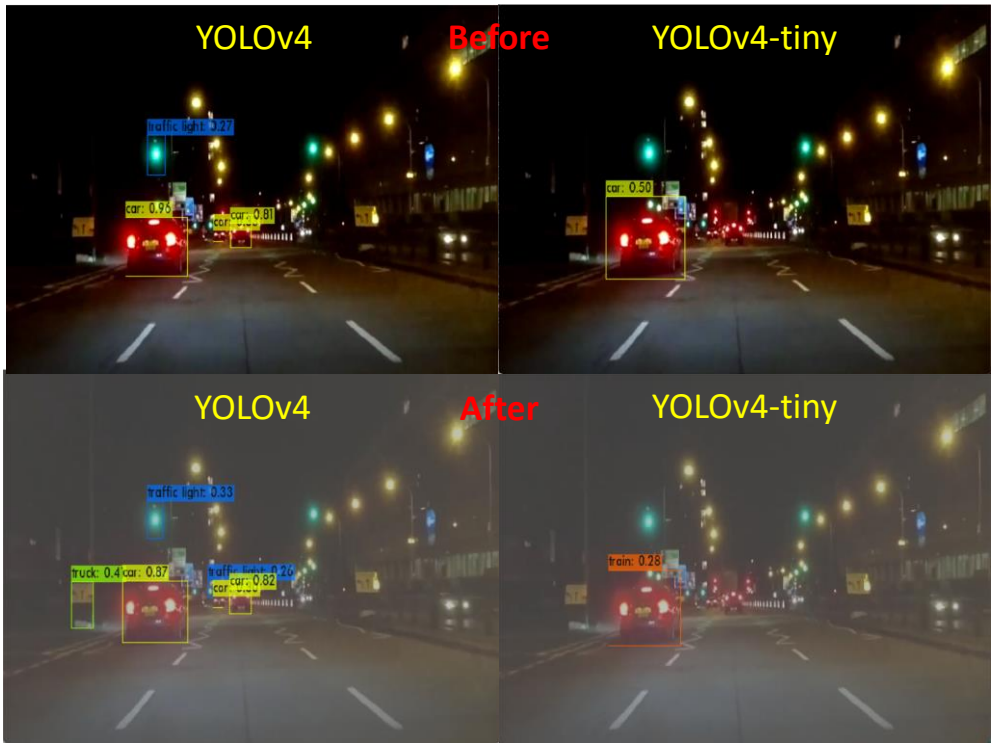


*Figure 23 - YOLOv4 vs YOLOv4-tiny before and after application of Contrast change*

## 5.2 Creating a Speed and Traffic Signs (SaTS) Dataset

### 5.2.1 Objectives of SaTS

For the completion of this project, the creation of this model is paramount. The purpose of this model is to provide an autonomous vehicle with instructions dependent on the object detected and extra information presented by the detected object. Without this model, the project would not have been able to progress, as there would be no objects to detect.
The following objects to be detected are shown in Table 2 below:

| Class Name | Example | Class Name | Example |
|---|---|---|---|
| pedestrian |  | speed sign 60mph |  |
| school crossing |  | speed sign 70mph |  |
| speed sign 20mph |  | speed sign 80mph |  |
| speed sign 30mph |  | stop sign |  |
| speed sign 40mph |  | traffic light |  |
| speed sign 50mph |  | vehicle |  |

*Table 2 - Classes of SaTS Dataset*

With these objects, we are able to demonstrate certain decisions that must be made depending on these hazards, such as stopping at a red traffic light or changing the speed once a speed limit sign is presented.

## 5.3 Installation pre-requisites

- OIDv4 Toolkit

Can be installed simply by entering the following command into Powershell:

```
git clone https://github.com/EscVM/OIDv4_ToolKit.git
```

Then enter the following command to ensure all the requirements are installed:

```
pip install -r requirements.txt
```

## 5.4 Methods of Obtaining Images

### 5.4.1 Google Open Image Dataset (OID)

Google have their own open source website full of labelled image classes. However the annotations do not come in YOLO format, so using the script that was provided with the OIDv4 Toolkit, we will convert these annotations into the YOLO format. To download the classes available (maximum 2000 images), the following command is entered into powershell:

```
python main.py downloader --classes (class_names) --type_csv train --limit 2000
```

Navigate to the "OID_Toolkit" folder that was generated upon installation and edit the "classes.txt" file to the names of the classes in our dataset. This will renumber the classes of the newly downloaded classes to correspond with our current dataset. Once the classes are set, run the following command in powershell to convert the annotations to the YOLO format:

```
python convert_annotations.py
```

### 5.4.2 Google Images/Shutterstock

Images can be obtained from Google Images and open source websites such as ShutterStock. Time was saved by mass downloading images by using an "Image Downloader" google chrome extension.

### 5.4.3 Image Augmentation

Image augmentation is a method of generating "fake data" using the images already collected. This involves methods such as (but not limited to) changing the brightness, rotation or saturation of an image. This is useful for a CNN learning the distinct features of an object regardless of the context the image is situated in.

### 5.4.4 Training a Deep Learning Model on YOLO

In comparison to other machine learning methods, Deep Learning requires high performance hardware, such as GPU hardware with a lot of memory. This is because to even compute a version of YOLO, there is a VRAM requirement to train weights. YOLOv4 has a requirement of minimum 8GB of GPU-VRAM, compared to YOLOv4-tiny which can generate weight files on devices with at least 4GB GPU-VRAM thus those with less capable hardware can train their dataset with YOLOv4-tiny weights, but would suffer when it comes to detection accuracy.

To achieve optimal results from training an object detector, it is important to try and replicate real world scenarios with regards to how the desired object would appear in video feedback. Using YOLO makes this a lot easier to achieve due to the fact it takes extra contextual information from looking at the input image in its entirety as opposed to just the highlighted object in question. This can be achieved by taking pictures in different weather conditions and lighting, or ensuring that the object can appear in different sizes or angles as it could potentially appear on video. In addition to this, a

significant amount of data is required to build a model that accurately detects objects, as a detector trained on a small dataset does not perform adequately. [45] The objects we intend to detect (particularly speed limit signs) have extremely limited availability online in comparison to objects like vehicles, pedestrians and traffic lights. Not to mention the majority of what is available come in the form of stock images, which provide no contextual information outside of the content of the sign. To expand the dataset as well as the variety of data within it, we can use data augmentation methods. To implement data augmentation, we will use the “ImgAug” library.

### 5.4.5 ImgAug Library

There are only so many images that can be taken from the internet of each object class, considering you need a scale of hundreds of thousands of images. Images online also do not always represent real world scenarios which these objects can be found on the road in relation to weather conditions, lighting from the sun or in the night etc. Using ImgAug library in python, we can try to replicate the desired effects whilst also expanding our dataset.

To install the imgaug library, the following command must be typed into powershell:

```
pip install imgaug
```

#### 5.4.5.1 Discussion of Effects used

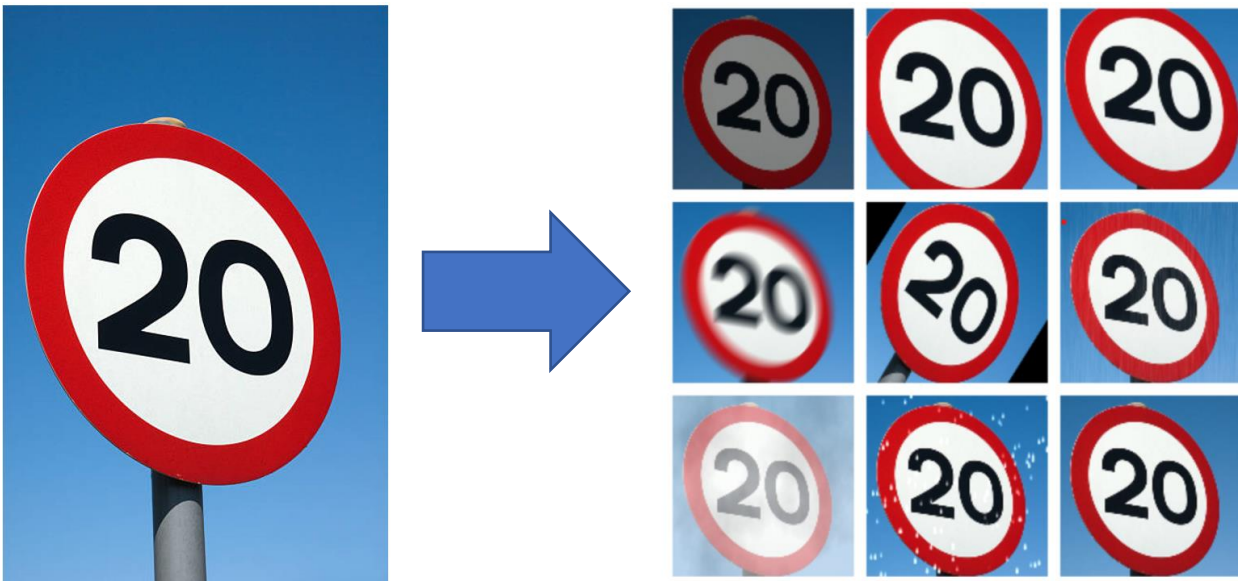

*Figure 24 - Example of Augmentation Methods carried out with ImgAug*

To replicate real world scenarios of speed signs, we considered the conditions which the speed signs would be visualised in video footage. For example, the model may have to detect the speed sign whilst it could be raining or snowing, or the video feed could show the speed sign be subject to a motion blur as the vehicle speeds past the sign. With these scenarios considered, using the “imgaug” library, the following effects added are outlined in Table 3:

| Desired Effect | Function | Outcome |
|---|---|---|
| **Motion Blur** | iaa.MotionBlur() |  |
| **Brightness (brighter & darker)** | iaa.Multiply() |  |
| **Fog** | iaa.Fog() |  |
| **Grayscale** | iaa.Grayscale() |  |
| **Perspective Transform** | iaa.PerspectiveTransform() |  |
| **Rain** | iaa.Rain() |  |
| **Rotate** | iaa.Affine(rotate) |  |
| **Snow** | iaa.Snowflakes() |  |
| **Zoom** | iaa.Affine(scale) |  |

*Table 3 - Added ImgAug Effects*

To begin with, this process was only applied to 10-15 images per class across the 12 classes mentioned in section 5.2.1, giving the initial dataset a size of approximately 1300 images. A dataset this size is considered very small, and not expected to produce the desired results.

### 5.4.6 Setting up Training

Once a dataset has been created, the next step is to ensure that the dataset can be utilised by the convolutional neural network for training. With YOLO, the network operates with supervised learning, meaning we must produce labels for the dataset which indicate what class is being trained and where it is in the image.


*Figure 25 - YOLO File generation*

YOLO files are '.txt' files that correspond to a given '.jpg' image, containing information about the class and where the object is situated within an image. The files are produced with the following format:

```
<object-class> <x_center> <y_center> <width> <height>
```

Where:

- `<object-class>` represents the object class number that is predefined in your dataset (values range from 0 to (Number of Classes – 1)).
- `<x_center> <y_center> <width> <height>` represent values that are relative to the size of the image that has been labelled, as well as the coordinates of the centre of the annotated bounding box.

An example of a generated YOLO file for its corresponding image is shown in Figure 25. A YOLO file must be produced for every image within the dataset. To achieve this, image labelling software was installed (LabelImg). This software allows you to simply draw a bounding box around the object and it will generate the required information for the text file.

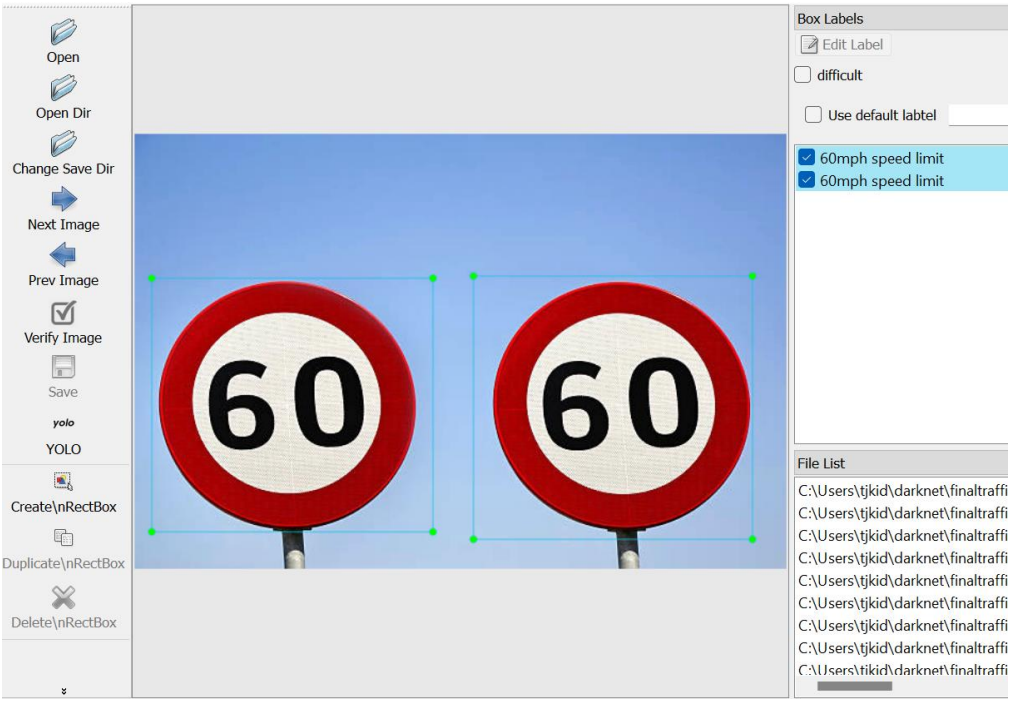


*Figure 26 - LabelImg GUI*

To achieve this, we must set the class names in a text file so the software can provide the option of what class had just been labelled with the bounding box just drawn. To do this, in the installation folder of LabelImg (windows_v1.8.1), exists a "data" folder. In this folder, place a ".txt" file with the desired class names, with each name being on a new line, as shown in Figure 27.

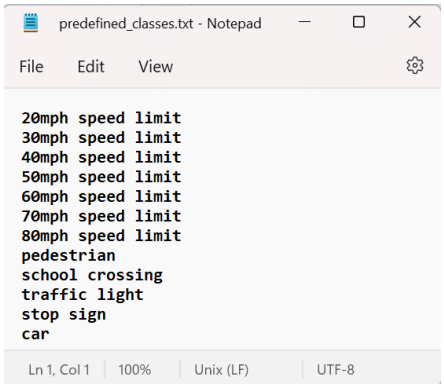


*Figure 27 - Predefined Classes File*

To draw the bounding box, click “Create\nRectBox” (shown in Figure 26) and then highlight the object within the bounding box. Choose the class and click OK. Then click “Save” on the GUI and save to the folder where the dataset images are located. This process was repeated until every image in the dataset was labelled. This will give you a YOLO text file for each image as shown in Figure 25. Once completed, make a copy of the folder and store it in the “data” folder in the darknet directory and rename it as “obj”.


*Figure 28 - Example of allocating a class to an object*

### 5.4.6.1 Configuring YOLO files for training

Within the original darknet folder, copies of “`coco.names`”, “`coco.data`” and “`yolov4-tiny.cfg`” were made, as our own versions of these files will be used for training our dataset. In the “`coco.names`” file should just be the names of the classes to be trained. In “`coco.data`” are the addresses of files that the YOLO algorithm must use to access the training set, as well as where the weights files are to be stored.

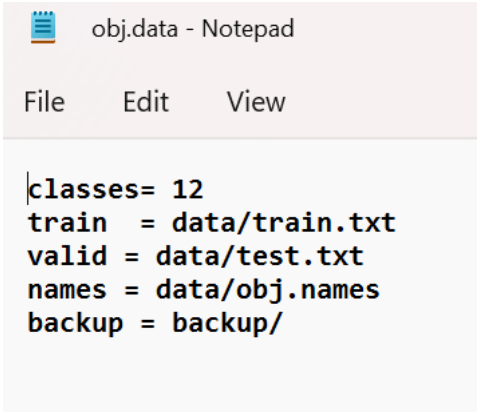


*Figure 29 - obj.data file to set addresses of files YOLO must use for training*

Set the number of classes within our dataset on the first line of the data file. For the training and validation dataset, text files must be generated stating where each “.jpg” file of the dataset can be found and which pictures are to be used for each dataset. A python script was developed to produce these text files.

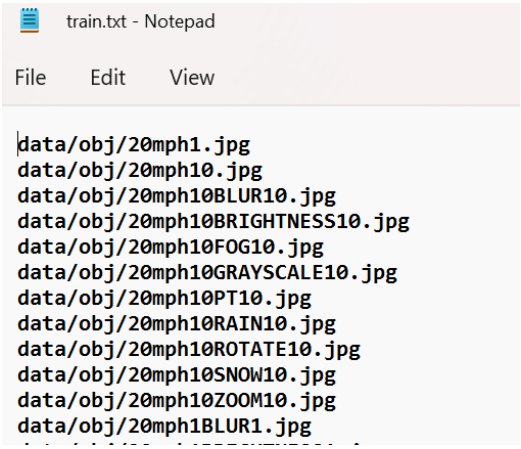


*Figure 30 - Format of training files*

In the "`yolov4-tiny.cfg`" file, there are some settings that must be adjusted for optimal training results that correspond with the dataset.

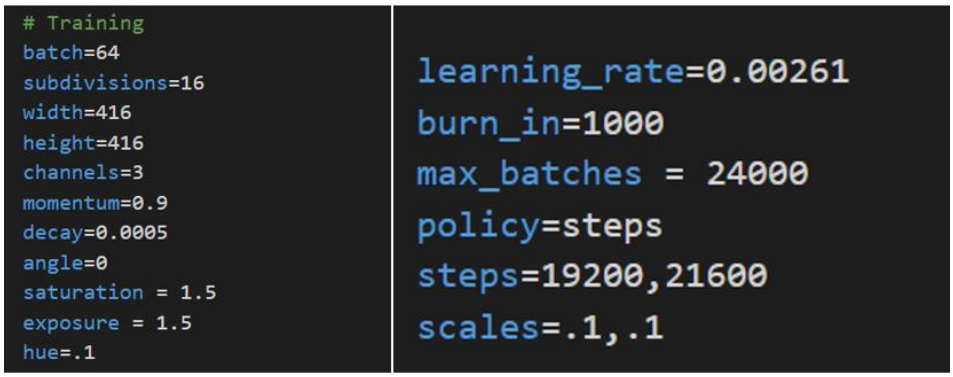


*Figure 31 - Settings to change in YOLO configuration file (1)*

For the first training section, batch must be changed to 64 and Subdivisions must be changed to 16. This determines how many images will be processed in parallel by the training algorithm (batch/subdivisions). Depending on how much VRAM the processing unit has, memory errors may occur. In this case, you may up the subdivisions to 32 or 64. [46] The input image size can be scaled to 416x416, or any resolution as long as the height and width are of equal length and is a multiple of 32. The larger the resolution, the more accurate the model can be as it gives the network the capability to detect smaller objects in the images. However this significantly increases training time, as well as inference time when detecting an object during run time. The channels variable can be left at 3 if your dataset consists of images primarily of a RGB colour scale. If the dataset was full of grayscale images however, the number of channels must be changed to 2.

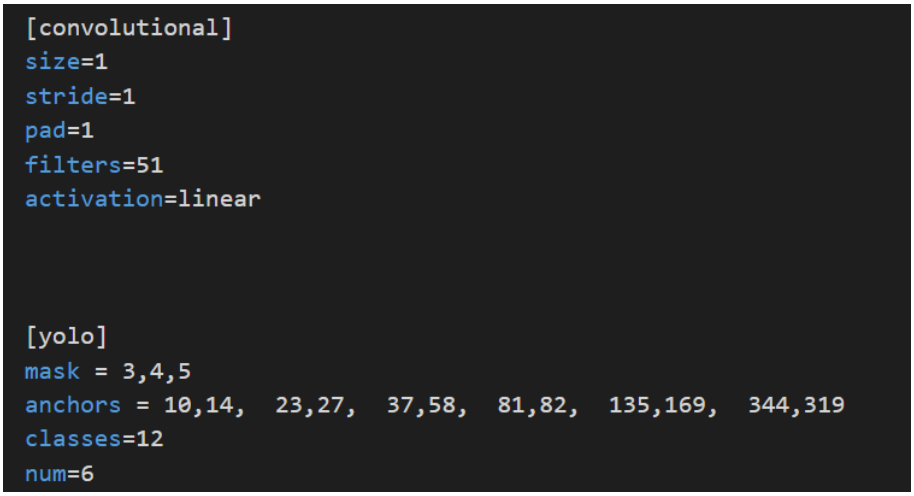


*Figure 32 - Settings to change in YOLO configuration file (2)*

Then search through the file for a section where “[convolutional]” and “[yolo]” layers are defined. In the convolutional layer, the number of filters must be changed satisfy equation 6:

$$\text{filters} = (\text{number of classes} + 5) * 3 \quad (6)$$

This must only be changed in convolutional layers declared right before each “[yolo]” section. In our case, as shown in Figure 32, we have 51 filters. Then in the “[yolo]” section, we change the number of classes to 12 to fit our dataset. Depending on which version of YOLO you intend to train your model with, you may need to complete this 2 or 3 times, as YOLOv4 has 3 YOLO heads whereas YOLOv4-tiny only has 2.

It is said that to train a model using YOLOv4, it is required to have an NVIDIA gaming graphics adapter (GPU) with 8-16GB of VRAM. Due to lack of resources in this regard, we have to train our model with YOLOv4-tiny weights, which will mean that our final model won’t be as accurate, however it will have faster inference speeds when predicting bounding boxes around objects and can run videos at a higher FPS to conduct object detection. To run training, open a command prompt, navigate to the darknet directory and run the following command:

```
darknet.exe detector train data/obj.data cfg/yolov4-tiny-obj.cfg yolov4-tiny.conv.29
```

### 5.4.7 Training Results

#### 1st Model

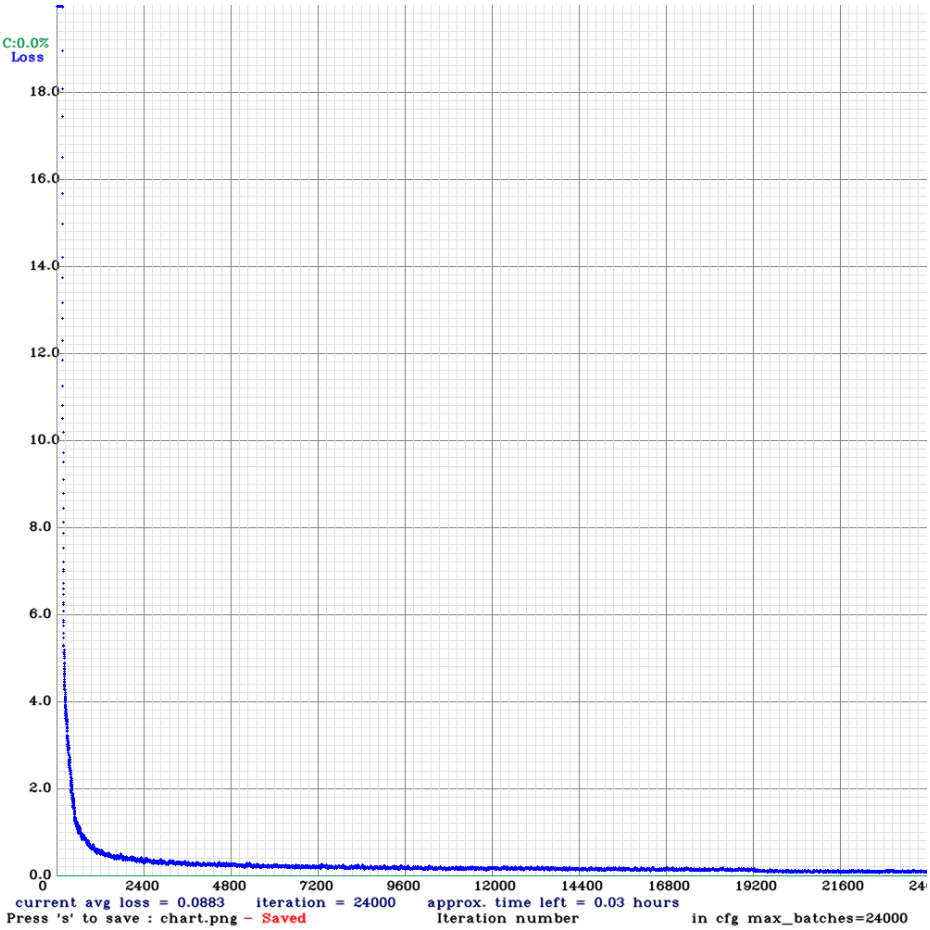


*Figure 33 - Loss Chart from YOLO Training*

The first model to be trained consisted of 1662 images across 12 classes. This did not include images taken from Google's open source image dataset, just images saved from Google Images and augmented using the methods described in section 5.4.5.1. In some images, there were as many as 20 instances of an object that was labelled. The spread of annotated objects across all the images are shown in Figure 34 below.

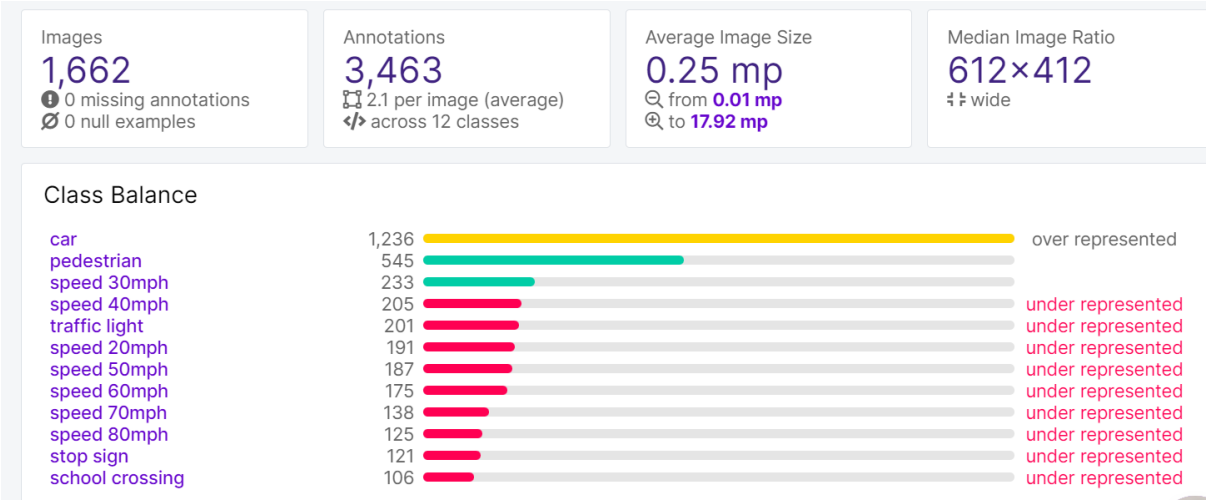


*Figure 34 - 1st Dataset Class Overview*

The larger the model, the more difficult it is for the YOLO CNN to train on it with great precision, however this model is very small and thus, a great mean average precision (mAP@50) can be expected, although this is not a true representation of how accurate the model actually is. This is due to the fact that no images were used for a validation dataset. A validation dataset is not a necessity but it is more ideal if you would like a better idea of how the model truly performs against objects situated in a different context compared to what the model had been trained on. Typically the ideal validation dataset would be significant percentage of the size of the entire dataset. For example, a dataset of 1000 images would typically be split so that 70-80% of the images would be used for training and the remaining 20-30% would be used for validation. However, for a small model such as this one, a validation set did not seem necessary.

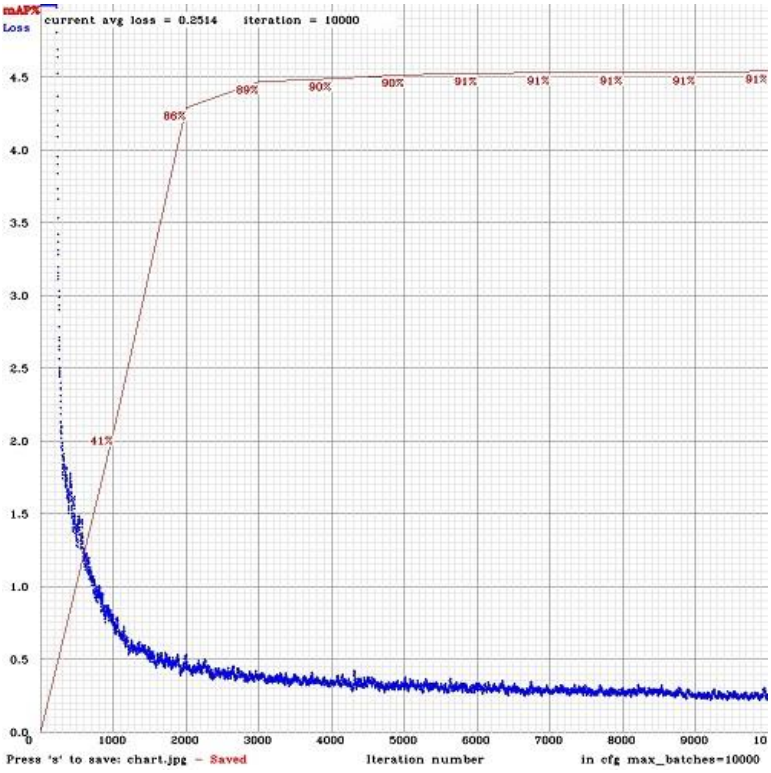


*Figure 35 - Loss Chart with mAP@50 plotted*

Once training was completed, the following command was run on the weights files:

```
darknet.exe detector map data/obj.data cfg/yolov4-tiny-obj.cfg backup/yolov4-tiny-obj_last.weights
```

Where the name will vary according to the names of the weights files generated. This stage is important as the weights named “last.weights” does not necessarily mean that is the model with the best mean average precision. This is because during training, there becomes a point where training becomes excessive, and it becomes too well trained on objects within the dataset as opposed to the same objects in a different context. Figure 35 above shows an example where the mean average precision (mAP@50) is plotted (the red trace), and as the number of iterations increases the mAP@50 comes to a plateau, but does not drop below its maximum value, suggesting that the training dataset was not overfitted. Due to memory errors, it was not possible to plot a chart with the mAP trace, which is why the next best option is to run the command detailed above on each weight file. For the 1st model, the best weights file was generated after 20000 iterations, where we achieved a mAP of 98.3%, as shown in Figure 36.

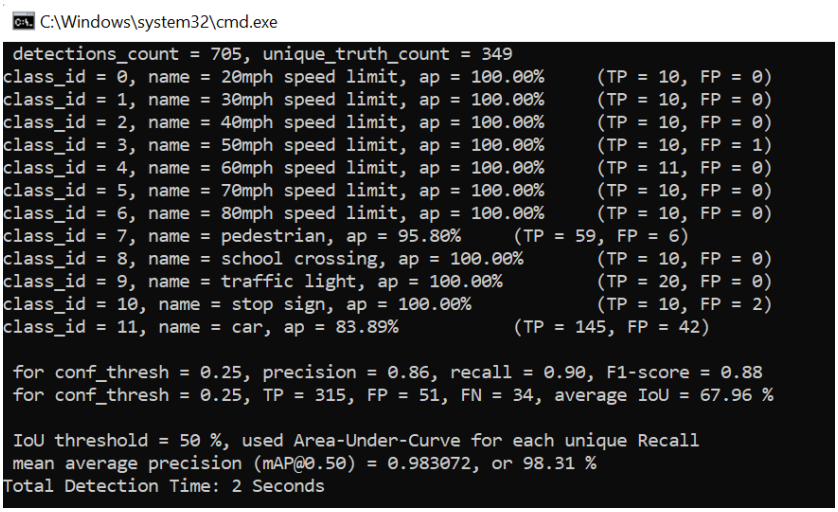


*Figure 36 - mAP@50 results for 1st Model*

When running tests on images already within the training dataset, it detected the object with great accuracy, as shown in Figure 37, however when tested on images not part of the dataset and in videos, there was not much detection at all, besides for vehicles, which was often detected with great accuracy. This would be a result of there being significantly more instances of cars compared to the rest of the classes in the dataset.


*Figure 37 - Prediction on sample traffic light from 1st model*

## 2nd Model

The main area for improvement from the previous attempt is by significantly increasing the volume of images in the dataset, as well as levelling out the distribution of classes. As shown in Figure 38, every class is under represented in comparison to the vehicle class. This just means as a result of training, the model may be significantly better at detecting vehicles compared to the other classes, however there is no evidence to suggest that the representation of the classes must be evenly distributed to achieve optimal results. The best results can be achieved purely by feeding the algorithm with a huge volume of object instances. An example of this is the COCO dataset originally provided in the darknet folder, where the preset YOLO weights file provided is trained on 330,000 images to represent 80 different classes, however there is said to be more than 2.5 million instances of these objects [43] (though the spread is unknown). In an attempt to increase the size of our dataset, we added some open source pre labelled classes from Googles Open Source Image Dataset. The classes that we were able to add from Google were vehicles, pedestrians, traffic lights and stop signs. The final makeup of the dataset is shown in Figure 38 below:

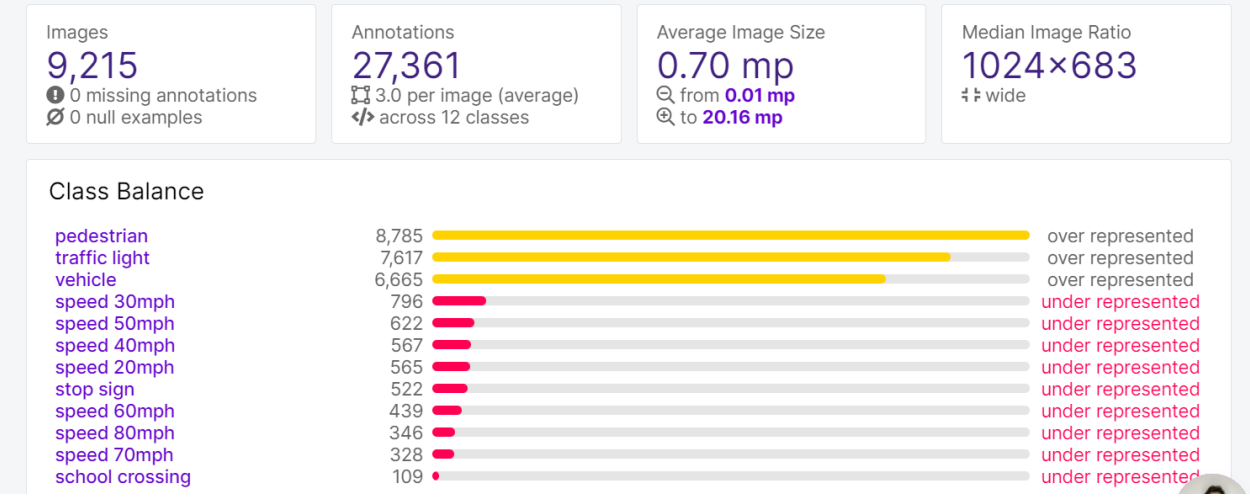


*Figure 38 - 2nd Dataset Class Overview*

Training was conducted on this dataset, just to see if there was an improvement in the detection of traffic lights in particular. The validation dataset used for this training were all of the images used in the 1st model. Entering the same command for the different weight files produced, we achieved a mAP@50 of 75.7%; a significant decrease on the 1st model, however there was an improvement on inference for traffic lights. For vehicles, the overall performance for inference and confidence of classification seemed to drop. In the case of traffic lights, there were a lot of incorrect classifications, which is an improvement on no inferences at all. In the case for vehicles, the inference dropped off quite significantly, as well as the confidence score. The confidence scores dropped from 70-90% to about 40-60% in general over the course of the video. This may be a result of the network being aware that the possibility of an object being a traffic light and not just a car is significantly higher, and in some cases, there are cars that may look like traffic lights at a fundamental level (i.e. a black car with red headlights being similar to a black traffic light showing a red light). There were also a lot of scenarios where there were a lot of “false positives” being picked up. As shown in Figure 39 below, it has drawn a bounding box for a traffic light where there is no traffic light present. In addition to this, it has drawn bounding boxes for the white vans but it is not the most accurate placement to depict where exactly the vehicle is in this specific frame. It has also neglected the black vehicle completely in this frame. This could potentially be a sign indicating that the model is getting better at recognising certain features of the vans, such as the rear headlights, but more work needs to be done in increasing the diversity of the pictures of vehicles that were included in the dataset. Another potential factor for these results could be due to the quality of the video presented. The video takes place on a day with grey skies, which may not be well represented in the images of the dataset. A potential reason for a traffic light being detected when there isn’t one must be related to how the algorithm identifies distinct features of a traffic light. A traffic light typically would be represented as a dark

rectangular shape, which is what the algorithm seems to have picked up in this frame within the bounding box. With more instances of labelled traffic lights, errors like these could be reduced.


*Figure 39 - 2nd Model Testing*

### 3rd Model (SpeedSigns)

For the 3rd model, it was important to begin increasing the representation of the speed limit signs. Websites such as Shutterstock and other open source image sites were looked at to perform mass downloads of desired speed limit signs. As many as 3000 extra images of these speed signs were acquired, however it did not seem anywhere near enough to achieve the desired performance from the model after training. The previously mentioned image augmentation methods were applied to these extra images, as well as labelling all of these images manually using the YOLO annotation application (LabelImg). One method of augmentation that was applied to this dataset was reducing the zoom to make the speed signs appear smaller in the image, which had not been applied to any of the images in the previous models. The overview of the newly added images is shown in Figure 40 below:

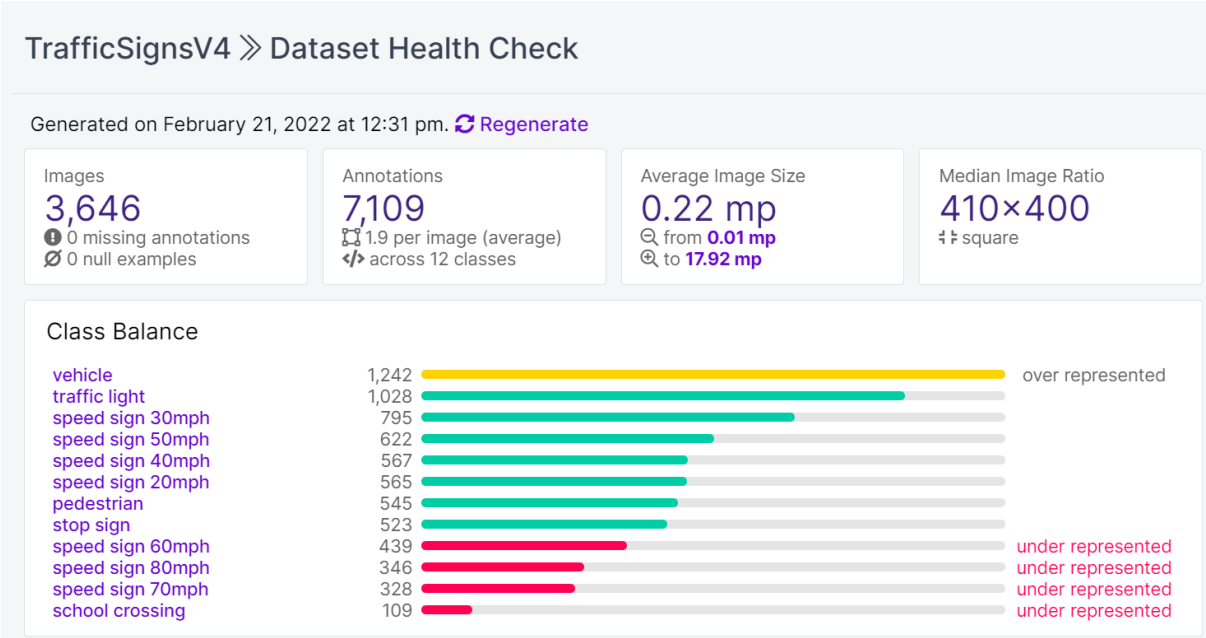


*Figure 40 - 3rd Dataset Class Overview*

In addition to the previous point to improve the dataset, it was also important to prevent overfitting as much as possible. Overfitting is a sign that shows that the CNN is becoming “lazy”, in the sense where it has become very well trained on the dataset it is given to learn. This is typically a result of

several images in the dataset lacking diversity in the context which the object is situated, or the network focuses on finding only the most distinct features of an object and not the other features which also contribute to the formation of that object. [47] The main idea that has been proposed to combat this issue for training is by including “occlusions” in the dataset. Occlusion is a method used to block certain elements of an object in an image, so that the network can learnt to identify an object without having to rely on only looking for the obvious distinct features. Testing the effects of occlusion was conducted in the following papers ( [48], [49] ), which yielded mixed results. It is said that the localisation of objects in the image improved, however object detection itself did not improve. [47] A common occlusion augmentation method is “Cut-Out”, which populates the images with random squares of value “0”, meaning pixels in these squares are completely hidden. However the original paper for Cut-Out augmentation states that the pictures with hidden pixels are only provided to the input layer of the CNN, so the other layers of the CNN could observe the original picture with full context [50].

From this point, more augmentation methods were applied to the dataset via Roboflow. Roboflow provides a wide range of extra options to augment images in bulk, including the Cut-Out augmentation method as shown in Figure 41 below, from which you can adjust the settings to your preference, such as increasing the saturation of an image by 25%, or determining how the random squares should be placed on the images.

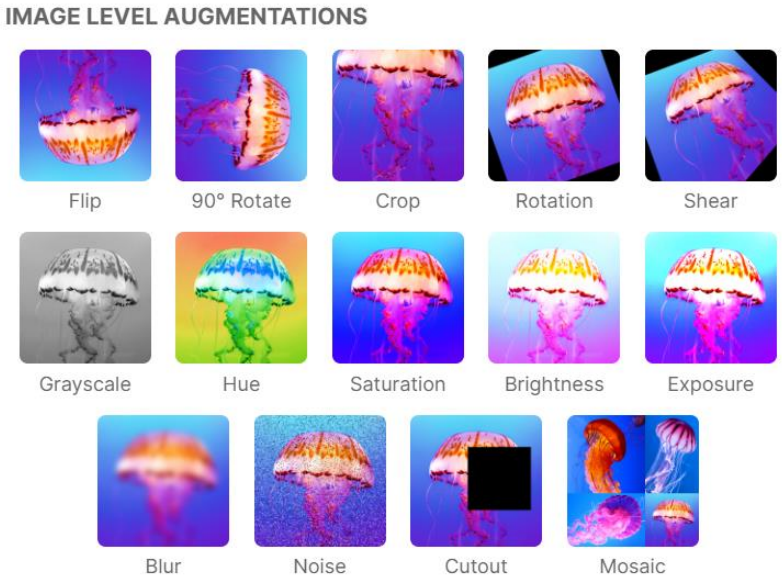


*Figure 41 - Roboflow Augmentation Options*

According to a YOLO paper, during the training of its Classifier and Detector, the algorithm incorporates Mosaic Augmentation and CutMix Augmentation. [36] Both of these augmentation methods mixes images together to provide extra context to the objects being detected. Mosaic mixes 4 images together whilst CutMix mixes 2 images together, however CutMix is more specific with the placement of the image that is being mixed with the other image. Figure 42 shows an example of CutMix, where a cat’s head is placed on another cat’s body. The context is of a cat’s body situated in a new area with different conditions, however the head of a cat is a different breed of cat, which would help the classifier to expect and be able to predict a different breed of cat being placed in those similar conditions. Mosaic however provides even more context to the background which the object finds itself situated in, as it is mixed with 3 other images. Considering YOLO takes each image as a single regression problem, providing more random contexts can only be a benefit for training. In addition to this, each object would be positioned in different regions of the final image and would vary in size across these images. Although YOLO does not perform region-based detection, it still helps the algorithm to be more diverse in where it would look in a frame for a specific object to make predictions. Tests were carried out to see what effect each method of data augmentation had on training as shown in Figure 43 below [36]. Both Mosaic and CutMix had a positive effect on the final

mean average precision of the weighted model separately and an even greater impact when both used together alongside other augmentation methods. CutMix is not available on Roboflow, however Cut-Out and Mosaic are.

*Figure 42 - CutMix & Mosaic Augmentation Techniques*

| MixUp | CutMix | Mosaic | Bluring | Label Smoothing | Swish | Mish | Top-1 | Top-5 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | 77.9% | 94.0% |
| ✓ | | | | | | | 77.2% | **94.0%** |
| | ✓ | | | | | | **78.0%** | **94.3%** |
| | | ✓ | | | | | **78.1%** | **94.5%** |
| | | | ✓ | | | | 77.5% | 93.8% |
| | | | | ✓ | | | **78.1%** | **94.4%** |
| | | | | | ✓ | | 64.5% | 86.0% |
| | | | | | | ✓ | **78.9%** | **94.5%** |
| | ✓ | ✓ | | ✓ | | | **78.5%** | **94.8%** |
| | ✓ | ✓ | | ✓ | | ✓ | **79.8%** | **95.2%** |

*Figure 43 - YOLO Augmentation Tests [36]*

Using Roboflow, four different versions of the added “SpeedSigns” were generated utilising the Cut-Out and Mosaic methods, mainly with both of these methods used separately, and only one of these versions where both methods were combined. When combining these images along with images generated in the 2nd model, a total of approximately 20000 images were generated, with a much more equal spread of the number of instances for each class. However at this stage, it was very difficult to generate a substantial number of images for the Stop Sign, School Crossing and the 60, 70 and 80 mph speed limit classes in comparison to the rest, due to a lack of availability of these pictures online. Thus, at test time, the inference performance for these signs were expected to be below par.

*Figure 44 - Speed Sign Tests for 3rd Model*

Post training results for this model showed an increase on the mAP@50 (80.4% on the 20000 weights file), and overall a significant improvement on the inference performance in video tests. However there seemed to be a lot of instances where the model would incorrectly classify the identity of an object, particularly the speed signs. As shown in Figure 44 above, it could detect the presence of a vehicle in the distance, but not the vans in front of that vehicle, neither could it pick up the van in the bottom left region of the image, which is very partially covered by the motorway barrier. These are the problems which occlusion in the dataset intend to solve, however it did not seem to have an impact in this scenario. Another worrying sign is the 30mph speed limit being detected as a 50mph speed limit. This may be a result of the cut-out augmentation method removing elements of the numbers “3” and “5” that make them easily distinguishable. As previously mentioned, the pictures with the cut-out augmentation were only provided to the input layer of the CNN, however there is no way to make that distinction for the images in our dataset before beginning training on YOLO.
Another important step utilised in this model was converting all the images into a resolution of 416x416, which helped improved the speed of training. The first two models featured images with a wide variety of resolutions and took approximately 10+ hours to train, whereas this model only took 4 hours, saving the YOLO algorithm a significant amount of time from converting the images to the desired input size by itself.

### 4th Model

The next model intended to combine all the ideas gathered from the results of the previous models into a final complete model.
Due to the negative impact Cut-Out seemed to have on object classification, this method of augmentation was no longer used, as to obtain the full benefits of the augmentation method, the original image had to be fed to the latter layers of the network. A copy of the original image may have been passed to these layers, but the outcome would not be the same as the network would not be forced to train itself on other features of the objects via the input image.
Another idea incorporated into this model is levelling out the distribution between the classes, but on a scale where there are close to 20-30,000 instances per class.
This model also includes a much greater volume of images to be trained. Approximately 300,000 images were generated, however in comparison to the MS COCO dataset, the total number of instances across the classes do not compare.

The results of this model were significantly more promising in comparison to the previously generated models. An mAP@50 of 91.82% was achieved on the “last.weights” file, the best score compared to the other weights files, suggesting that there was no overfitting during training for this model. In a lot of cases, the object classification scores were relatively low, but the overall classification of the objects were relatively accurate. For example, the video could inference the presence of a moving car relatively well in a lot of scenarios despite giving a confidence score typically between 40-60%. The model still seems to detect some false positives when it comes to the classification of traffic lights however. Generally for the speed signs, the classification has improved a lot, however the video playback has to get relatively close to the speed sign before giving a concrete prediction. The classification can still be hit and miss however, as we can see at the bottom of Figure 46 mistaking a 40mph speed sign for a 20mph sign. These issues can only be improved with more high quality data being passed into the network for training. To improve the quality of data, there must be more variation in the placement of the objects in the input images, as well as how large the objects are. This would reduce the chances of the network being unable to predict an object purely because of how unfamiliar it is with making predictions in a certain region of an image.


*Figure 45 - 4th Model Testing (1)*


*Figure 46 - 4th Model Testing (2)*

Despite the performance issues of the model, the model is still consistent enough to produce a software solution that could provide instructions (in a demonstrational sense) to a model vehicle. However it is not strong enough to implement fully on a model vehicle, as there is still a lot of work to be done when it comes to ensuring the correct object is detected with greater confidence. Should we implement this model on a model vehicle at this stage, there would be occasions where the model vehicle would be following the incorrect instruction due to the model incorrectly classifying objects too frequently.

## 5.5 Testing SaTS vs COCO

Although we have established the deficiencies of the model's consistency with regards to accurately detecting an object, we must investigate to see if this is a case regardless of the contrast and brightness of the video presented. As discussed in section 3.2.1 of this report, the YOLO algorithm takes an image as a single regression problem and thus uses the context surrounding the object to increase its confidence in the object being detected. As the model was trained using YOLOv4-tiny weights, we will compare object detection for the model based on the COCO dataset and the dataset created at the end of section 5.4.7 (4th model). See below examples of how both models perform on some test images.


*Figure 47 - SaTS vs COCO (1)*


*Figure 48 - SaTS vs COCO (2)*

As shown in the collage of images above, we can see that the performance of SaTS is comparable to COCO on the YOLOv4-tiny network. There are cases where SaTS outperforms COCO, more specifically when it comes to the inference of traffic lights. In the stock images of traffic lights where there are little to no obstructions of the object, SaTS picks up some traffic lights that the COCO model does not. In most cases, the confidence of the traffic lights detected were also higher than the COCO model. When it comes to detecting the presence of vehicles however, the SaTS model really

struggles. We can see in Figure 47 and Figure 48 how inconsistent SaTS is with detecting the presence of vehicles. A few of the test images are taken a few seconds between each other in a video feed, a few seconds between each frame. The SaTS model is shown to not detect a single car, and then detect all vehicles a few seconds later, all under the same light conditions. The cause for this is not evident, however a few things that could be implemented to improve this is by providing the network with a wider range of vehicles, such as different models of cars, as each car is shaped slightly differently. We can see at the top of Figure 48 above that the model especially struggled with 4x4 and saloon vehicles, which are shaped differently to the average car. To improve this, we can also provide images that give a 360 degree view of vehicles. This would make the network more familiar with cars at all angles, as we can see that at times the model does struggle to detect a vehicle from directly behind it, and it actually performs better when the vehicle is slightly slanted at an angle from the side. The SaTS model also struggles to detect the presence of vans and trucks, which does highlight the lack of these types of vehicles in the dataset.

# 6 Software Design

## 6.1 Overview of Object Detection Software

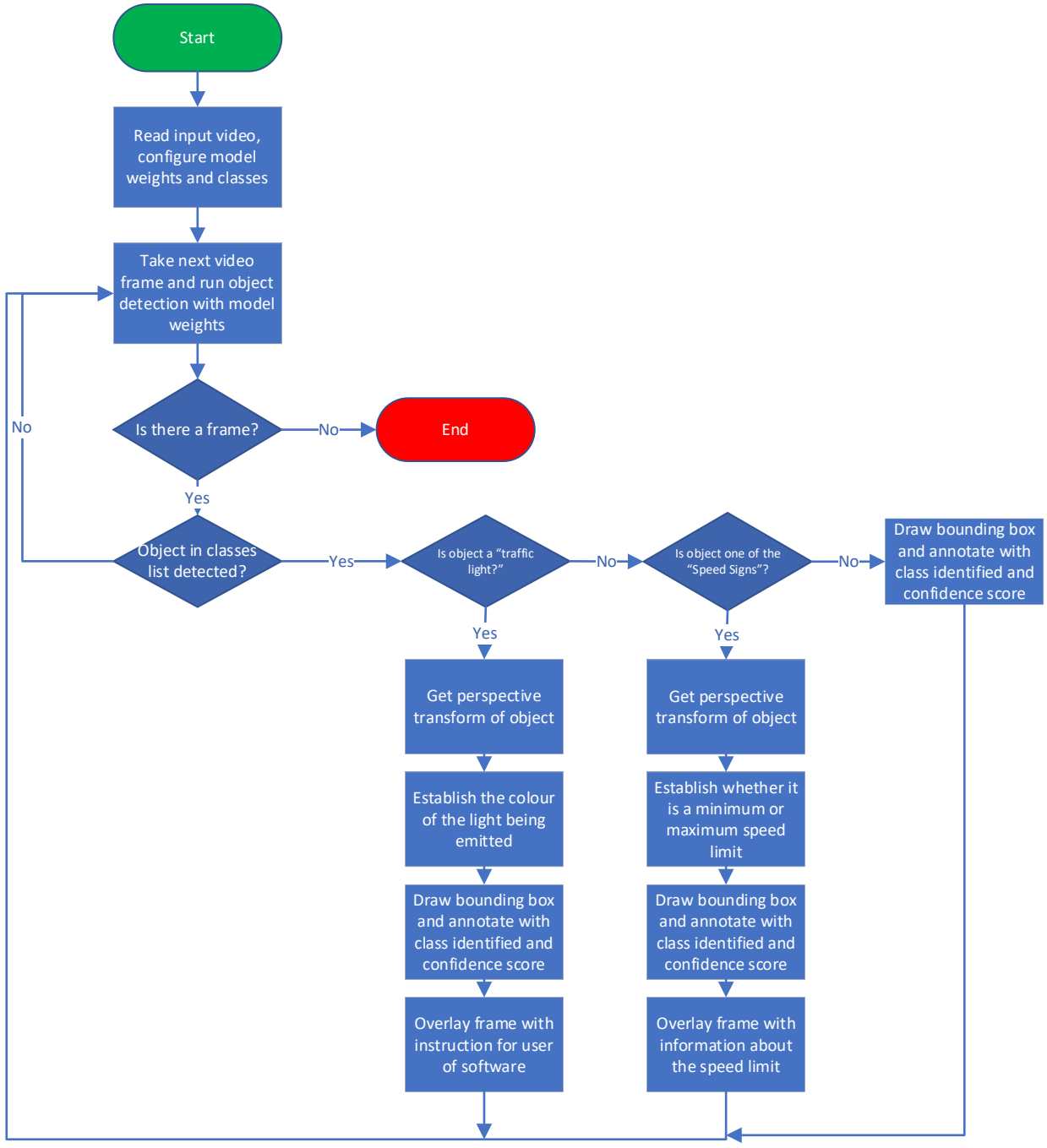


*Figure 49 - Flowchart of Software Operation*

The purpose of this software is to use the model generated in section 5.5, to provide feedback to the user of the software regarding the objects that the model is detecting. From the objects detected, we can extract information about the object and thus come to a logical decision regarding what action should be taken.

To develop this software, footage of the road was recorded from the dashboard of vehicles which emulate a variety of real world conditions, lighting, weather conditions etc. From this footage, we can establish what exactly the model was able to detect and make modifications to the software if needed. Sample images were also taken from google images to establish some parameters to be

used in our code, such as the HSV value of the red light from a traffic light. In this section, we intend to further explore methods that were used to develop this software.

## 6.2 Colour Isolation


*Figure 50 - Examples of Images used for Colour Isolation*

Using the model created, we can extrapolate desired information from the objects detected. Traffic lights and speed limit signs in particular give the road user crucial information just by the colour of the sign or the light emitted. We can use OpenCV to define colours that the software should look for. For this project, the colours were initially identified using sample test images with slightly different lighting conditions to try and replicate a range of potential realistic scenarios. A separate python script was developed specifically for this purpose.

To accurately define colour ranges for these different lighting scenarios, the input frame is converted from a BGR scale to a HSV scale using OpenCV's "inRange" function. BGR (for Blue/Green/Red) is unsuitable for this application because the BGR scale is very specific with how colours are defined. Each scale has a range between 0-255, and each colour is represented via a mix of blue, green and red, which shows how limited the BGR scale is with how it represents colour. It highlights the luminance of the colour detected as opposed to the colour information itself. For example the traffic lights in Figure 50 above, if we were to look for the colour red in BGR scale, the true red values would be defined as (0,0,255). However, in realistic scenarios and lighting, the red that we see would be composed of a mix of red/green/blue values as a result of the lighting context surrounding the desired colour. The HSV (Hue/Saturation/Value) scale allows us to separate the colour information from how intense the light is. This is similar to how humans actually perceive colour, in comparison to the BGR scale which is more useful to a display screen where a pixel is instructed to output a specific colour as a very small part of a bigger picture.

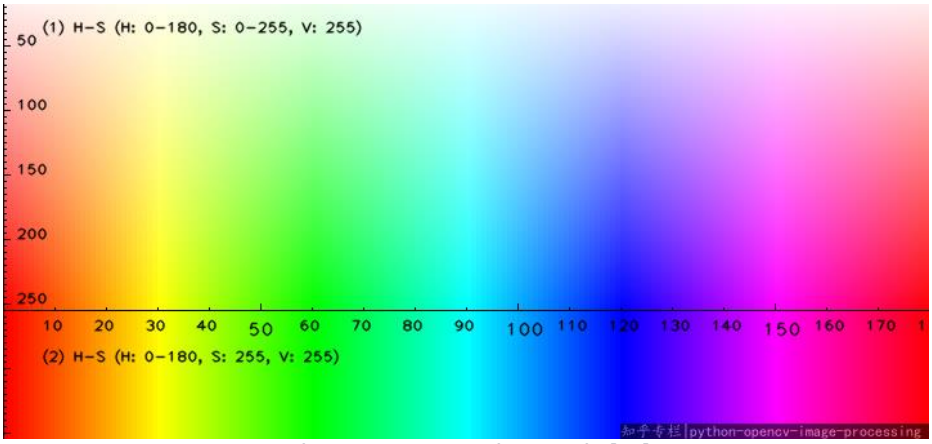

*Figure 51 - HSV Colour Scale [51]*

### Hue
In relation to the HSV chart shown in Figure 51 above, Hue is represented on the x-axis. Every colour is represented on a scale between Hue represents the actual colour that we see, i.e. Green can be represented between values 35-85 on the Hue scale.

### Saturation
Saturation describes how vibrant the colour being detected is, which is represented on the y-axis of the HSV chart. The larger the saturation value, the more vibrant the colour is.

### Value
Value describes the intensity of the colour, primarily associated with the light level of the colour (how bright or dark it is).

This scale is much more ideal to determine the colour of an object as we can account for the light intensity of a particular colour regardless of the lighting conditions which would be presented in an image frame. Once the frame has been converted to HSV, we can investigate the desired colour ranges of a particular colour. Each desired colour has an upper and lower limit to represent the colour in a vibrant lighting scenario and a darker/more dull scenario. This process was repeated for other traffic lights and speed limit signs.

This was achieved with the help of a “mask”, so we can focus on isolating the desired colour from the test image. Masks are produced as a result of “bitwise” operations. For example, when the desired red colour is isolated from the test image, each pixel with the desired red colour is assigned the value “1” and is depicted as white in our mask version of the frame, whilst the unwanted colours are hidden in black with a pixel value of “0”.  An example of this is shown in Figure 52 below.

When we achieved the desired results, we could detect each traffic light and count the number if pixels that fall within each colour range so the traffic light state can be detected. So if a traffic light is detected and it features more pixels from the green light than the red light, we can establish that it is a green traffic light. This was achieved using OpenCV’s “`countNonZero`” function.

Once we establish the difference in colours required to determine the state of the traffic light, we are able to output this information on the objects label so the user can see that the software understands the difference between a red, amber and green light.


*Figure 52 - Example of Isolating Red Traffic Light*

There are scenarios that can confuse the software if the colour ranges hat are established are not correct or robust enough. For example, traffic lights are commonly black with coloured lights; however if a situation arises where the traffic light is yellow, the software will count the yellow pixels which will end up being more than any of the coloured lights emitted. To counter this, the HSV value of desired colours have to be more specific to suit the object being detected. For example, a smaller Hue range for Amber so it does not branch into the Yellow hue region would counter the software from counting pixels of strictly yellow traffic lights.

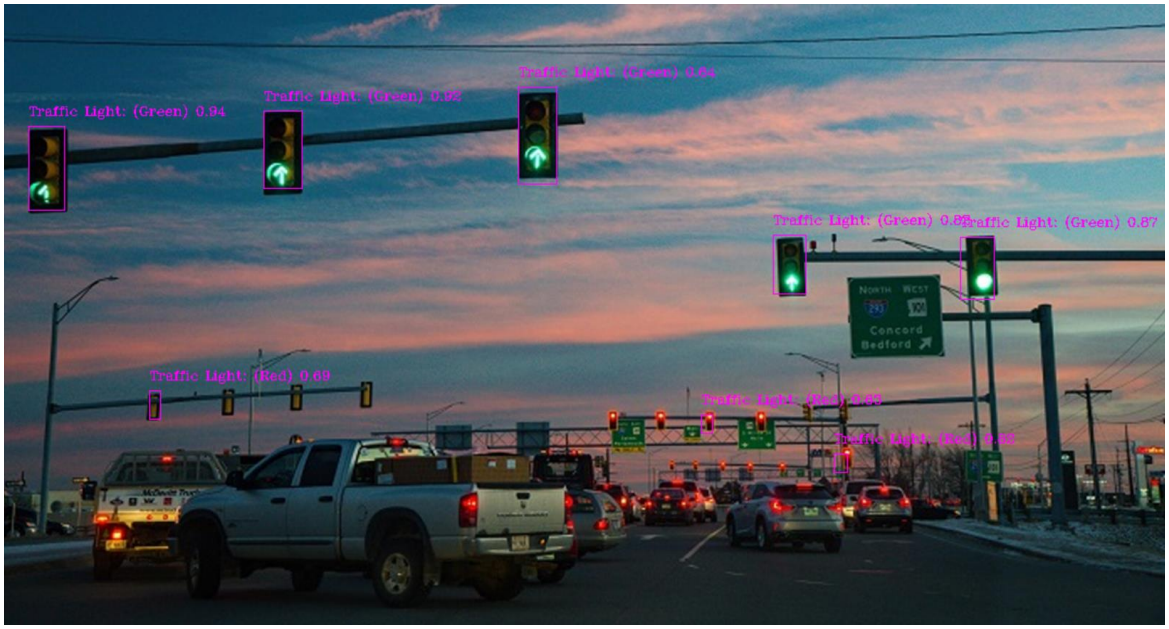


*Figure 53 - Annotating Traffic Light according to their states*

A similar approach was taken for speed signs, however a mask was created to detect blue pixels for minimum speed limits and red pixels for maximum speed limits. From the speed signs, there are some examples which show what exactly the neural network detects. Despite the fact a box is drawn around the object, it seems to purely focus on what it can detect of the object itself as opposed to the context provided within the box drawn. As shown in Figure 54 below, we can see an example of a speed limit sign with a blue background which is the same blue that would be used on a minimum speed limit sign. However the model picks up the sign and comfortably counts significantly more red pixels than blue pixels, suggesting that we may assume that when objects are detected, it detects the actual object rather than the bounding box surrounding it, which the model was trained to do as shown in section 5.4.7.

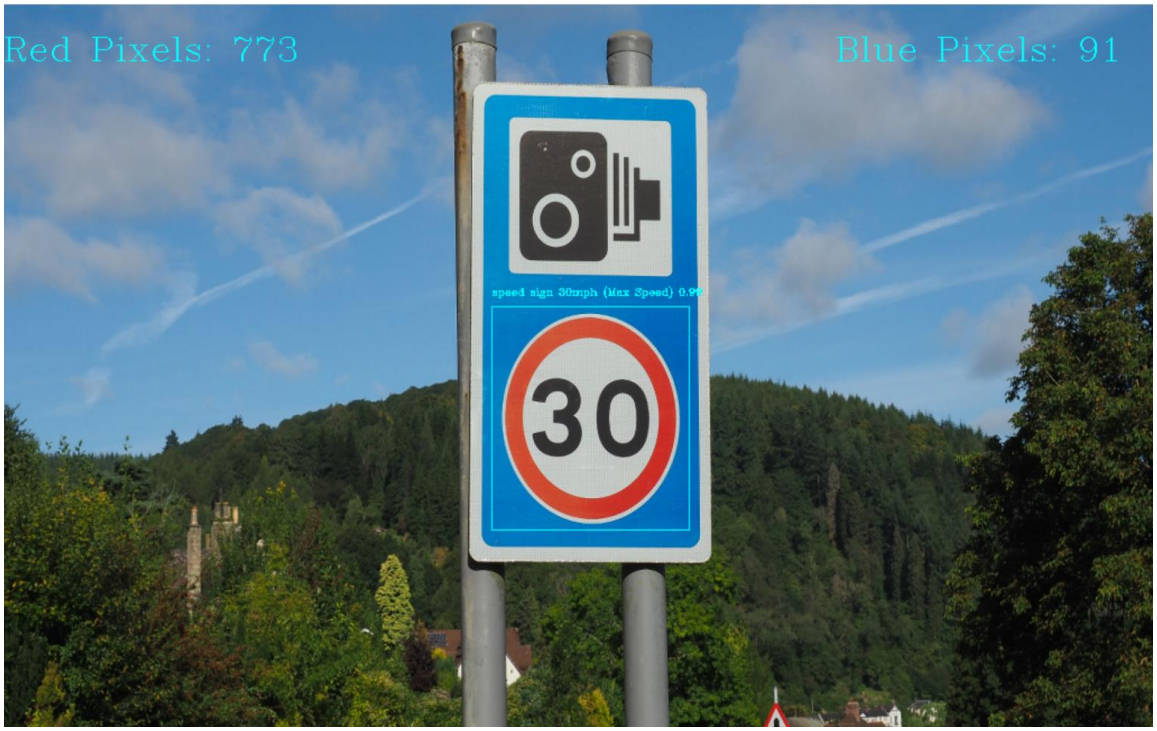


*Figure 54 - Coloured Pixel Count for detected speed sign*

*Figure 55 - Incorrect Colour Range (1)*

The importance of ensuring the colour ranges are robust enough to be accurately detected in a range of lighting conditions are shown in Figure 55, Figure 56 and Figure 57. In Figure 55, traffic lights are correctly identified, however due to the colour range of Amber not being specific enough, the yellow

of the traffic light was picked up, thus producing the wrong state. In Figure 56 the correct sign is detected, the lighting of this image fools the software into thinking that this is a minimum speed limit, and understandably so, as the disparity between red and blue isn't so clear here. The red is shown to be leaning into the Purple range of Hue values with significantly less saturation.


*Figure 56 - Incorrect colour range (2)*

In Figure 57 below, the red present on the sign indicates the speed sign is for maximum speed, however the blue tint in the background of the sign would fool the software into thinking it is a minimum speed limit sign. It is important to find more contexts like these to avoid confusing an autonomous vehicle if it were to use this software, otherwise it would find itself speeding in a maximum speed limit zone.


*Figure 57 - Example of where incorrect colour range would result in incorrect identification of speed sign state (3)*

## 6.3 Model Configuration

As shown in the flowchart in Figure 49, the initial stages of the software begin with reading the input video to extract a frame. This occurs in a while loop so every single frame of the video is taken and object detection is run on it using the model we created in section 5.4.7. Each video is resized to 1080x720 resolution. Before the while loop begins however, we initialise the neural network we want the software to utilise.

To utilise the weights file and configuration file, we specify the path to these files and use them as the input to OpenCV's "readNet" function in its DNN module. The model is then configured to match the parameters we gave it in the configuration file, such as specifying the resolution of the input data (416x416) and the state of the red/blue colour channels (they are swapped). We are also able to specify that we want to enable CUDA's Backend and Target settings within the CUDA Toolkit which was installed earlier in this project. We must also load in the class names that can be detected with this model from our "obj.names" file so they can be used for labelling and attaching a confidence score to each object identified within each frame. More parameters can be specified for the network, such as threshold levels for NMS suppression, and the minimum confidence score a detected object should have before a box is drawn around it. As our model is not the most consistent, both of these parameters are set to 0.4.

## 6.4 Creating a Heads-Up Display (HUD)

The next stage of our flowchart shows that we run object detection on the frame. If no objects are detected we simply move onto the next frame and run object detection on that frame until we can detect objects that are in one of our pre-specified classes. The object detection process returns a list consisting of the class name of the object detected, the coordinates of the top left corner of the box as well as the width and height of the box to be drawn around the object, and its confidence score per object. This is after NMS suppression and the confidence threshold is applied to the object, so the object is provided with the best possible confidence score. If the object is detected isn't a traffic light or speed limit sign, a box is drawn around it, which is annotated with the confidence score. However if the detected object is a traffic light or a speed limit sign with a confidence greater than a score of 0.6, more steps are required for processing to produce the output that shows we can identify the difference in colours emitted by these objects. Thus, from the colours emitted, we can display an output message of instruction for the user to interpret.

### Traffic Light

If the object detected is a traffic light, warp perspective is performed on it, so we can gain a bird's eye view of the object utilising the box the model is about to draw around the object. From this warped image of the detected object, we can create a masked version of the warped image for each colour (red/amber/green), so we can count the number of pixels each colour has and compare them to determine the state of the traffic light.


*Figure 58 - Example of Warp Perspective on detected object*

Once the model has decided what the state of the traffic light is, we can provide a message to the user with instructions by overlaying an instruction on top of the video feed. The overlaid message constantly updates depending on the state of the traffic lights detected with the highest confidence at that moment in time.


*Figure 59 - Overlay telling user to stop*

## Speed Signs

If the object detected matches the class name of one of the speed signs, warp perspective is also applied to the object. However in this scenario, we want to determine whether it's a maximum speed limit (indicated by a red circle around the outer edge of the sign), or a minimum speed limit (blue background in the sign). A similar process was applied to detect speed signs, by using perspective transform and warping that image of the object to provide a bird's eye view to prepare it for pixel colour counting. Different HSV values were provided for red pixels compared to what was used for traffic lights, as the red emitted in traffic lights are much more consistent regardless of the lighting conditions. Once the colour information of the object is established, we can provide a message via an overlay indicating what speed limit was detected and whether it was indicating maximum or minimum speed.


*Figure 60 - Minimum and Maximum Speed Limit Overlays*

Combining the overlay functions for both the traffic light and speed sign detection produces an interface shown in Figure 61 below.

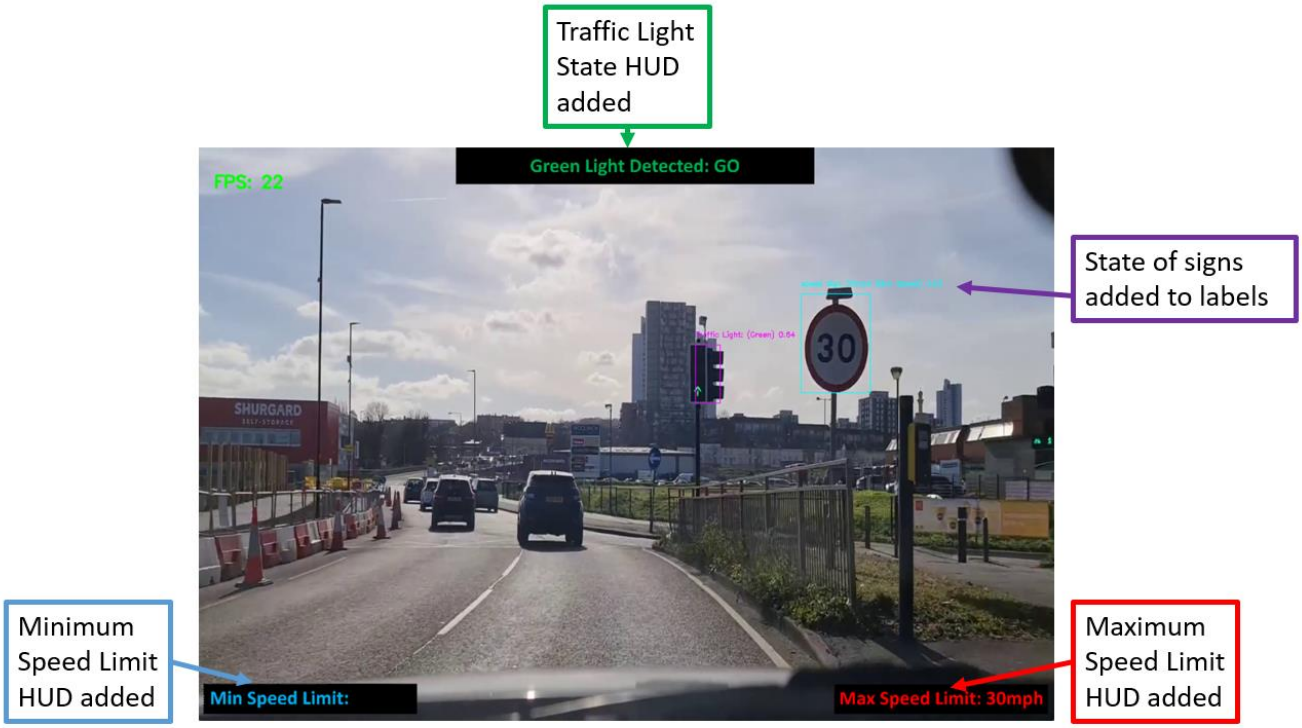


*Figure 61 - Final HUD showing information for states of traffic lights and speed signs*

## 6.5 Further Work

With the completion of the software solution, we have observed a few methods of extracting information that is processed by humans from these signs to execute instructions without thinking. However to simulate proper driving behaviour with an autonomous vehicle with this software, a lot of features need to be added to ensure the best possible performance is achieved. As discussed in sections 5.4.7 and 5.5, we are aware of how weak the object detection model is if we want to compare it to a level which would be safe for state of the art projects currently in development. Other features such as lane detection need to be added so a region of interest (ROI) can be determined with regards to what objects the vehicle actually needs to focus on. More details regarding improvements to the software solution are discussed below.

### 6.5.1 Dataset Improvements

As discussed in Section 5.5, the SaTS model showed that it struggles to detect the presence of vehicles at times in comparison to the COCO model. It is inconsistent regardless of the lighting conditions or if there are objects obstructing the view of the vehicle. Even if the frame shows from a person's point of view on a road that we are directly behind the vehicle, there is no guarantee that the SaTS model will detect its presence, let alone with a high confidence score. As a result, whilst designing the software, we had to lower the confidence threshold of the network (set to confidence of 0.4) so that objects detected with a lower confidence can have bounding boxes drawn around them. This means that falsely detected objects would be presented to the user of the software. The end goal is to have this software be utilised by an autonomous vehicle, and having a low confidence threshold is detrimental as it could throw the autonomous vehicle off, or mislead the vehicle into following the wrong instructions if an object in the SaTS model is detected incorrectly. The main way to improve this issue is increasing the variety of vehicle data in the original dataset. More images that depict the complete 360 degree view of a vehicle will help the algorithm to expect a vehicle at particular angles in first person view. The same can also be done for translating the vehicle around the frame of different zoom sizes. For example a car driving towards the top of a hill would be depicted as a much smaller vehicle and potentially towards the top left or right corner of a frame depending on how the road is shaped. Thus feeding the algorithm more mini-sized vehicles at a top

corner of a frame would reduce uncertainty of a vehicle at test time as the algorithm is trained to expect vehicles at any place in a frame.

A similar thing can also be said for speed sign detection in the SaTS model. Unfortunately, due to the lack of speed limit signs specifically online, it was very difficult to generate a good number of instances of speed signs for training. Especially with stop signs and school crossing signs, it was very difficult to produce a huge volume of these classes, and thus being able to demonstrate detection of these classes was near enough pointless. A lot of the images used were stock images without any context, or very similar images of a speed sign with a bright sky in the background without any occlusion or other objects obstructing the speed sign. Images also lacked diversity with regards to how they were positioned in the image. Many images had speed signs that were very central and would occupy the entire frame. This affected where the speed sign had to be in the frame at test time for the model to detect its presence. For example, a traffic light could be detected in the distance, however with a speed sign, it had to be a lot closer in the frame to be detected. This would mean there is less opportunity for the model to make sense of which speed sign it is looking at, as in a lot of cases, the frame would almost completely go past the speed sign before being able to produce a proper identification of the sign. As there are limited resources available online to resolve these issues, applying extra augmentation to the images we already have can be of real benefit just to aid detection of a speed sign earlier in a frame. This would allow us to process the frame even more and help control the motion of an autonomous vehicle if it were to adopt this software. Augmentation methods such as reduced zoom of the object as well “Cut-out”, which is discussed in section 5.4.7, would help with placing the object of reduced size in different places of the frame, so the YOLO algorithm can be more accustomed to a speed sign being smaller and can expect it to appear anywhere in frame. Although YOLO does treat each frame as a single regression problem and the region in which the object is detected does not matter as much, it would still be good practice for the algorithm during training time. Rather than just displaying that a speed limit sign has been detected, we can signal that the autonomous vehicle may need to slow down as it is approaching the speed sign, which is much more natural human driving behaviour as opposed to just immediately slowing down as soon as the speed limit sign has been passed. This would limit potential accidents caused from robotic behaviour.


*Figure 62 - Example of SaTS model being unable to identify a speed sign that is too distant*

Traffic Light detection excelled and is a highlight in this model. It detects the presence of traffic lights very well to the point where it outperforms what was achieved on the COCO model, as well as consistently producing higher confidence scores. The majority of the traffic light segment of the dataset was taken from Googles Open Image Dataset, however from how it performs and the types of images present in the dataset, there is a much wider variety of traffic light sizes and translations. As previously mentioned, the traffic light could be detected at a far distance as depicted in an image frame, and thus the state of the traffic light could be determined very early. This means a decision could be made very early with regards to what an autonomous vehicle should be prepared to do as

it approaches a traffic light, thus allowing us to provide instructions to the user of the software with regards to if it needs to slow down or carry on driving as normal at the presence of a traffic light.

### 6.5.2 Identifying Objects with Shapes?

Another addition that could potentially improve the software is the ability to recognise shapes where necessary. An example of this would be arrows shown by traffic lights when vehicles are allowed to go only if they intend to turn left, right or straight.


*Figure 63 - Traffic Lights with Arrows*

As shown in Figure 63 above, the software accurately detects the presence of the red light present, however those who want to turn left in this instance are able to as the green light present is in the form of an arrow. The standard shape of these arrows are the same across the UK, thus setting up the software to identify the shapes using OpenCV is something that can be implemented. As demonstrated in section 6.4 with regards to how the “warpPerspective” function is used to obtain a bird’s eye view of the object in the bounding box, we can also conduct a check to see if the arrow is also featured on the traffic light. The next check would be to identify the orientation of the arrow to determine what direction the vehicle must go. Then we can check to see if the colour green is actually being emitted from the light so that information can be displayed to the user that they can only turn left should the arrow be lit.

### 6.5.3 Lane Detection

Figure 64 below shows a scenario where are multiple traffic lights that are in opposite states to control traffic in different lanes. Although correct information is presented regarding the state of the traffic lights, the information is not helpful for our user as the vehicle is not in the lane where the traffic lights are red. Implementing lane detection can help us specify a region of interest either side of the detected lanes for traffic lights and speed signs that we want to detect for the lane our vehicle is occupying. This would prevent confusion for the vehicle using the software, and in a real world scenario this would prevent accidents due to the autonomous vehicle following incorrect instructions. This is a huge reason as to why implementing the model vehicle with the model in its current state is not logical.

*Figure 64 - Multiple Traffic Lights of different states in different lanes*

Another example where Lane Detection can prove to be useful is the potential to implement overtaking vehicles or switching lanes if it logically makes sense to. This would be achieved by identifying the contour of the lanes and then the presence of vehicles on the lane. This would require a strong model to detect the vehicles position on the lane at all times, as the vehicle could decide to speed up or slow down, or if there are other vehicles ahead of the vehicle directly in front of our vehicle and the gap between them, all of which can affect the decision making process as to whether it is safe to overtake the vehicle ahead or not. As well as overtaking other vehicles present on the road, there are other scenarios where lane detection would be useful; such as turning at a junction, following road markings, navigating to the correct lane when approaching a roundabout, switching lanes to exit a roundabout, and many more scenarios. We can begin controlling the motion of vehicles to emulate human behaviour based on the destination of the vehicles and the lanes that the vehicle must take up.
Lane detection can also help a vehicle follow directions set by a satnav. Satnav directions can be added to the HUD along with a geographical map of where the autonomous vehicle is, so the vehicle knows when and where exactly to make a turn as instructed by the satnav.

### 6.5.4 Distance/Speed of Objects in Frame

As previously discussed in section 2.1, rather than implementing LIDAR sensors to help with perception of distance and depth presented from the video feed, it would be good to be able to implement a similar concept with OpenCV. One approach that can be taken to achieve this is to use Pythagoras from the bottom corners of the input video feed to the bottom edge of the bounding box drawn around detected objects. A scenario where this would be useful would be bounding boxes drawn around vehicles presented directly in front of the view shown in the camera frame. An example of this is shown in Figure 65 below, where we use the coordinates of the bottom corner pixels of the frame as well as the bottom corner pixels of the bounding box drawn to get an angle and the opposite length to θ to get the distance from the vehicle to the front of our vehicle, which is represented by the bottom edge of the frame. A safe separation guideline is to maintain a time of 2 seconds between yourself and the vehicle ahead [52], which equates to approximately a separation distance of 1 metre for every 1mph your vehicle is travelling at. This would be difficult to implement as we would require the precise distance from the source of the camera frame to every object detected within the frame. State-of-the-art projects accomplish this with the help of LIDAR sensors, or companies like Tesla have implemented this by accurately defining the depth that each pixel presents with computer vision techniques. From obtaining an accurate depth, speed calculations can be made for the actual speed of moving objects in the frame. An autonomous vehicle system would have speed information from the motors which could be used to aid with these calculations. Then the relative velocity of vehicles

surrounding the autonomous vehicle can be made, and we can extract information such as whether vehicles are traveling towards or away from our camera frame, and how far away each object is from us.
The main issue with the SaTS model to implement this is how weak it is at inferencing vehicles. Upgrading the model to be utilised on YOLOv4 would definitely improve it, to what extent however we have no idea until we have access to more capable hardware. It would need to be very consistent with detecting the presence of vehicles otherwise a situation could arise where the chance of an accident is drastically increased due to the model being unaware of where the vehicle in front is. Another solution to this problem would be to train our model with more images of vehicles with methods previously described in section 5.4.7.

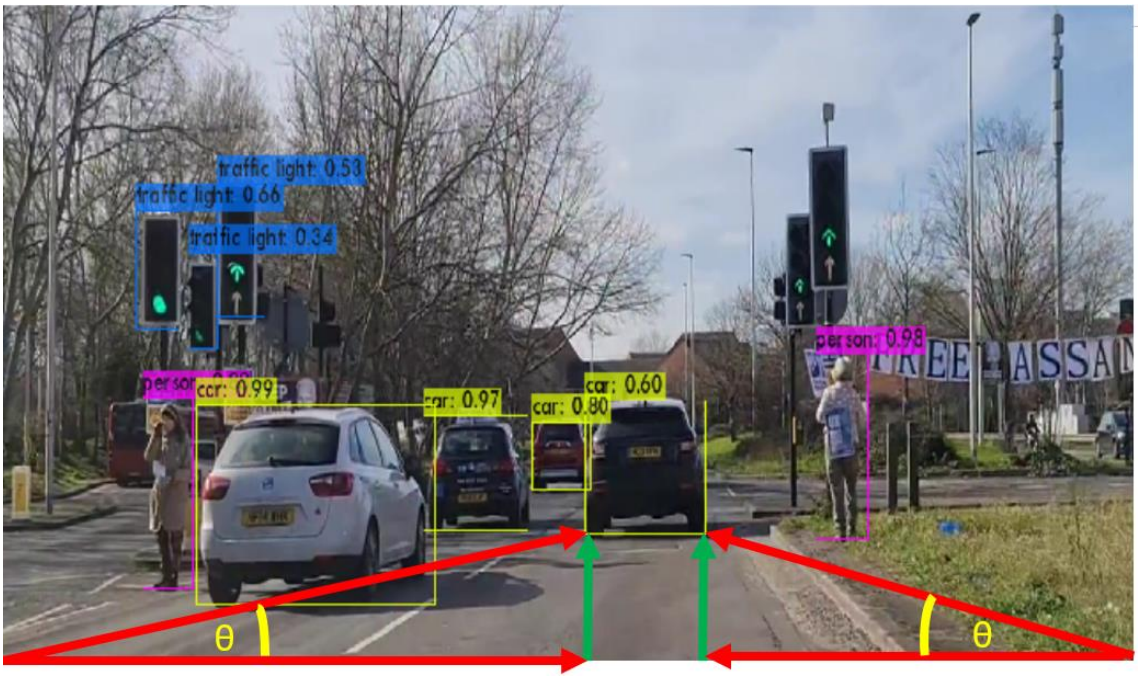


*Figure 65 - Pythagoras calculations to calculate distance of vehicles in front*

# 7 Project Proposal Review

A software package was developed that can identify speed limit signs, vehicles and traffic lights, as well as provide instructions to the user of the software via HUD, depending on what object had been detected from the video input. Using a variety of test videos on the software, we identified weaknesses in the object detection of input videos with a lower resolution as well as videos presenting the objects in difficult conditions, such as videos taking place in the night, or areas where the lighting is not natural. The software struggled to consistently localise the presence of an object, typically it would draw a bounding box around the object with a fairly high confidence score approx. 60%), yet the bounding box would only be drawn for a very short period of time (<0.5 seconds). As previously discussed in section 5.1.2, there is a significant difference in performance of localising the presence of an object, as well as the identification of the object once detected when using YOLOv4 as opposed to YOLOv4-tiny. One of the risks outlined in the project proposal was the lack of processing power of our device to achieve desired results. This project was carried out using an NVIDIA GeForce RTX 3050Ti with 4GB VRAM. The processor has a CUDA Compute Capability of 8.6, however the lack of VRAM means we were not able to train the model on YOLOv4 to achieve the best possible results using a YOLO framework. To mitigate this risk, YOLOv4-tiny was used instead for this project, however the significant drop in performance compared to YOLOv4 means that the model was not strong enough to even consider using a robot car to demonstrate a robot following the instructions provided by the HUD. As outlined in section 6.5.3, should we have built the robot vehicle, more things must be considered to have the robot successfully follow instructions. Section 6.5.3 highlights the importance of implementing a form of lane detection so that the robot car follows instructions that truly applies to the vehicle. As shown in Figure 64, we can see two sets of red traffic lights, one set are guiding vehicles to another exit whilst the vehicle recording the footage is following another set of green traffic lights. The software currently detects all traffic lights and their states in the frame, however, how can we ensure the vehicle follows the correct set of traffic lights? Methods to achieving this were highlighted in section 6.5.3, but this meant we had to reassess our final objectives with regards to the model vehicle tasks.

Figure 66 shows the original time plan along with the milestones that were set, and Figure 67 shows how long it took to complete milestones. The delivery of the software package only occupied 3 of the 8 milestone targets, however as previously discussed, attempting to implement the object detection model on the robot car was not logical. As a result of poor planning and research prior to the beginning of the project, a lot of the time frames set to complete each milestone were unrealistic. For Milestone 1 (M1), significantly more time was required to understand how to build and train a model on the YOLO framework, let alone a model that performs consistently well. More time was required to research existing datasets, how many images and annotations these datasets had of objects and how we could apply techniques used in these datasets for the model created in this report. Then a significant amount of time was taken to generate augmented images and move them around, which computationally is hugely demanding for any processor being used to create a model.

The aspect of the project where we intended to identify lanes so we can detect turnings at junctions for a vehicle to turn into could not be implemented due to the nature of the project. Perhaps if we were to implement a GPS following vehicle that would be guided to turn into a junction, it would have been more logical to implement this feature, however the nature of the input videos used at testing were based on drivers that had made their own decisions to turn into a junction. A better way to have addressed a feature like this would have been to implement Lane Detection, which has been previously outlined as further work to implement on this project in section.

Milestones 4-8 were related to the implementation of a model vehicle, which did not happen due to reasons discussed earlier in section 6.5. Perhaps with access to a GPU with more VRAM, a better performance of object detection could have been attained and more useful to implement on a model vehicle, however this was not the case.

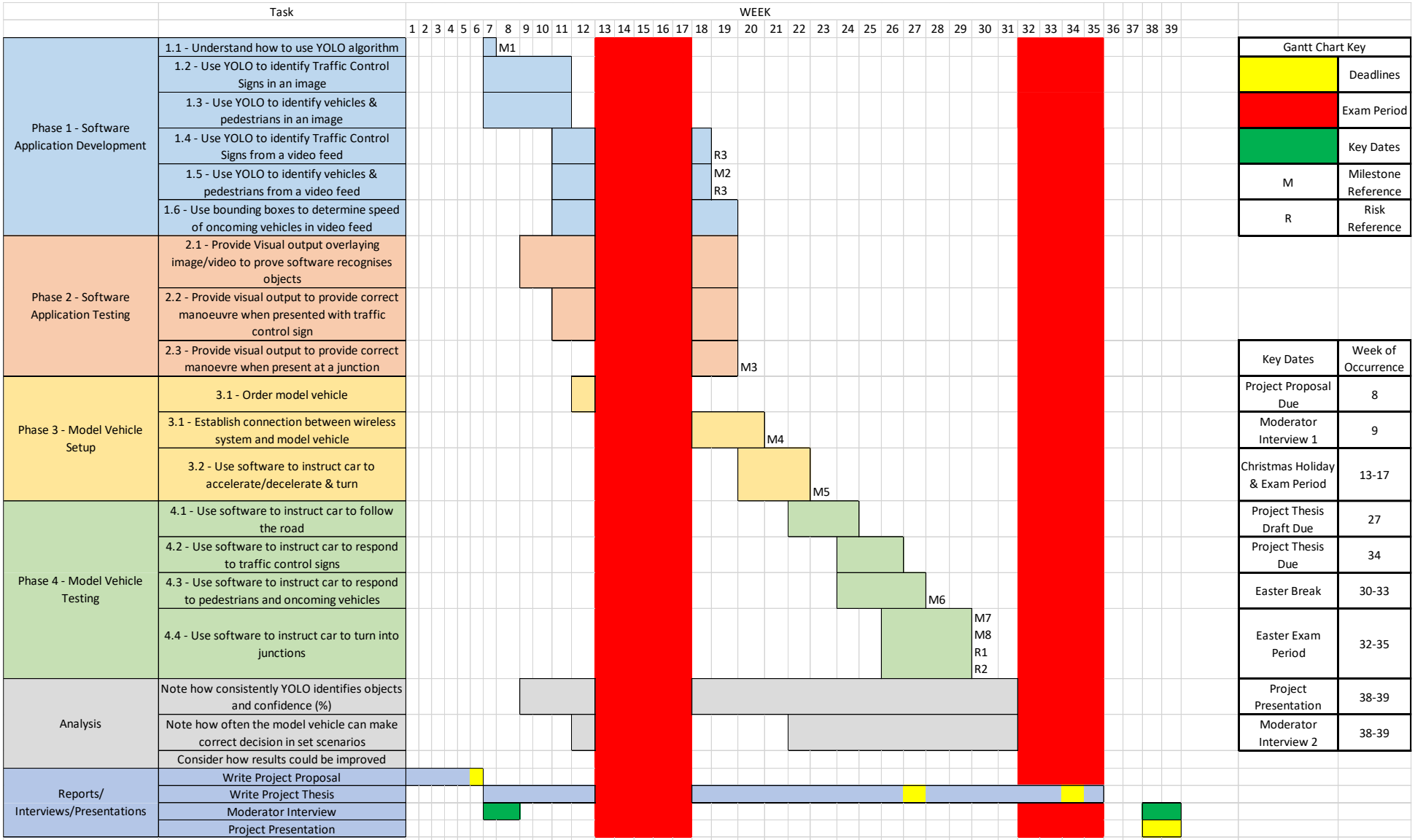


*Figure 66 - Original Time Plan*

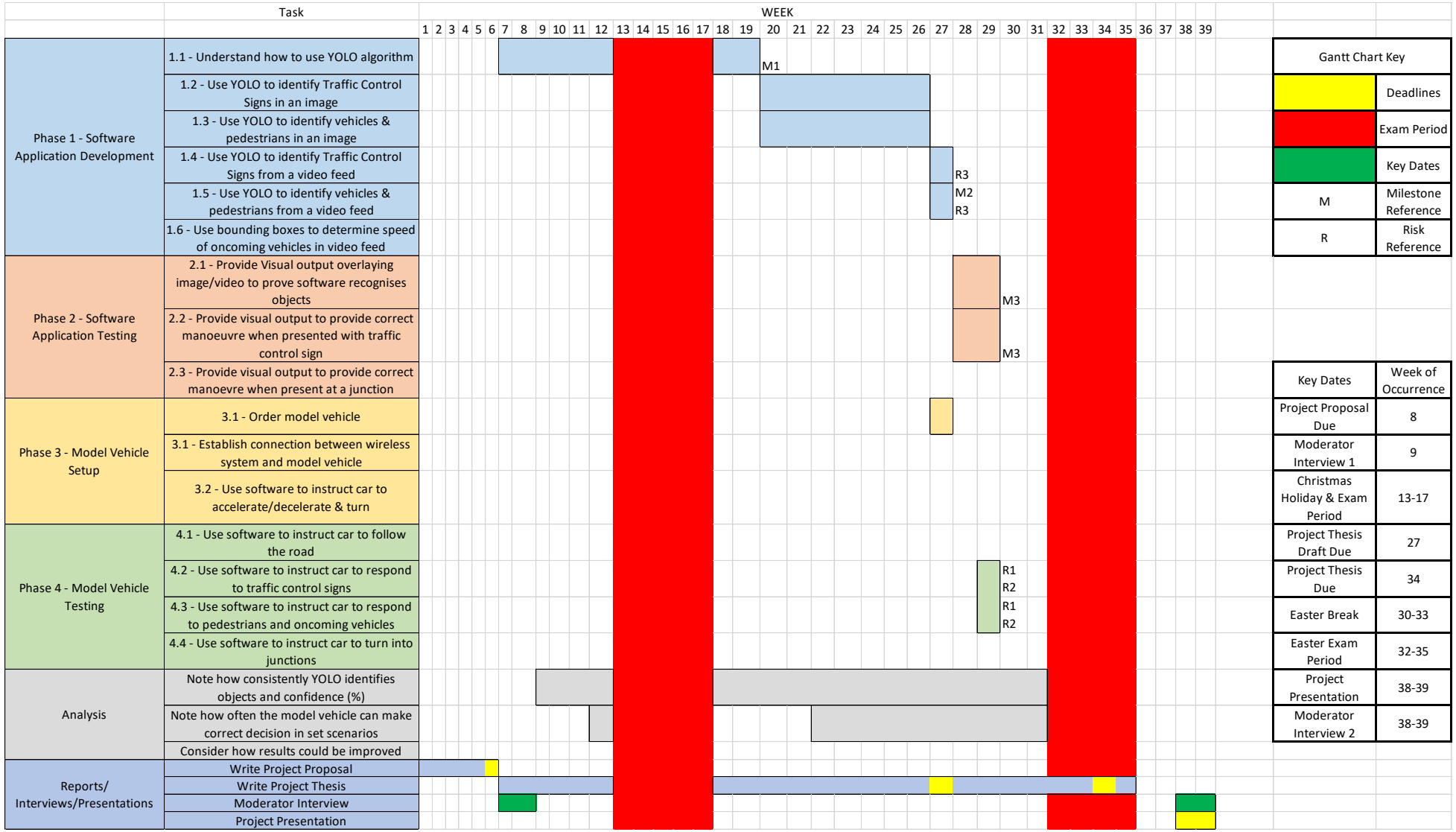


*Figure 67 - Actual Execution of Project*

# 8 References


[1] O. Goldhill, “We’ve had driverless cars for almost a hundred years,” QUARTZ, 22 October 2016. [Online]. Available: https://qz.com/814019/driverless-cars-are-100-years-old/. [Accessed 19 April 2022].

[2] D. Rice, “The driverless car and the legal system: Hopes and fears as the courts, regulatory agencies, waymo, tesla, and uber deal with this exciting and terrifying new technology,” *Journal of Strategic Innovation and Sustainability,* vol. 14, no. 1, pp. 134-146, 2019.

[3] Innovate UK, “UK Transport Vision 2050,” 5 August 2021. [Online]. Available: https://www.gov.uk/government/publications/uk-transport-vision-2050. [Accessed 27 October 2021].

[4] SAE International, “SAE Levels of Driving Automation™ Refined for Clarity and International Audience,” SAE International, 3 May 2021. [Online]. Available: https://www.sae.org/blog/sae-j3016-update. [Accessed 28 April 2022].

[5] Carscoops, Youtube, 31 January 2020. [Online]. Available: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DfKXztwtXaGo&psig=AOvVaw2d6c4f77ibaPC1EqaWp_-B&ust=1651426516146000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCJijt7KpvPcCFQAAAAAdAAAAABAD. [Accessed 28 April 2022].

[6] Tesla, “Artificial Intelligence & Autopilot,” Tesla, [Online]. Available: https://www.tesla.com/en_GB/AI. [Accessed 29 October 2021].

[7] “Transitioning to Tesla Vision,” Tesla, [Online]. Available: https://www.tesla.com/support/transitioning-tesla-vision. [Accessed 29 October 2021].

[8] H. Jin, “Explainer: Tesla drops radar; is Autopilot system safe?,” Reuters, 2 June 2021. [Online]. Available: https://www.reuters.com/business/autos-transportation/tesla-drops-radar-is-autopilot-system-safe-2021-06-02/. [Accessed 29 October 2021].

[9] V. LIDAR, “A Guide to Lidar Wavelengths for Autonomous Vehicles and Driver Assistance,” VELODYNE LIDAR, 26 March 2021. [Online]. Available: https://velodynelidar.com/blog/guide-to-lidar-wavelengths/. [Accessed 26 April 2022].

[10] Velodyne Lidar, “What is Lidar?,” Velodyne Lidar, [Online]. Available: https://velodynelidar.com/what-is-lidar/. [Accessed 26 April 2022].

[11] G. Sharabok, “Why Tesla Won’t Use LIDAR,” Towards Data Science, 1 September 2020. [Online]. Available: https://towardsdatascience.com/why-tesla-wont-use-lidar-57c325ae2ed5. [Accessed 26 April 2022].

[12] N. Cristovao, “98 Safety Scorers to get Tesla FSD Beta 10.5 instead,” Not A Tesla App, 12 November 2021. [Online]. Available: https://www.notateslaapp.com/news/633/98-safety-scorers-to-get-tesla-fsd-beta-10-5-instead. [Accessed 26 April 2022].

[13] HumanDrive, “HumanDrive project in countdown to Grand Drive,” HumanDrive, 4 September 2019. [Online]. Available: https://humandrive.co.uk/humandrive-countdown-to-grand-drive/. [Accessed 28 April 2022].

[14] HumanDrive, “NISSAN LEAF COMPLETES THE UK’S LONGEST AND MOST COMPLEX AUTONOMOUS JOURNEY,” HumanDrive, 5 February 2020. [Online]. Available: https://humandrive.co.uk/nissan-leaf-completes-the-uks-longest-and-most-complex-autonomous-car-journey/. [Accessed 27 October 2021].

[15] M. K. Pargi, B. Setiawan and Y. Kazama, “Classification of different vehicles in traffic using RGB and Depth images: A Fast RCNN Approach,” in *2019 IEEE International Conference on Imaging Systems and Techniques (IST)*, 2019, pp. 1-6.

[16] A. Piergiovanni, V. Casser, M. S. Ryoo and A. Angelova, “4D-Net for Learned Multi-Modal Alignment,” in *Computer Vision and Pattern Recognition*, 2021.

[17] L. Bouchard, “Combine Lidar and Cameras for 3D object detection - Waymo,” 25 March 2022. [Online]. Available: https://www.louisbouchard.ai/waymo-lidar/. [Accessed 28 Aprl 2022].

[18] R. Molla, “Self-driving cars: The 21st-century trolley problem,” Vox, 6 October 2021. [Online]. Available: https://www.vox.com/recode/22700022/self-driving-autonomous-cars-trolley-problem-waymo-google-tesla. [Accessed 2022 April 29].

[19] Waymo, “Waymo One,” Waymo, [Online]. Available: https://waymo.com/waymo-one/. [Accessed 28 April 2022].

[20] E. Houghton, “Is the public ready for self-driving cars?,” 18 January 2022. [Online]. Available: https://www.dgcities.com/blog/is-the-public-ready-for-self-driving-cars. [Accessed 18 April 2022].

[21] Department for Transport, “Reported road accidents, vehicles and casualties tables for Great Britain,” 24 June 2021. [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1021695/ras50002.ods. [Accessed 27 October 2021].

[22] A. Charlton, “Volvo: We will be responsible for accidents caused by our driverless cars,” 9 October 2015. [Online]. Available: https://www.ibtimes.co.uk/volvo-we-will-be-responsible-accidents-caused-by-our-driverless-cars-1523260. [Accessed 1 November 2021].

[23] S. Szymkowski, “Tesla FSD 'issues' in 10.3 update result in rollback, before apparent fix,” Cnet, 25 October 2021. [Online]. Available: https://www.cnet.com/roadshow/news/tesla-fsd-full-self-driving-issues-fix-update/. [Accessed 1 November 2021].

[24] IBM, “What is computer vision?,” IBM, [Online]. Available: https://www.ibm.com/topics/computer-vision. [Accessed 29 October 2021].

[25] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu and M. S. Lew, “Deep learning for visual understanding: A review,” *Neurocomputing,* vol. 187, no. 0925-2312, pp. 27-48, 2016.

[26] R. Vargas, A. Mosavi and R. Ruiz, “Deep learning: A review,” *Advances in Intelligent Systems and Computing,* 2017.

[27] J. Brownlee, “Supervised and Unsupervised Machine Learning Algorithms,” Machine Learning Mastery, 20 August 2020. [Online]. Available: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/. [Accessed 22 March 2022].

[28] X. Zhu and A. B. Goldberg, “Introduction to Semi Supervised Learning,” *Synthesis Lectures on Artificial Intelligence and Machine Learning,* 2009.

[29] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” towards data science, 15 December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 29 October 2021].

[30] K. O'Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” *CoRR,* vol. abs/1511.08458, 2015.

[31] Wolfram Mathworld, “Convolution,” Wolfram Alpha, 10 March 2022. [Online]. Available: https://mathworld.wolfram.com/Convolution.html#:~:text=A%20convolution%20is%20an%20integral,blends%22%20one%20function%20with%20another.. [Accessed 19 March 2022].

[32] S. Albawi, T. A. Mohammed and S. Al-Zawi, “Understanding of a convolutional neural network,” *2017 International Conference on Engineering and Technology (ICET),* pp. 1-6, 2017.

[33] MathWorks, “What is a Convolutional Neural Network?,” MathWorks, [Online]. Available: https://uk.mathworks.com/discovery/convolutional-neural-network-matlab.html. [Accessed 29 October 2021].

[34] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779-788.

[35] A. Singh, “Selecting the Right Bounding Box Using Non-Max Suppression (with implementation),” Analytics Vidhya, 3 August 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/. [Accessed 20 March 2022].

[36] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *CoRR,* vol. abs/2004.10934, 2020.

[37] V. Gupta and A. Murzova, “Keras Tutorial : Using pre-trained Imagenet models,” Learn OpenCV, 26 December 2017. [Online]. Available: https://learnopencv.com/keras-tutorial-using-pre-trained-imagenet-models/. [Accessed 21 March 2022].

[38] Y. Li, H. Wang, L. M. Dang, D. Han and H. Moon, “A Deep Learning-Based Hybrid Framework for Object Detection and Recognition in Autonomous Driving,” *IEEE Access,* vol. 8, 2020.

[39] Q. Liu, X. Fan, Z. Xi, Z. Yin and Z. Yang, “Object detection based on Yolov4-Tiny and Improved Bidirectional feature pyramid network,” *Journal of Physics: Conference Series,* vol. 2209, no. 1, p. 012023, 2022.

[40] P. Xu, Q. Li, B. Zhang, F. Wu, K. Zhao, X. Du, C. Yang and R. Zhong, “On-Board Real-Time Ship Detection in HISEA-1 SAR Images Based on CFAR and Lightweight Deep Learning,” *Remote Sensing,* vol. 13, p. 1995, 2021.

[41] techzizou, “YOLOv4 vs YOLOv4-tiny,” 24 February 2021. [Online]. Available: https://medium.com/analytics-vidhya/yolov4-vs-yolov4-tiny-97932b6ec8ec. [Accessed 21 March 2022].

[42] NVIDIA Corporation, “Your GPU Compute Capability,” NVIDIA, [Online]. Available: https://developer.nvidia.com/cuda-gpus. [Accessed 18 January 2022].

[43] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, “Microsoft COCO: Common Objects in Context,” *Computer Vision and Pattern Recognition (cs.CV),* 2014.

[44] infoaryan, “CLAHE Histogram Equalization – OpenCV,” Geeks for Geeks, 9 November 2021. [Online]. Available: https://www.geeksforgeeks.org/clahe-histogram-eqalization-opencv/. [Accessed 11 April 2022].

[45] I. H. Sarker, “Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions,” *SN Computer Science,* vol. 2, no. 6, 2021.

[46] AlexeyAB, “Yolo v4, v3 and v2 for Windows and Linux,” [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed 5 April 2022].

[47] R. Fong and A. Vedaldi, “Occlusions for Effective Data Augmentation in Image Classification,” *CoRR,* 2019.

[48] Y. Wei, J. Feng, X. Liang, M.-M. Cheng, Y. Zhao and S. Yan, “Object Region Mining with Adversarial Erasing: A Simple Classification to Semantic Segmentation Approach,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[49] K. K. Singh and Y. J. Lee, “Hide-and-Seek: Forcing a Network to be Meticulous for Weakly-supervised,” *CoRR,* 2017.

[50] T. Devries and G. W. Taylor, “Improved Regularization of Convolutional Neural Networks with Cutout,” *CoRR,* 2017.

[51] “Choosing the correct upper and lower HSV boundaries for color detection with`cv::inRange` (OpenCV),” Stack Overflow, [Online]. Available: https://stackoverflow.com/questions/10948589/choosing-the-correct-upper-and-lower-hsv-boundaries-for-color-detection-withcv. [Accessed 13 April 2022].

[52] GOV.UK, “Highways Agency warns tailgaters that 'only a fool breaks the 2-second rule',” GOV.UK, 22 May 2014. [Online]. Available: https://www.gov.uk/government/news/highways-agency-warns-tailgaters-that-only-a-fool-breaks-the-two-second-rule. [Accessed 27 April 2022].

# 9 Appendix

Area: ELECTRONICS
Sub-area: - image processing, control, software engineering;
Nature of project – mixed hardware and software design

**Project description**

Connected and autonomous vehicles (CAVs) will be the primary feature of future transportation systems [1]. Whilst CAVs will rely upon extensive networks of infrastructure external to the vehicle they will also heavily rely upon on-board systems that will allow these vehicles to acquire data about their surroundings that will improve safety and efficiency [2][3].

**Aims and Objectives**

The aim and objective of this project is to implement computer vision-based system that will identify and respond to road based markings that will provide information to a vehicle to; maintain lane position, allow a safe overtaking manoeuvre, leave the road at a junction, come to a stop at a junction etc. The objectives of this project of the above are:

1. Familiarisation with the topic of CAVs.
2. Familiarisation of the basic principles of video image processing.
3. Application of these principles to identify road-based markings, making use of freely available video clips e.g. https://www.youtube.com/watch?v=Fb9HfdCVTAo
4. Using the knowledge gained in 3 to implement a computer vision based system on a Raspberry Pi that can identify road based marking and generate instructions that may be used to control the motion of a vehicle in response to different markings.