DEEP LEARNING FOR TRACK BASED OBJECT DETECTION USING VIRTUAL ENVIRONMENTS FOR TRAINING

by

Vladislav Shishkin, 1816289

ES327: Individual Project

3rd Year

School of Engineering

University of Warwick

18th March 2021

Abstract

Object detection is a key component to creating a fully autonomous vehicle (AV), as it provides the vehicle with a significant understanding of the world. This study focuses on the scenario of a racetrack AV, where the detection targets are traffic cones, which mark path boundaries. Since little labelled data from real tracks is available, the use of virtual environments is proposed to generate data sets for training deep learning models to detect the cones. Using the Unity game engine, five different labelled image datasets of the cones were generated to perform detection efficiency tests. Various deep learning models are introduced and discussed, narrowing the selection down to the You Only Look Once (YOLO) model and introducing its detection method in detail. A number of experiments are carried out using YOLOv5 which reveal that using the virtual environments to train deep learning models is a feasible approach to real-time and real environment object detection. The most balanced model is YOLOv5 small (YOLOv5s), which reaches the detection speed of 132 frames per second (FPS), mean average precision (mAP) of 0.979 in the generalised intersection over union (GIoU) at 0.5, and mAP of 0.767 in the GIoU range of 0.5 to 0.95.

Contents

List of Figures

List of Tables

Glossary

Term	Meaning
Batch size	The number of images worked through by the learning algorithm to estimate the error gradient before the model weights are updated.
Compiler	Software that translates programming language into machine code.
Image label	A text file that contains the position and classification data of objects on the respective image.
Number of epochs	The number of times that the learning algorithm works through the entire dataset.
Object classification	Assigning an object to a category.
Object detection	Locating where an object is on an image and assigning it to a category.
Object localisation	Locating where an object is on an image.
Overfitting	Detection model fits the training dataset extremely well, however, fails to generalise to new data examples.
Prefab	Fully configured objects saved for reuse.
Texture	The visual appearance of an object.

Abbreviations

Acronym	Meaning
AI	Artificial Intelligence
ANN	Artificial Neural Network
AV	Autonomous Vehicle
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
FPS	Frames Per Second
FS	Formula Student
FS-AI	Formula Student Artificial Intelligence
GIoU	Generalised Intersection over Union
GPU	Graphics Processing Unit
IoU	Intersection over Union
mAP	mean Average Precision
ML	Machine learning
NN	Neural Network
P	Precision
R	Recall
RAM	Random Access Memory
V2V	Vehicle-to-Vehicle
YOLO	You Only Look Once
YOLOv5*n*	YOLO version 5 for model size *n* = s (small), m (medium), l (large) and x (extra-large)

1 Introduction

The autonomous vehicle (AV) technology allows cars to navigate and commute without any human interaction. It is growing fast in the industry, with companies including Tesla, Toyota, Microsoft, Samsung and many more working to make it an everyday technology within society.

There are several goals that AVs are aiming to achieve. The primary goal is to deliver higher safety to all road users. Per annum, there are approximately 5.5 million car crashes in the U.S. alone, about 2.2 million of which result in injury or death [1], and over 90% of which are results of a human as the primary cause [2]. AVs' adoption on public roads could significantly eliminate the errors made by humans as the primary cause of car accidents.

The annual cost of all car crashes to the U.S. economy comes to approximately $277 billion [3]. Further economic impacts arise from the possibilities of public car sharing and the movement of parking spaces outside of busier urban areas. Car sharing can decrease the amount of parking spaces required and allows anyone incapable of driving, such as people too old or too young to drive, to easily commute anywhere at any time. By moving a parking space from a central business district to a suburban area, the comprehensive (land, construction, maintenance and operation) costs per parking space can be decreased by up to $3,000 per year [4]. Vehicle-to-infrastructure communication can help to make smart parking decisions, avoid additional fuel consumption on "cruising for parking", as well as to enable driverless drop-offs and pickups in addition to dynamic ride sharing [5].

Another benefit that AVs can bring is the reduction of congestion on roads. Installed sensors and vehicle-to-vehicle (V2V) communication can provide adjustments of traffic flow, such as decreasing the distance between individual vehicles, setting a consistent velocity in a vehicle stream, smoothing the acceleration and the braking of all vehicles, leading to lowered fuel consumptions and lessened wear of brakes. Complete integration of sensors and V2V communication in all vehicles can increase the total capacity of highways by about 273% [6].

1.1 Problem definition

A lot of work needs to be done before AVs can infiltrate the market to the level where they reach the potentials outlined above. One of the first steps towards producing a fully equipped AV is providing it with the perception of the surrounding environment. There are three main tools used in industry to provide AVs with the perception of the real world: monocular cameras, stereo cameras and LiDARs [7]. Due to the simplicity of monocular cameras and the fact that

they already exist on most modern cars, as well as the time and funding restraints of this project, the sole focus will be on the use of monocular cameras to sense the real world. The fundamental mechanism used to provide AVs with the understanding of the real environment, in the most similar way to humans, is known as object detection. It allows AVs to perceive and navigate their environment without active human involvement. Figure 1 gives an example of an object detector in action.

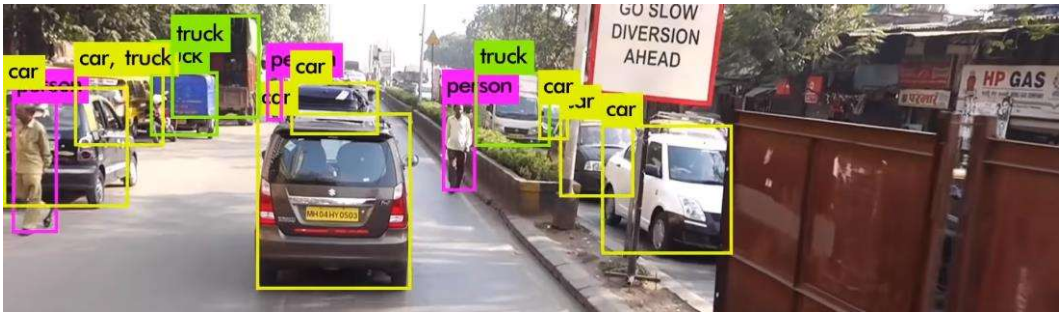


Figure 1. Object detection using YOLOv3 [8] (adapted).

The general problem definition of object detection is determining where certain objects are located on an image (object localisation) and which category every object belongs to (object classification). The most efficient approach to object detection in two dimensional images is using deep neural networks (DNNs) [9], which require supervision to learn to detect the target objects. Supervision implies manually classifying and locating each target object in an image before these images can be used for the training of an object detector. To narrow down the scope of detection targets, this project focused on object detection for track based vehicles.

1.2 Deep Learning via virtual environments

The original implications of this project were creating an object detector to be used on a Warwick Racing [10] AV, for the Formula Student – Artificial Intelligence (FS-AI) [11] competition, where the detection targets were traffic cones (orange, blue and yellow), which marked the boundaries of the Formula Student (FS) racetracks. However, manually creating a large enough dataset fell out of the scope of the project due to time and financial constraints. The closed source FSOCO dataset [12], ideally suited for the purpose of this work, could not be obtained within the time limitations of this project due to the dataset's waiting list. For these reasons, virtual environments were used to create labelled synthetic image data.

This work will focus on creating labelled synthetic images of cones in simulated environments, followed by training an object detector on these images and then evaluating its detection performance in virtual and real environments.

2 Literature Review

This chapter further explores the idea of deep learning for the object detection problem, introduced in section 1.1. Object detection is one of the key elements for creating fully functional AVs and is therefore the focus of this work. Here, DNNs are utilised to successfully detect objects. They fall into the broader concept of artificial neural networks (ANNs) which, in turn, is a part of the machine learning (ML) area of research, one of the subjects of artificial intelligence (AI). The aim of AI consists of creating systems that mimic human thinking which can adapt to changes in the environment and unfamiliar scenarios. ML focuses on a machine searching for rules and patterns in data, which it can then generalise and apply to previously unseen data. ANNs aim to mimic the function of biological neural networks (NNs). Finally, DNNs are NNs with a larger amount of hidden neuron layers, which allows the ability to learn more complicated details in the given data.

There are three main categories of object detectors [13]: feature extraction methods [14, 15], template searching methods [16, 17] and movement detection [18]. The deep learning methodology falls into the first group and was chosen as it is the most optimised and the best performing method [9]. The deeper NNs show higher performance than the shallower NNs as they can learn more complex non-linear functions with enough data, allowing them to better discriminate between different classes [19]. The primary tool of DNNs that makes them very efficient at the object detection task is the convolutional neural network (CNN), which specialises in processing data that has grid-like topology, such as images. CNNs are fully connected feed forward NNs, where each layer of neurons represents a feature map. The feature map of the input layer is a single matrix of pixel intensities for every colour channel, whereas the feature maps of internal layers are multi-channel image matrices, where every pixel corresponds to a specific feature [13]. Every neuron has a connection with a small portion of neurons from the previous layer. The working principle of CNNs is applying convolutions to patches of adjacent pixels on an image to extract features, filling the feature maps with the results of the convolutions. CNNs initially learn the abstract features of objects, such as horizontal, vertical and curved lines that outline the objects, in the first layers [20]. Then, in the deeper layers, they filter and combine the abstract features with the more complicated features, such as the smaller details of objects [21]. Pooling layers are additionally used to transform (or down sample) different features without any supervision, which allows the network to learn these features in a variety of rotations, translations and scales [22].

2.1 Object detection tools

There is a variety of different tools that have been developed for object detection, however, they can be broadly categorised into two types: two stage and single stage. The main representatives of the modern two stage family are Fast R-CNN (Region-based Convolutional Neural Network) [23] and Faster R-CNN [24]. The first stage is the generation of areas of interest on a target image using a region proposal network (RPN), which uses a CNN to predict object bounds. The second stage consists of object classification and bounding box regression in each of the predicted regions. The single stage detectors are mainly represented by YOLO (You Only Look Once) [25], SSD (Single Shot MultiBox Detector) [26] and MobileNets [27]. These models approach object detection as a regression problem by directly using the input image to learn the class probabilities and bounding box coordinates. Generally, the single stage detectors provide real-time detection, whereas the two stage models are too slow for detection in real-time. However, the two staged are more accurate at locating and classifying objects [28]. As this project aims to develop a system to be used in real-time on a racetrack, where it is extremely important to recognise objects quickly due to the fast-paced environment, a model with the single stage approach was used.

The above mentioned MobileNets, is a simple set of models, which are aimed to operate on mobile devices, as they have very low computational power requirements and work at high speed. However, this comes at a great cost to detection accuracy [29]. As this work is directed towards developing a system to be installed on a vehicle which can accommodate more powerful hardware, the models providing higher real-time detection accuracy were the preferred methods. Out of the remaining YOLO and SSD models, YOLO was chosen as overall the more efficient detector [30].

There are five main detector models that belong to the YOLO family. Chronologically, YOLOv2 [31] and YOLOv3 [32] were direct improvements of the original YOLO [25] structure along all aspects. YOLOv4 [33] then took inspiration from YOLOv3 and modified it, including the introduction of the newer CSPDarknet53 backbone [34] and the Mosaic method of data augmentation [33], to achieve improvements in both detection speed and accuracy. Lastly, the main improvement brought by YOLOv5 [35] was the transition from the Darknet [36] to the PyTorch [37] compiler, which provides much easier deployment, flexibility and productivity due to the features of PyTorch.

2.2 Detection principle of YOLOv5

The working principle behind YOLOv5 is that it treats object detection as a regression problem, obtaining the target bounding boxes and the respective confidences at once. Confidence is the probability score of an object existing in a bounding box. As shown in Figure 2, the input image is first split into SxS grids. In each grid cell, B bounding boxes (centred in the respective cell) are predicted as anchor boxes using dimension clusters [38]. Each anchor box has a respective confidence score predicted; it is then associated with the predicted C class probabilities.

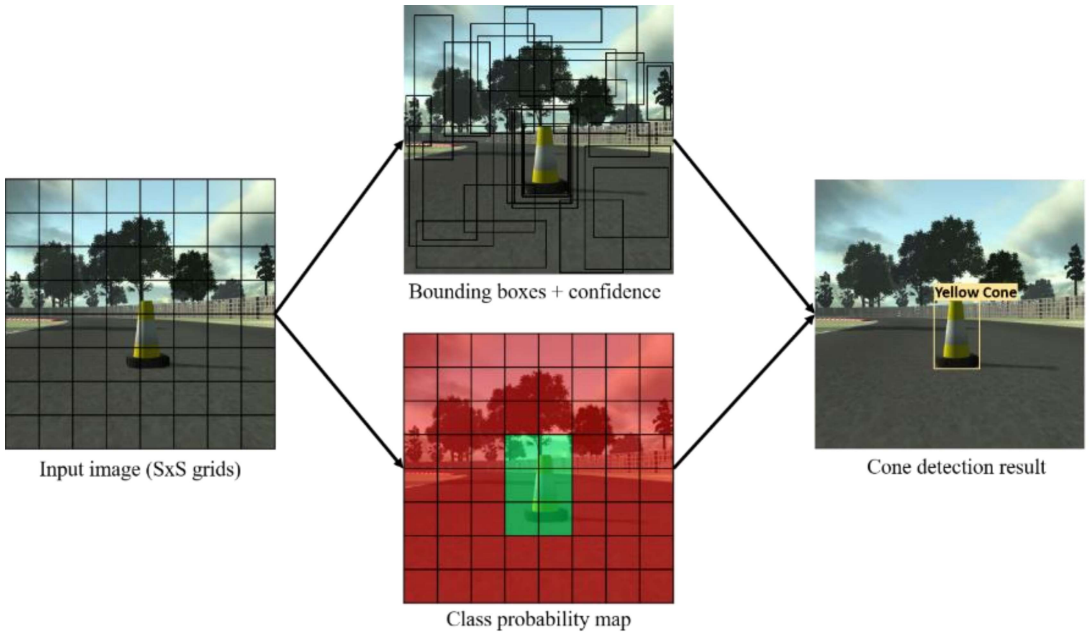


Figure 2. YOLOv5 detection method.

All predictions are therefore encoded in a SxSx[Bx(4+1+C)] tensor, where SxS represents the grid dimensions, B represents the number of anchor boxes, 4 represents the number of coordinates for each anchor box (box centre along x-axis, box centre along y-axis, box width, box height), 1 represents the confidence score for each anchor box and C represents the number of classes. The tensor therefore takes the general form of SxSx[Bx(5+C)]. YOLOv5 constructs three of these tensors for predictions of objects of different sizes: 19, 38 and 76 grids S. It then uses the generalised intersection over union (GIoU, detailed in section 3.4) technique, where only the anchor boxes that overlap with the target ground truth object (manually labelled box around a target object) by a set threshold remain. Each remaining anchor box then predicts the classes within it using binary cross-entropy loss. Lastly, to avoid detecting the same object multiple times, non-maximum suppression is used to remove redundant detections for each target object, leaving only the detection with the highest GIoU and confidence score of the correct class.

2.3 Architecture of YOLOv5

The three main building blocks of YOLO models are the backbone, which is used to extract features, the neck, which is used to collect feature maps from the different extraction stages, and the head, which is used to predict the bounding boxes and the classes. The architecture of YOLOv5 is depicted in Figure 3, where the backbone is CSPDarknet [34], the neck is PANet [39] and the head is that of YOLOv3 [32].

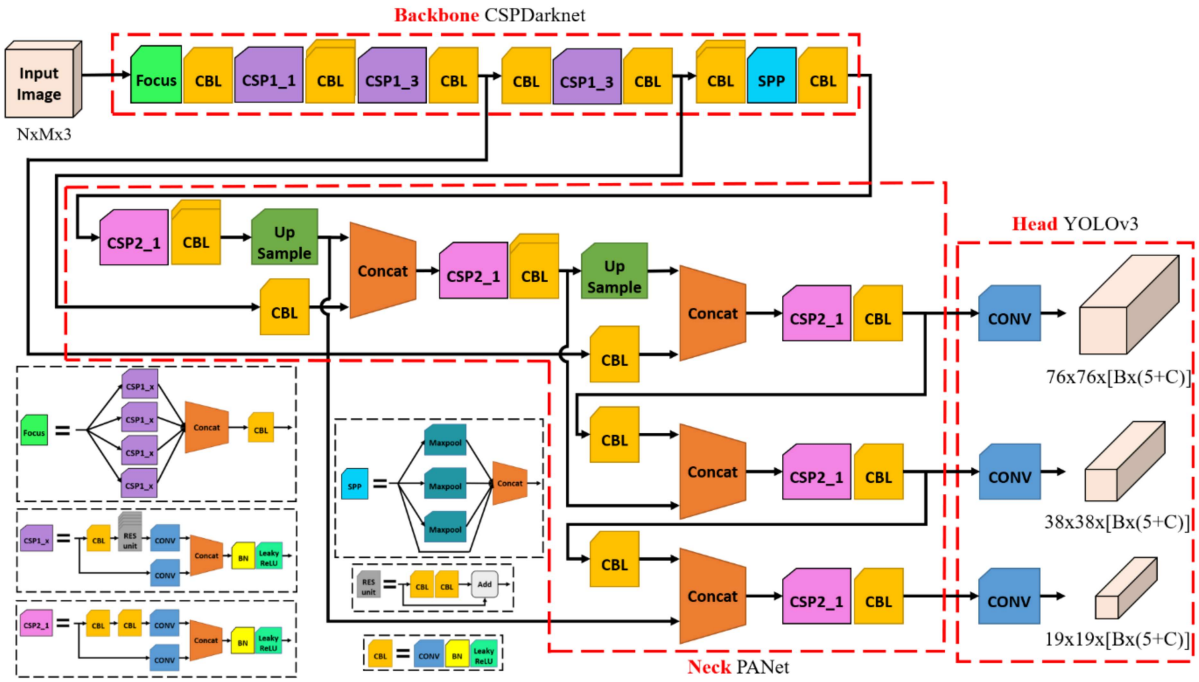


Figure 3. YOLOv5 architecture.

The input to the network is a NxMxH image tensor, where N and M are multiples of 32, NxM is the image dimensions and H is the number of channels (3 for RGB). The remainder of the network consists of several modules. The first is the focus, which reduces the resolution of the input image and increases the number of channels. This tensor reshaping achieves a reduction in the cost of further computation. The most frequently used module in YOLOv5 is the CBL, composed of a convolutional layer (CONV) [40], a batch normalisation layer (BN) [41] and a leaky-ReLU activation function [42]. The CONV layer reduces the number of parameters and therefore operations required to process an image by applying convolutions to patches of adjacent pixels. The BN layer is aimed at stabilising the learning process of the network by standardising all inputs, leading to faster training times. The purpose of an activation function is to decide whether a neuron in the network is activated or not. The up sample modules double the dimensions of the input tensors. The centre and scale prediction (CSP) [34] modules are used to divide low-level image features into two parts and then fuse cross-level features, which

enhances the learning ability of the network. The *concat* module is used for tensor stitching, where stitching AxAxB with AxAxC results in AxAx(B+C), whereas the *add* module adds the values in tensors with the same dimensions, not affecting the tensor dimensions. The residual (RES) unit is a structure aimed to make the network deeper [43]. Lastly, the spatial pyramid pooling (SPP) layer [44] removes the constraint of square input images by transforming the convolution features of different sizes into pooled features of the same dimensions. The final network outputs are the three tensors, described in more detail in section 2.2.

2.4 Summary

First, the topic of AI was introduced – explaining and narrowing the scope down towards object detection via ML, ANNs and DNNs. It was then explained how CNNs are utilised to achieve efficient object detection. Afterwards, various methods of object detection were explored. Single stage and two stage object detectors were introduced, choosing the single stage method due to the real-time detection speed and the YOLO model due to the overall efficiency. Lastly, YOLO version 5 was chosen as the most efficient out of the YOLO models, and it is among the best performing and easiest to deploy. The detection method and the structure of YOLOv5 was then explained.

3 Experimental Method

This chapter covers the approach taken to develop an object detection system. Section 3.1 explains how the object detection tool of choice was narrowed down to YOLOv5. In Section 3.2, detailed explanation of how the image data was synthesised is given. The method of labelling the cones in the synthesised images is explained in Section 3.3. Then, the metrics of measuring the performance of the trained object detection models are presented in Section 3.4. Lastly, detailed explanation of each experiment is given in Section 3.5.

3.1 Problem approach

The purpose of this work is to develop an object detection system for track based vehicles using virtual environments. An example of a working detector on a FS-AI track is shown in Figure 4.

Figure 4. FS cone detection example [45] (adapted).

The objectives of this project include synthesising a labelled image dataset, followed by training an object detection mechanism on it and then evaluating its performance.

As introduced and discussed in chapter 2, deep learning, in form of CNNs as the most prominent example [46], was chosen as the framework for the object detector, as it has proven to be the most successful method of object detection [19, 47]. CNNs provide deeper understanding of the target object's features by optimising the work of multiple tasks together, such as region selection, feature extraction and classification. The depth results in increased learning capacity and training speed of CNNs over the shallower methods [48]. The structure of CNNs also provides a solid backbone, only needing relatively small adjustments to adapt to different object detection tasks [49]. Single stage detection mechanisms are faster at detecting than two stage, providing the required real-time object detection. They also offer higher simplicity of training and deployment [50], whilst still offering high detection accuracy, overall being the method of choice for this work. YOLO has been shown to be the most efficient single stage object detection mechanism, balancing real-time detection and accuracy [51]. Improvements on

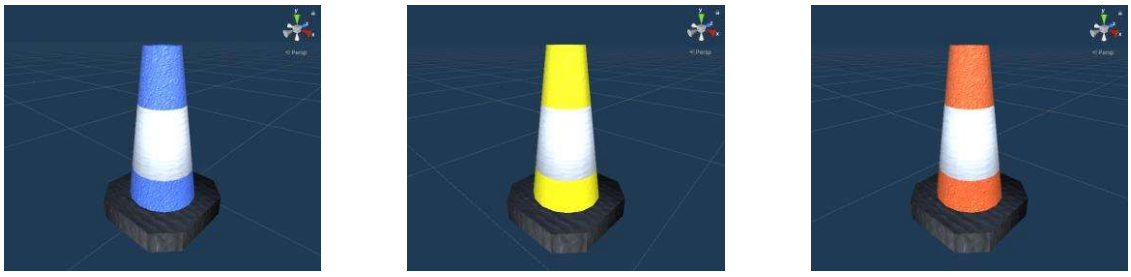
YOLO have been proposed, the latest version, YOLOv5 [35] outperforms YOLOv3 [32] in detection accuracy and detection speed, whereas it outperforms YOLOv4 [33] in detection speed only [51, 52]. Therefore, the object detection system of choice for the purpose of this work was YOLOv5 (see sections 2.2 and 2.3 for details of YOLOv5's principle of work).

3.2 Image generation

As discussed in chapter 1, object detectors require labelled images to learn to recognise objects from. It was also discussed that real world images could not be acquired and therefore, virtual environments had to be used to simulate the racetrack views from a vehicle's perspective, such as shown in Figure 4. Training and testing datasets were therefore generated to train and evaluate YOLOv5 on.

3.2.1 Unity game engine

The image generation was done in the Unity game engine. It was chosen as it was faster to learn and adapt to the task than the more complex graphical tools, such as Autodesk Maya or Blender. CARLA Simulator, based on the Unreal Engine 4 game engine, is the research standard for autonomous vehicle simulation. However, it is less optimised for random scene generation, and the PC used for the graphical simulation did not meet the minimum hardware requirements of CARLA and could not run it efficiently. The YOLOv5 requirement for the training images resolution is a multiple of 32 pixels. Following the industry standards [53], 512x512 pixels resolution was chosen as sufficiently sized images, as larger images would require too much computational power and slow down the detection, whereas objects' details would be lost on smaller images. The Unity program for generating the simulated environments was developed over time to more closely match the real view from a vehicle on a FS-AI racetrack, such as shown in Figure 4. As the intent of this project is to detect the FS-AI traffic cones used on FS-AI racetracks [54], the cones shown in Figure 5 were chosen as the detection targets (refer to Appendix I for the full list of assets used).

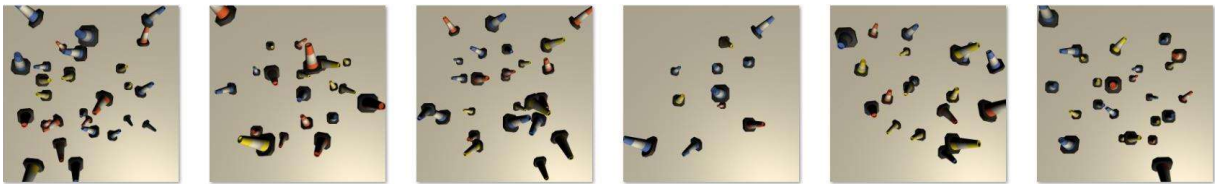


(a) (b) (c)

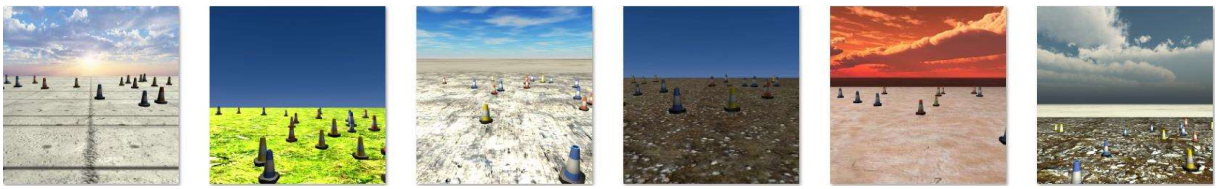
Figure 5. Detection target cones.

3.2.2 Datasets

A total of six datasets, A, B, C, D, E and F, were generated to perform various experiments on (refer to section 3.5 for experiments' details) and to evaluate the set that trains the best performing detector. Image samples of each dataset are presented in Figure 6, where the scene quality evolution is demonstrated by parts a, b, c and e.



(a) Dataset A sample.



(b) Dataset B sample.

(c) Dataset C sample.

(d) Dataset D sample.

(e) Dataset E sample.

(f) Dataset F sample.

Figure 6. Dataset samples.

The working principle of the program was randomising the training, validation and testing images as much as possible within realistic boundaries, as an object detector that has been trained on a larger variety of scenes is more robust. The generation of all datasets was done using the following steps:

1. creation of an empty scene,
2. settings randomised,
3. objects instantiated,
4. snapshot taken,
5. objects destroyed,
6. return to step 1 for the next iteration,

where step 2 was the only step that varied between the generation of different datasets. The settings randomisation followed a set of control parameters, quantified in Table 1.

Table 1. Data sets' randomisation parameters.

Dataset	No. of training images	No. of validation images	Min-max no. of cones in scene	Min-max no. of props in scene	No. of props	No. of ground textures	No. of sky textures	No. of asphalt textures	Min-max light source rotation (°)	Min-max light source intensity multiplier	Min-max camera height (Unity length units)	Min-max camera tilt (°)
A	10,000	2,000	10 – 50	0	0	1	n/a	n/a	0	1	0	0
B	10,000	2,000	5 – 30	0	0	22	37	n/a	360	0.5 – 2	5 – 10	-10 – +20
C	10,000	2,000	5 – 30	15 – 40	46	22	37	n/a	360	0.5 – 2	5 – 10	-10 – +20
D	10,000	2,000	5 – 30	15 – 40	38	16	27	n/a	360	0.5 – 2	5 – 10	-10 – +20
E	1,000	200	5 – 30	15 – 40	46	22	37	13	360	0.5 – 2	5 – 10	-10 – +20
Test dataset												
	No. of test images											
F	1,000		5 – 30	15 – 40	46	22	37	13	360	0.5 – 2	5 – 10	-10 – +20

For the simplest dataset, A, shown in Figure 6 (a), the camera was placed above the scene and there were three randomised settings. First, the cones' locations in front of a plain background, as the data was only aimed at teaching a detector what the cones look like. Secondly, the cones are rotated up to 60°, covering all possible viewing angles of the cone when it is placed on the ground. Lastly, the number of cones in each scene was altered, as a vehicle's sight on a track could be either exposed to many cones on a long straight or only a few cones in a bend.

Dataset B advanced the view of the scene to a more realistic one, as shown in Figure 6 (b), by placing the cones on the ground and the camera to view them from the side. In this and the following datasets, the cones were placed on the ground and only rotated about their vertical axis, as this better simulated the vehicle's view of them. The limits and the range of the number of cones per scene were also reduced to better reflect the real scenarios. The cones were spawned at a set maximum distance from the camera, so that their size would still be sufficiently

large for the detector to be able to extract features from them. There were six other randomisation settings, aimed at recreating various real conditions. The first two randomised settings were the textures of the ground and the sky, randomly picked from the respective predefined sets, samples of which are demonstrated in Figure 7 and Figure 8 respectively. Then, the light source was randomly rotated about its vertical axis, to simulate the different positions of the sun with respect to the camera. There was also intensity variation of the light source, simulating different times of day and weather conditions. Lastly, the camera height and tilt were randomised to positions where a real camera could be attached to a vehicle. These randomisation parameters also apply for the remaining datasets.

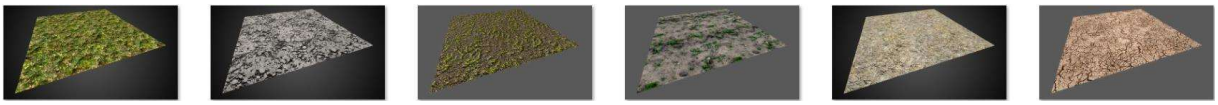


Figure 7. A sample of ground textures.

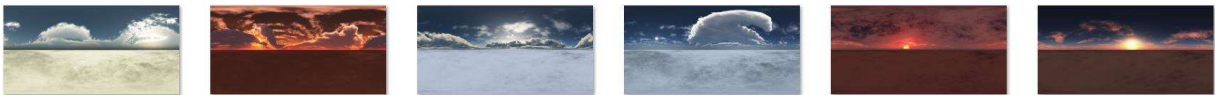


Figure 8. A sample of sky textures.

Dataset C introduced props into the scenes as shown in Figure 6 (c). It aimed at further improving the robustness of the detector by exposing it to objects other than the target cones, as there are other objects on a racetrack that can be seen by the vehicle, such as barriers, fences and trees. The randomisation process included picking a prop from a pool and spawning a random number of various props in the scene. The props' spawn locations were also randomised, with the maximum distance set to 20 Unity length units further than that of the cones. The props were randomly rotated about their respective vertical axis. A sample of various props is shown in Figure 9. These randomisation parameters also apply to the remaining datasets.

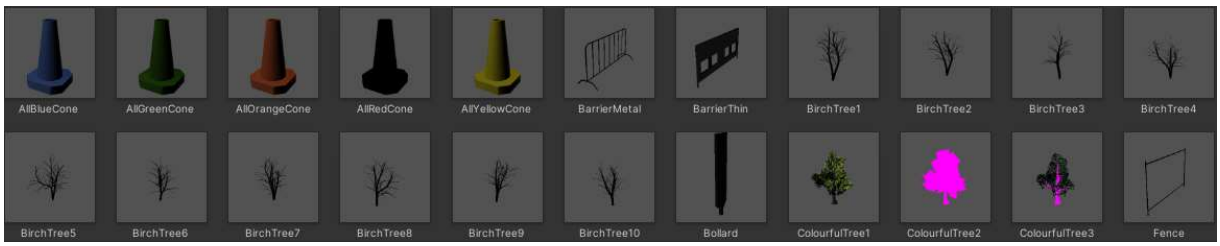


Figure 9. A sample of props.

Dataset D was a selective scenarios dataset, demonstrated in Figure 6 (d). The generation followed the exact same randomisation settings as in dataset C, however, to test the detector's performance on scenes not previously encountered, 8 props, 6 skies, and 10 ground textures

were randomly omitted from the respective pools, therefore reducing the number of possible scenarios by approximately 20%.

Dataset E introduced more organised randomisation, as shown in Figure 6 (e). To better simulate a real racetrack, a road surface was included in the scene. The texture of the road surface was randomly allocated from a pool of asphalt textures, a sample of which is demonstrated in Figure 10. The spawn locations of the cones were set to within the edges of the road surface, whereas all props were spawned outside of it.

Figure 10. A sample of asphalt textures.

The final dataset F, a sample of which is shown in Figure 6 (f), was generated with the exact same parameters as dataset E, its purpose being the test dataset (see section 3.5 for further details on experiments).

Datasets A-E each consisted of two parts, the training and the validation images. The training images were aimed at teaching a detector what the cones look like and the validation images were aimed at verifying how well the detector was performing during the training process. 10,000 images were generated to train on in each of the sets A-D, as this number of random images covers the majority of possible scenarios. Dataset E was used for transfer learning and therefore only consisted of 1,000 images, as detectors already trained on dataset C were further trained on dataset E for the more specific scenarios with a road surface. Following the industry standards [55, 51], the validation sets were 20% of the size of the training sets and the test dataset F consisted of 1,000 images. Its purpose was to be the direct comparison set of data for all detectors. See section 3.5 for more details on experimental setup.

Figure 11 demonstrates an example of an image generated for dataset F in Unity with the free camera mode view in the left camera window and the snapshot camera view in the right camera window. All objects in the scene are listed in the window to the left of the cameras, all available assets in the project are located below the cameras and the settings window of a selected item is located to the right of the cameras.

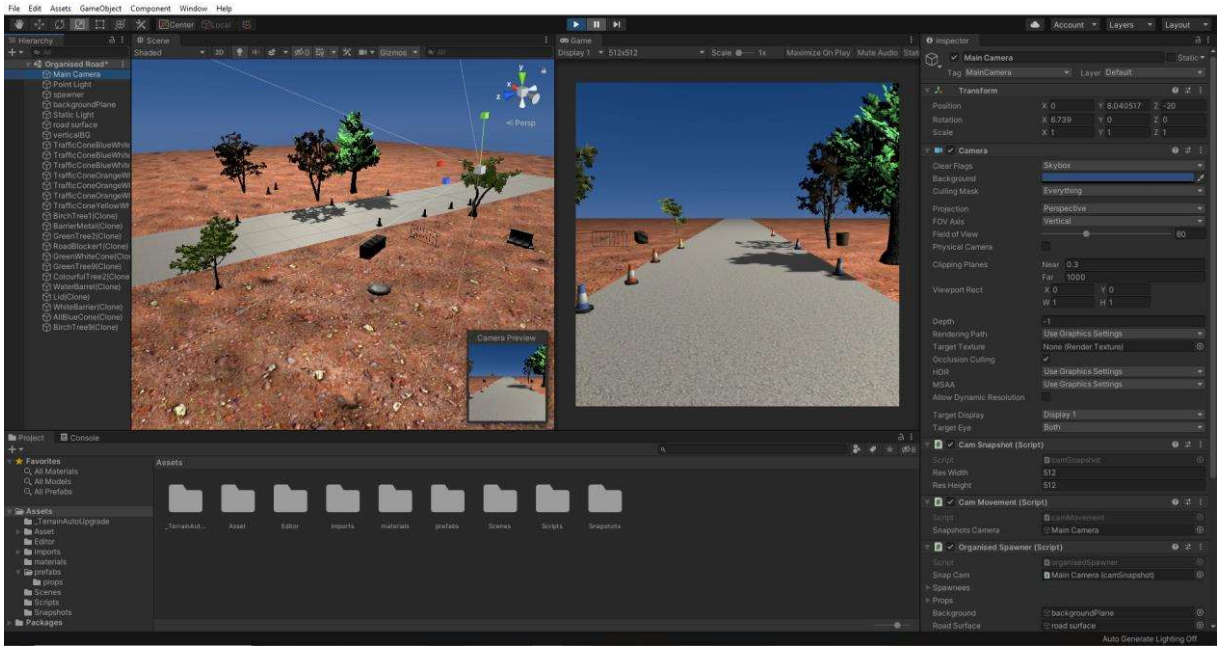


Figure 11. Scene example.

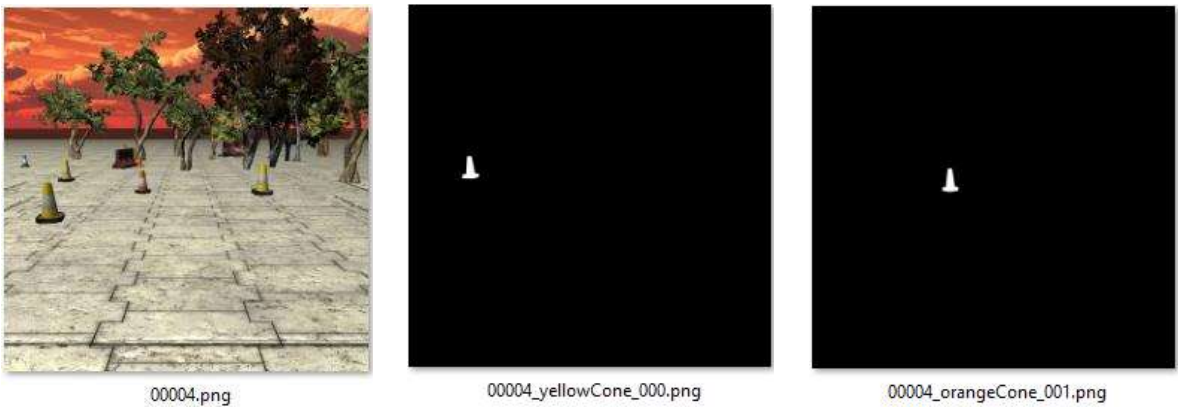
The final version of the program's source code for generating images is openly available[1].

3.3 Label generation

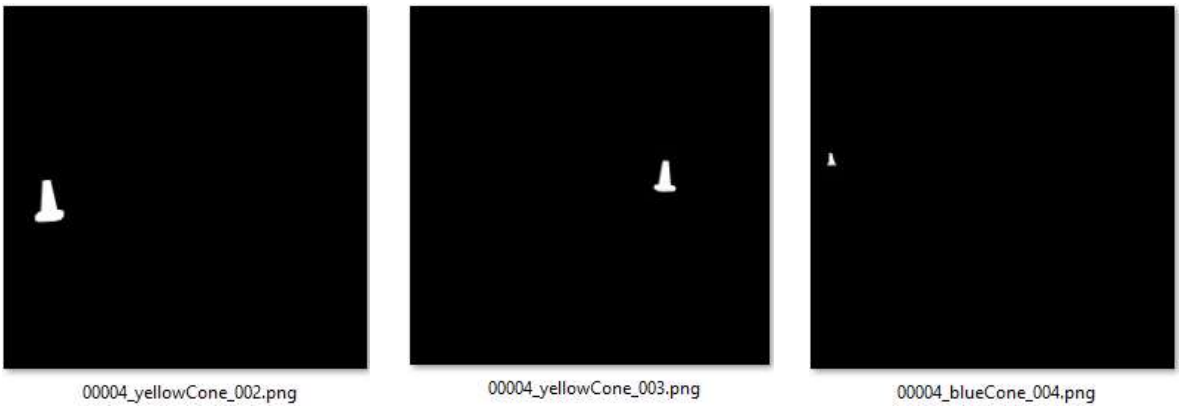
To train a CNN to detect objects, it is required that the training images have labels. An image label contains data about the class, size and location of every target object in the respective image. The challenge of labelling real world images is that the process must be completed by a human, which is time consuming, especially for large datasets of 10,000s images. However, an automated approach was developed to label the virtual cones, which consisted of the two main stages: *(i)* generating contrasting greyscale images for every cone and *(ii)* extracting the minimum and the maximum positions of every cone's pixels.

Before the instantiated objects were destroyed (refer to section 3.2 for the detailed image generation workflow description), for every cone on the synthetic image, an individual image was generated, where the colour of the cone of interest was set to grayscale 255 (white) and the colour of everything else in the camera view was set to grayscale 0 (black), as shown in Figure 12, where there are 5 cones in the original image (a) and (b-f) are the respective individual images per cone.

[1] Generation of synthetic images in Unity for cone detection source code: https://github.com/vshis/Cone-synthesis-in-Unity.



(a) Original image. (b) Cone 1. (c) Cone 2.



(d) Cone 3. (e) Cone 4. (f) Cone 5.

Figure 12. Synthetic image with 5 cones (a) and greyscale images per cone in the original image (b-f).

The labelling of the images used indexing via the filenames. As shown in Figure 12, the original image (a) is the fifth generated image (00004), where the count starts at zero (00000), and filenames of the individual cone images (b-f) contain the colour and the index of the respective cone. The filenames for individual cone images therefore take the following general form:

OriginalImageIndex_ConeColour_ConeIndex.png.

An openly available Python script was developed to process this data[2]. It recursively scanned for all ".png" files with the correct filename format in the selected directory. It then split the filenames using "_" to fetch the information about the cones in the original image. The original image index was used to define a single ".txt" file, where the label information would be stored. The cone colour was used to define the class of the cone. The cone index was used to define the individual instances of cones. YOLOv5 requires labels [56] of the following format for every target object on the image:

ClassNumber ObjectCentreAlongXAxis ObjectCentreAlongYAxis ObjectWidth ObjectHeight.

[2] YOLOv5 format label generation source code: https://github.com/vshis/YOLOv5-format-image-labelling.

The class numbers of cones were set to 0 for a blue cone, 1 for an orange cone and 2 for a yellow cone. The details of other label data are depicted in Figure 13.

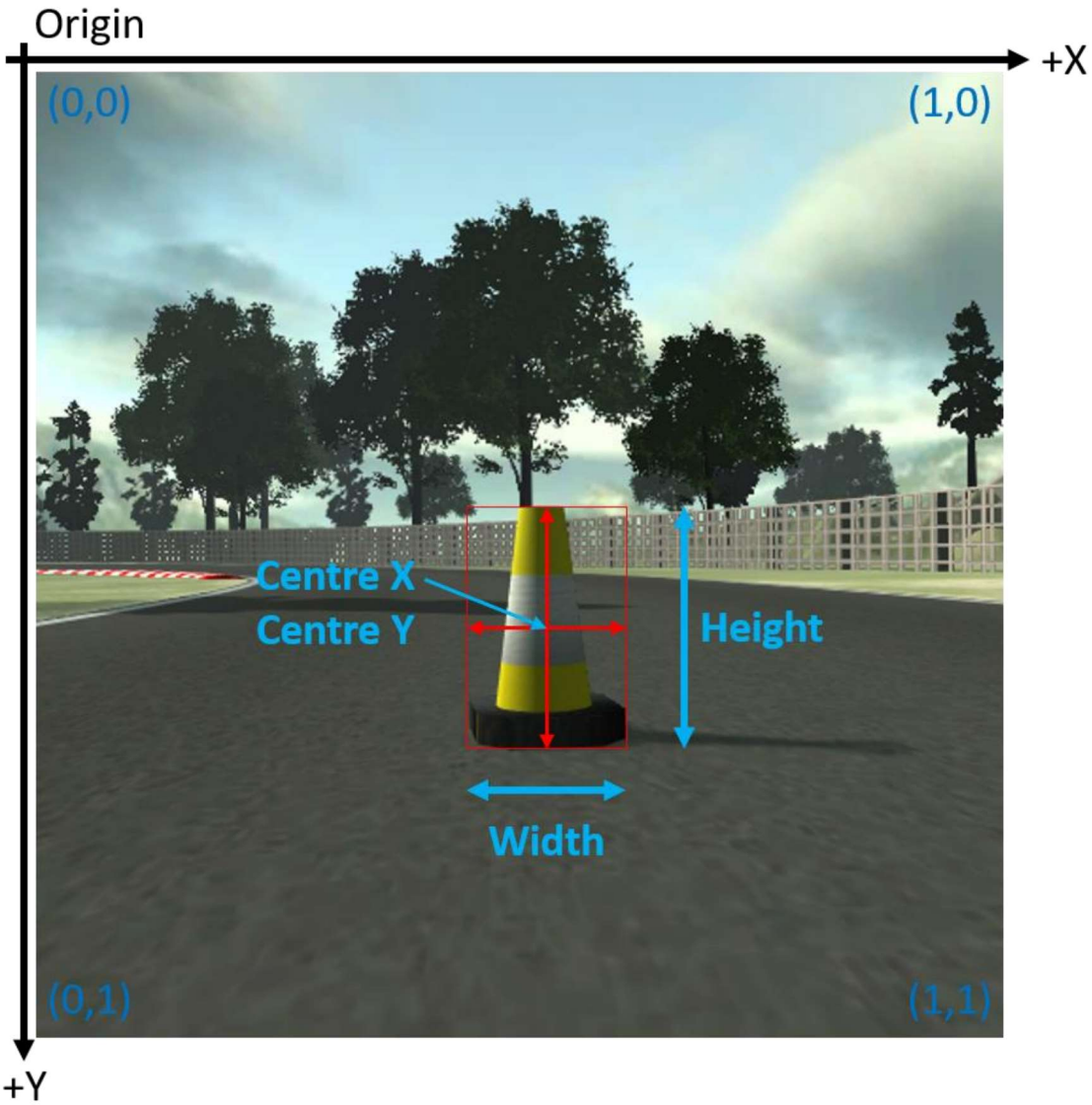


Figure 13. YOLOv5 labelling format depiction.

To achieve the desired labelling format, the script first found the minimum and the maximum positions of the white pixels on an image, along the X (X_{max}, X_{min}) and the Y (Y_{max}, Y_{min}) axes. YOLOv5's format value for the centre coordinate along the X axis is defined by Equation 1.

$$YOLOv5_{centreX} = \frac{X_{min}+X_{max}}{2*(image\ horizontal\ resolution)}. \quad (1)$$

YOLOv5's format value for the centre coordinate along the Y axis is defined by Equation 2.

$$YOLOv5_{centreY} = \frac{Y_{min}+Y_{max}}{2*(image\ vertical\ resolution)}. \quad (2)$$

YOLOv5's format value for the width is defined by Equation 3.

$$YOLOv5_{width} = \frac{X_{max} - X_{min}}{(image\ horizontal\ resolution)}. \quad (3)$$

YOLOv5's format value for the height is defined by Equation 4.

$$YOLOv5_{height} = \frac{Y_{max} - Y_{min}}{(image\ vertical\ resolution)}. \quad (4)$$

The output of the script therefore took the format demonstrated in Figure 14, where the label data is for the image in Figure 12 (a).

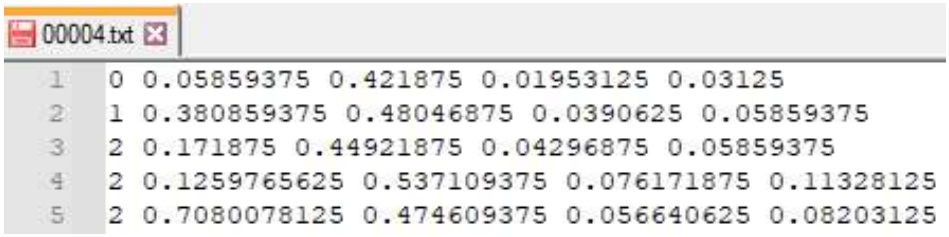


Figure 14. Example image label.

3.4 Model evaluation metrics

The fundamental indicator of performance of object detection models is the mean average precision (mAP). Precision and recall metrics are used to compute the mAP. The precision *P* is defined in Equation 5 as the proportion of correctly detected objects to the total number of detected objects, where *TP* is the number of cones correctly detected and *FP* is the number of any other regions of an image detected as cones.

$$P = \frac{TP}{FP + TP}. \quad (5)$$

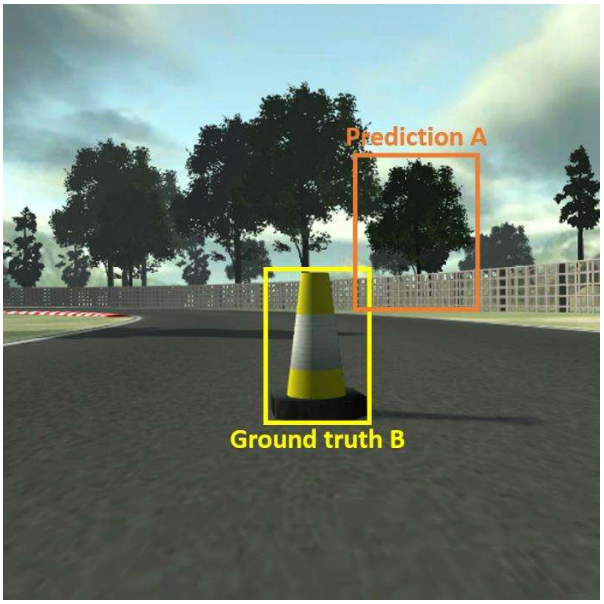
The recall *R* is defined in Equation 6 as the proportion of correctly detected objects and the total of ground truths, where *FN* is the number of cones not detected by the model.

$$P = \frac{TP}{FN + TP}. \quad (6)$$

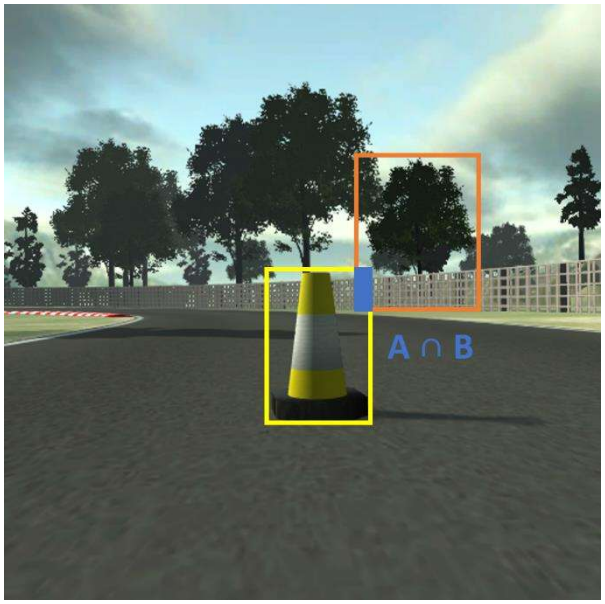
The *GIoU* is the metric for evaluating how closely the predicted bounding box *A* matches the ground truth bounding box *B*. It is defined in Equation 7.

$$GIoU = \frac{|A \cap B|}{|A \cup B|} - \frac{|C \backslash (A \cup B)|}{|C|}. \quad (7)$$

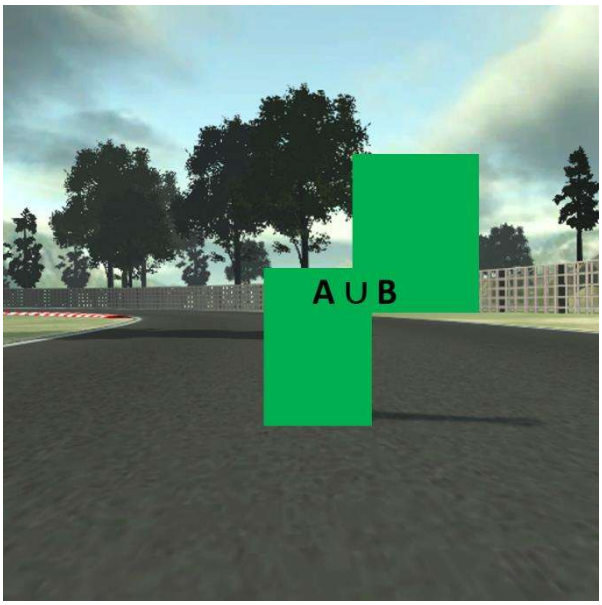
Where $A \cap B$ is the area of overlap of A and B, $A \cup B$ is the area of union of A and B, and C is the area of the smallest bounding box that encloses both A and B, as shown in Figure 15. A prediction is taken as true positive when the GIoU of this prediction is above a set threshold.



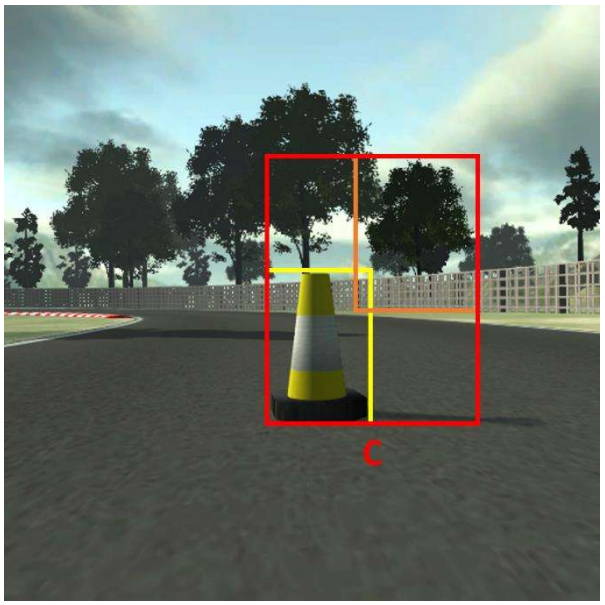
(a) Prediction and ground truth.



(b) Area of overlap of A and B.



(c) Area of union of A and B.



(d) Smallest bounding box enclosing A and B.

Figure 15. Depiction of terms in Equation 7.

The mAP is a comprehensive metric that takes into account both precision and recall. It is the mean value of the average precision AP across all classes. In the experiments performed, there were three classes – a blue cone, a yellow cone and an orange cone. AP is defined in Equation 8 as the area under the precision against recall curve.

$$AP = \int_0^1 P(R)\, dR. \tag{8}$$

The mAP metric at GIoU with a threshold of 0.5 and a second one with the threshold ranging from 0.5 to 0.95, in increments of 0.05, were the two chosen metrics for the evaluation of the results, as these are the typical standards used for object detection in the literature [43, 51, 57].

Further metrics used to evaluate the performance of the detection models were inference time and frames per second FPS. The inference time represents the time required to process a single image. The FPS represents the number of images processed per second. In general, real-time processing speed can be achieved when a model exceeds 30 FPS [58].

3.5 Experiments

All detectors were based on Python 3.8.5, PyTorch 1.7.0, YOLOv5 v4.0. All experiments were performed on a computer equipped with Intel Xeon E5-2620 CPU, 16GB DDR3 RAM and NVIDIA GeForce GTX 1080 8GB GPU.

There was a total of five experiments performed and eight detectors trained. The trainings and validations were performed on datasets A-E, whereas the numerical tests for all detectors were performed on the same dataset F, detailed in Table 1 (refer to section 3.2.2). Evaluating all models with the same test dataset ensured direct numerical performance comparison. All training cycles were run for 30 epochs. Experiments 1-4 were aimed at evaluating the training dataset that trained the best performing detector. These experiments were performed on the smallest YOLOv5 model (YOLOv5s) as it is the fastest and it demonstrates the trends of all YOLOv5 models. Following the recommended settings [56], the batch size for these experiments was set to 32. Experiment 5 then evaluated the efficiency of the full range of YOLOv5 models (YOLOv5s, YOLOv5m, YOLOv5l and YOLOv5x) trained on the best training dataset, established by the previous experiments. The training batch size for YOLOv5l and YOLOv5x were reduced to 20 and 8 respectively, due to the lack in GPU memory.

3.5.1 Experiment 1

The goal of the first experiment was to evaluate the effect of using more realistic synthetic training data on the detection performance. Three YOLOv5s models (a), (b) and (c) were trained on the three evolving sets of data, A, B and C respectively (refer to Table 1, section 3.2 for more details on the datasets). Transfer learning was adopted for all trainings, to increase the speed of training and reduce the overfitting phenomenon. As transfer learning requires

pretrained weights, *COCO weights* were used, initialised by Ultralytics' YOLOv5s detector [35] trained on the COCO dataset [59]. YOLOv5 also uses 27 hyperparameters during training. Following the recommended settings for custom data training [56], the default set of hyperparameters, *scratch hyperparameters*, was used in the trainings (refer to Appendix II for the hyperparameters' details).

3.5.2 Experiment 2

The second experiment considered the case of the network not being exposed to the same objects during the training as during the testing. A YOLOv5s model was trained on the selective scenarios dataset D. The performance of this model was compared to the model in Experiment 1 (c) due to the similarity in the training data. Transfer learning with the *COCO weights* and the *scratch hyperparameters* was again adopted.

3.5.3 Experiment 3

To improve the detection performance and decrease the possible overfitting of detectors, the YOLOv5s model in Experiment 1 (c) was further trained on the small dataset with asphalts, E, as this dataset better simulated the racetrack views from the racing car's perspective. Established by the validation images, the best performing set of weights, *best weights*, output during the training of the Experiment 1 (c) model, was used for the further transfer learning. The set of hyperparameters, *finetune hyperparameters*, aimed directly at making small changes to a trained model, were used for the training (refer to Appendix II for the hyperparameters' details).

3.5.4 Experiment 4

This experiment was aimed at qualitative analysis of the trained models to help establish the best performing model for the further evaluation. All models were given the task to detect the cones on an inference dataset G, with 15 simulated racetrack environments, an example of which is shown in Figure 16 (a) (see Appendix III for the full dataset). As shown in Figure 16 (b), a real image of a FS racetrack with the traffic cones was also included in the inference dataset due to the original implications of the project of detecting the FS-AI cones. The real image also added further validation to the use of virtual environments. The confidence of cone detection was set to 30%, as the YOLOv5 default confidence is 25% [35], which was too low for the purpose of this task.

(a) Simulated environment. (b) Real environment [60].

Figure 16. A sample of the inference dataset for qualitative detection tests.

3.5.5 Experiment 5

The purpose of the final experiment was to evaluate the performance of the different YOLOv5 models: YOLOv5s (small, 7.3 million parameters), YOLOv5m (medium, 21.4 million parameters), YOLOv5l (large, 47.0 million parameters) and YOLOv5x (extra-large, 87.7 million parameters) [35]. As Experiment 3 produced the best performing YOLOv5s detector, its training pipeline was adopted for the training of the other YOLOv5 models. The training of each model was split into two stages. Stage one consisted of training on dataset C, using the *COCO weights* and the *scratch hyperparameters*. Stage two trained each detector further on dataset E, using the *best weights* from stage one and the *finetune hyperparameters*. Due to limitations in GPU memory, the batch size was reduced to 20 for the large and 8 for the extra-large YOLOv5 models.

3.6 Summary

The goal of this work was to create an object detection system for track based vehicles. Due to the initial intentions of the project of designing an object detector for the FS-AI competition, the chosen detection targets were similar to the traffic cones used on FS tracks: orange, blue and yellow.

Virtual environments were used to synthesise labelled images of cones to train the YOLOv5 based object detectors on. To establish the best performing detector model and the best training pipeline, five experiments were performed with a total of eight models trained. Table 2 summarises the key details of each of the experiments. The main metrics of detectors' performance were the mAPs in the GIoU at 0.5 and in the range of 0.5 to 0.95, as well as the FPS.

Table 2. Summary of experiments.

Experiment	**YOLOv5 model**	**Pre trained weights**	**Hyperparameters**	**Training and validation batch size**	**Training and validation dataset**	**Summary**
1 (a)	s	*COCO*	*scratch*	32	A	Evaluation of the level of resemblance between the training and the testing images on the detector performance.
1 (b)	s	*COCO*	*scratch*	32	B	
1 (c)	s	*COCO*	*scratch*	32	C	
2	s	*COCO*	*scratch*	32	D	Evaluation of removing selective scenarios from the training data but keeping all scenarios in the test data.
3	s	1 (c) *best*	*finetune*	32	E	Evaluation of tuning the best performing model so far on the smaller dataset E.
4	n/a	n/a	n/a	n/a	n/a	Visually evaluating the performance of all trained detectors on the manually created inference dataset G.
5	s	1 (c) *best*	*finetune*	32	E	Evaluation of performance of the larger YOLOv5 models.
	m	Stage 1: *COCO* Stage 2: *best* output from stage 1	Stage 1: *scratch* Stage 2: *finetune*	32	Stage 1: C Stage 2: E	
	l	Stage 1: *COCO* Stage 2: *best* output from stage 1	Stage 1: *scratch* Stage 2: *finetune*	20	Stage 1: C Stage 2: E	
	x	Stage 1: *COCO* Stage 2: *best* output from stage 1	Stage 1: *scratch* Stage 2: *finetune*	8	Stage 1: C Stage 2: E	

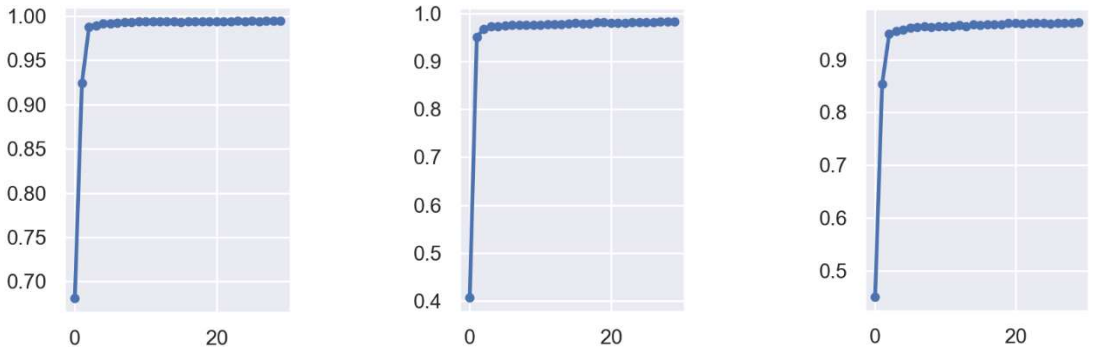
4 Results and discussions

The following section presents and analyses the results for the experiments discussed in section 3.5. It then summarises the outcome of the experiments with the best performing model.

The evaluation of the detectors was done via the validation and the testing results. Validation was performed on images generated by the same script as the training images for each model, demonstrating the performance of detectors during the training process. The numerical testing of all detectors was performed on the same dataset F, allowing for direct numerical comparison between all models.

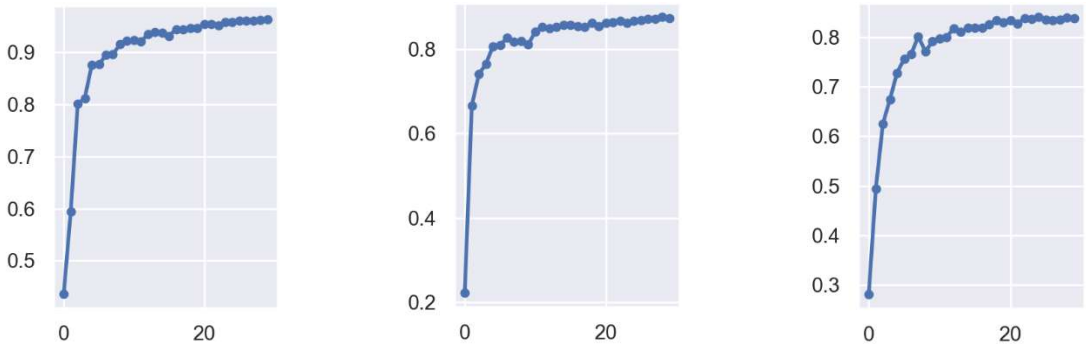
4.1 Experiment 1

Three YOLOv5s detectors (a), (b) and (c) were trained for this experiment, on datasets A, B and C respectively, aiming to evaluate the effect of resemblance of training and testing data. The validation results for mAP in the GIoU at 0.5 and in the range of 0.5 to 0.95, for the three models are presented in the respective parts of Figure 17 and Figure 18 respectively, where the y-axis represents the mAP and the x-axis represents the epoch number.



(a) Model trained on dataset A. (b) Model trained on dataset B. (c) Model trained on dataset C.

Figure 17. Experiment 1 mAP@0.5 validation results.



(a) Model trained on dataset A. (b) Model trained on dataset B. (c) Model trained on dataset C.

Figure 18. Experiment 1 mAP@0.5:0.95 validation results.

The results of the final validation are presented in Table 3.

Table 3. Experiment 1 final validation results.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95
a	0.997	0.982	0.994	0.963
b	0.996	0.961	0.982	0.874
c	0.991	0.952	0.970	0.838

Both sets of validation results demonstrate that all three models show high performance on the respective validation datasets A, B and C. The comparison of respective graphs in Figure 17 and Figure 18 shows that the mAP@0.5:0.95 increased slower than the mAP@0.5 and achieved lower final values, as shown on Table 3, which suggests that all three models suffered from overfitting. The small decrease along all four metrics from model (a) to model (c) in Table 3 was due to the increased complexity of the scenes.

The test results for the three trained models are presented in Table 4.

Table 4. Experiment 1 test results.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95	Average inference time per 512x512 image at batch size 32 (ms)	FPS
a	0.708	0.466	0.492	0.277	25.3	40
b	0.903	0.915	0.938	0.738	8.0	125
c	0.992	0.970	0.983	0.803	7.6	132

The numerical test results show that the lower exposure to various scenes of detectors (a) and (b) had a strong negative impact on real performance, shown by the drop in both mAPs from the validation results to the test results. The mAP@0.5 and mAP@0.5:0.95 of model (a) decreased by 50.5% and 71.2% respectively and of model (b) by 4.48% and 15.6% respectively. The mAPs of model (c) did not suffer from this effect between the validation and test results as the training dataset C already closely resembles the test dataset F.

As the training data resembled the test data more closely, the inference speed increased. Model (a) took the longest to complete the step of removal of redundant detections (refer to section 2.2), as it was not exposed to anything other than the cones. It therefore made more, mostly poor, predictions. The FPS increased by 213% from models (a) to (b), then further increased from (b) to (c) by 5.6%.

The best performing model across all metrics was therefore model (c), trained on dataset C.

4.2 Experiment 2

The second experiment was aimed at evaluating the performance of the YOLOv5s detector trained on the selective scenarios dataset D, where approximately 20% of all possible scenarios

were omitted from the training set. The detector was still tested on the full test dataset F. The validation results of the experiment are presented in Figure 19.

(a) mAP@0.5. (b) mAP@0.5:0.95.

Figure 19. Experiment 2 validation results.

The results of the final validation of Experiment 2 and Experiment 1 (c), for comparison, are presented in Table 5.

Table 5. Experiment 2 and Experiment 1 (c) final validation results.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Exp. 2	0.990	0.951	0.968	0.839
Exp. 1 (c)	0.991	0.952	0.970	0.838

Both sets of validation results demonstrate that the model shows high performance on the respective validation dataset D. The mAP@0.5:0.95 increased slower than the mAP@0.5, shown by Figure 19 and achieved a lower final value of 0.839, as shown by Table 5, which suggests that this model also suffered from overfitting. The variations between the results in Table 5 are minor and therefore suggest no further difference in validation results between the two models.

The test results for the selective model and the Experiment 1 (c) model, for comparison, are presented in Table 6.

Table 6. Experiment 2 and Experiment 1 (c) test results.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95	Average inference time per 512x512 image at batch size 32 (ms)	FPS
Exp. 2	0.988	0.973	0.983	0.804	7.5	133
Exp. 1 (c)	0.992	0.970	0.983	0.803	7.6	132

The comparison of the two detectors' results further confirms that no notable difference to the detector's performance was made by the removal of approximately 20% of all possible scenarios from the training.

4.3 Experiment 3

This experiment was aimed at tuning the most efficient model from the above experiments – Experiment 1 (c) to the specific scenarios of cones on road surface, using the smaller dataset E. Then evaluating the tuned model's performance against the initial Experiment 1 (c) model. The validation results for the tuned model are presented in Figure 20.

(a) mAP@0.5. (b) mAP@0.5:0.95.

Figure 20. Experiment 3 validation results.

The results of the final validation of Experiment 3 and Experiment 1 (c), for comparison, are presented in Table 7.

Table 7. Experiment 3 and Experiment 1 (c) final validation results.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Exp. 3	0.981	0.977	0.989	0.778
Exp. 1 (c)	0.991	0.952	0.970	0.838

Both sets of validation results again show high performance on the respective validation dataset E. Both mAPs converged slower than in the previous experiments. As shown by Figure 20 (a), the mAP@0.5 converged towards its limit at approximately 10th epoch as opposed to approximately 3rd epoch in previous experiments, demonstrated by Figure 17 (a, b, c) and Figure 19 (a). As can be seen from Figure 20 (b), the mAP@0.5:0.95 converged at approximately 17th epoch as opposed to approximately 10th epoch, demonstrated by Figure 18 (a, b, c) and Figure 19 (b). This suggests that the model is suffering less from overfitting. The variations between results in Table 7 are minor and therefore suggest no difference in validation results between the two models.

The test results for the tuned model and the Experiment 1 (c) model, for comparison, are presented in Table 8.

Table 8. Experiment 3 and Experiment 1 (c) test results.

Model	Precision	Recall	mAP@0.5	mAP@0.5:0.95	Average inference time per 512x512 image at batch size 32 (ms)	FPS
Exp. 3	0.975	0.964	0.979	0.767	7.6	132
Exp. 1 (c)	0.992	0.970	0.983	0.803	7.6	132

The test results further demonstrate negligible deviation between the two sets of results, further suggesting that there is no notable difference between the efficiencies of the models, other than the overfitting discussed in the analysis above.

4.4 Experiment 4

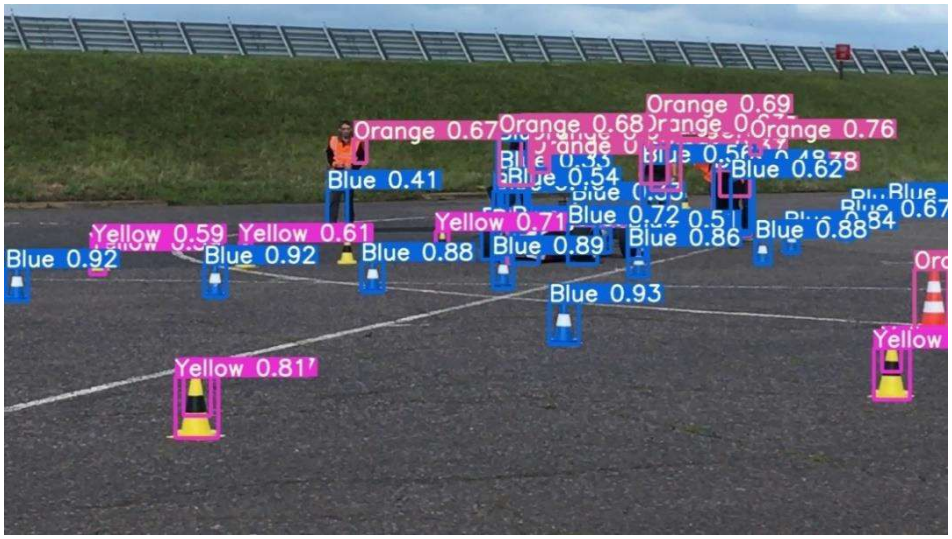
This experiment was aimed at providing visual representation of the performance of detectors trained in experiments 1-3 by running inference on 15 manually simulated and 1 real environment. Figure 21, Figure 22, Figure 23, Figure 24 and Figure 25 show the inference results of the models in experiments 1 (a), 1 (b), 1 (c), 2 and 3 respectively. Part (a) of each figure shows a sample of the inference output on the simulated environments, whereas part (b) shows the inference output on the real image (refer to Appendix IV for the full set of results).

(a) A sample of simulated environment inference results.

(b) Real environment inference result.

Figure 21. Inference output of the Experiment 1 (a) model.

(a) A sample of simulated environment inference results.



(b) Real environment inference result.

Figure 22. Inference output of the Experiment 1 (b) model.

(a) A sample of simulated environment inference results.

(b) Real environment inference result.

Figure 23. Inference output of the Experiment 1 (c) model.

(a) A sample of simulated environment inference results.

(b) Real environment inference result.

Figure 24. Inference output of the Experiment 2 model.

(a) A sample of simulated environment inference results.

(b) Real environment inference result.

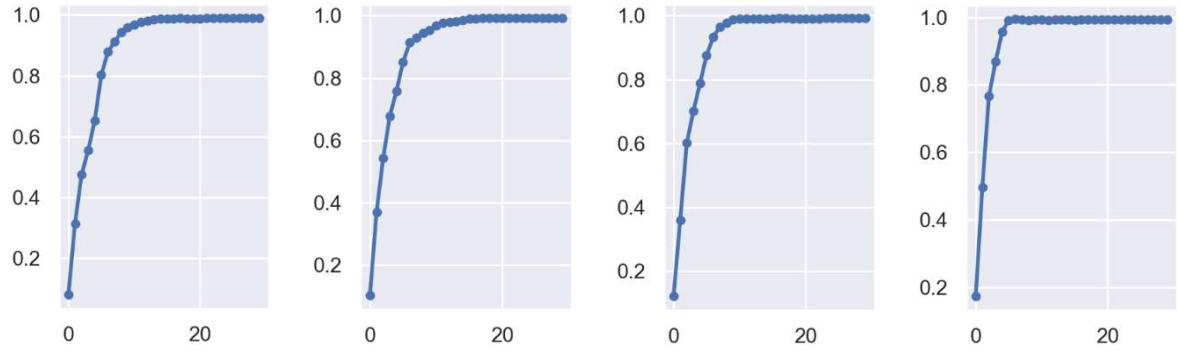
Figure 25. Inference output of the Experiment 3 model.

Parts (a) of Figure 21, Figure 22 and Figure 23 demonstrate improvement of detection quality as the training data more closely resembles the detection environments. Figure 23 and Figure 24 confirm the findings from Experiment 2, that the selective removal of scenarios did not have any impact on the detection accuracy. Visual analysis of parts (a) of each figure shows that models from experiments 1 (c), 2 and 3 show the same quality of detection on the simulated environment. Further visual analysis of parts (b) of each figure shows that the Experiment 3 model shows the best detection accuracy, as shown by Figure 25 (b), where the majority of cones on the real image have been correctly detected. The results in Figure 23 (b) also confirm the hypothesis from Experiment 3, that the Experiment 1 (c) model suffered from overfitting and that the transfer learning in Experiment 3 mitigated that, as demonstrated by Figure 25 (b).

4.5 Experiment 5

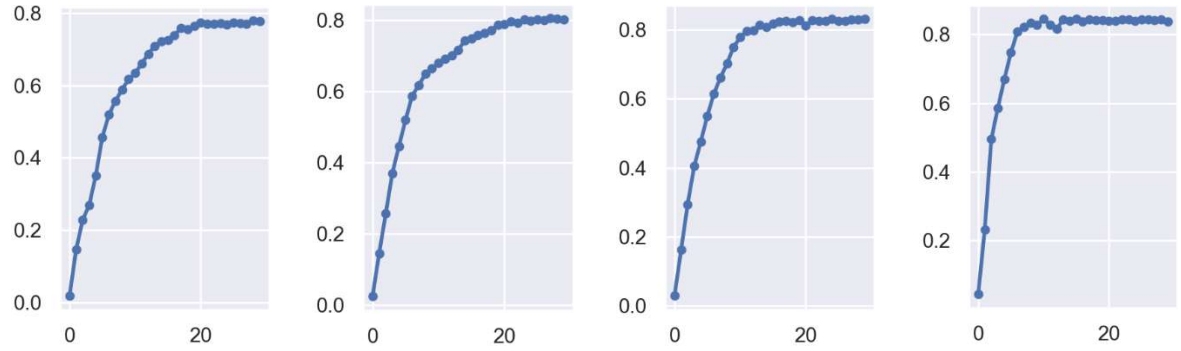
The final experiment was aimed at evaluating the performance of the other YOLOv5 models. As the Experiment 3 detector showed the best overall performance among YOLOv5s detectors, its training flow and datasets were therefore adopted for the training of the remaining YOLOv5

models. The Experiment 3 detector is referred to as the YOLOv5s model for the remainder of this analysis. Each model was first trained on dataset C, using the *COCO weights* and the *scratch hyperparameters*. They were then further trained on dataset E, using the *best* output weights from the first training and the *finetune hyperparameters*. The validation results for the small, medium, large and extra-large models are presented in parts (a), (b), (c) and (d) of Figure 26 and Figure 27 for mAP@0.5 and mAP@0.5:0.95 respectively.



(a) YOLOv5s. (b) YOLOv5m. (c) YOLOv5l. (d) YOLOv5x.

Figure 26. Experiment 5 mAP@0.5 validation results.



(a) YOLOv5s. (b) YOLOv5m. (c) YOLOv5l. (d) YOLOv5x.

Figure 27. Experiment 5 mAP@0.5:0.95 validation results.

The results of the final validation of each model are presented in Table 9.

Table 9. Experiment 5 final validation results.

YOLOv5 model	Precision	Recall	mAP@0.5	mAP@0.5:0.95
s (from Exp. 3)	0.981	0.977	0.989	0.778
m	0.995	0.975	0.991	0.802
l	0.992	0.978	0.992	0.831
x	0.998	0.986	0.993	0.838

Both sets of the validation graphs show that the larger YOLOv5 models converge towards the respective mAP limits faster. As shown by Figure 26, the mAPs@0.5 converged towards their limits at approximately 10^{th}, 9^{th}, 8^{th} and 6^{th} epochs for the models s, l, m and x respectively. Figure 27 shows the same trend with the mAPs@0.5:0.95 converging towards their limits at

approximately 17th, 18th, 12th and 7th epochs for the models s, l, m and x respectively. This suggests that the larger models suffer more from overfitting. Table 9 shows that the difference between the final validation precision, recall and mAP@0.5 across all models is negligible, whereas the mAP@0.5:0.95 increased gradually from model s to model x by 7.71%, suggesting only small overall improvements in detection performance.

The test results for the different YOLOv5 models are presented in Table 10.

Table 10. Experiment 5 test results.

YOLOv5 model	Precision	Recall	mAP@0.5	mAP@0.5:0.95	Average inference time per 512x512 image at batch size 32 (ms)	FPS
s (from Exp. 3)	0.975	0.964	0.979	0.767	7.6	132
m	0.984	0.971	0.983	0.789	15.5	65
l	0.987	0.972	0.984	0.819	25.1	40
x	0.990	0.972	0.984	0.832	43.7	23

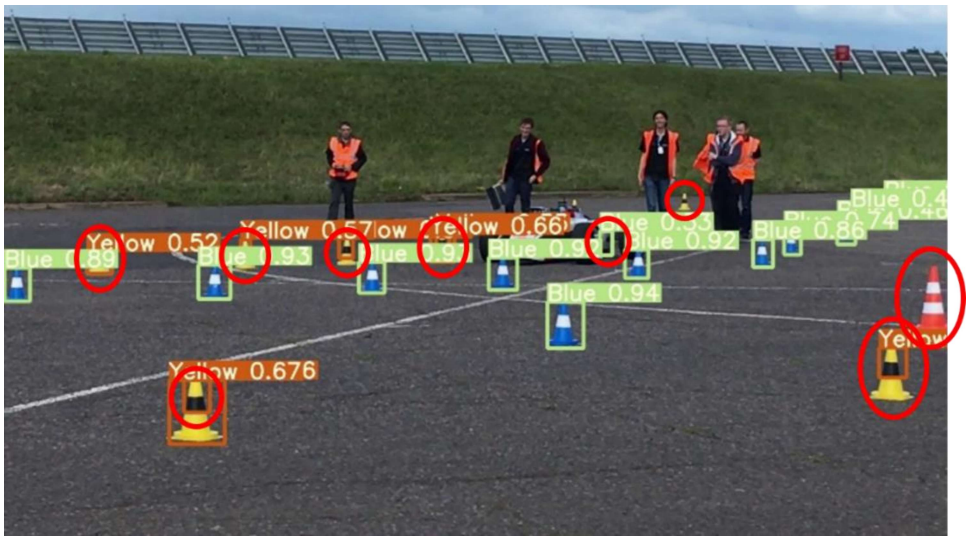
The results demonstrate that consecutively larger models did not gain any increase in mAP@0.5, however, achieved higher results in mAP@0.5:0.95 at the cost of inference speed. The FPS of the extra-large model fell short of real-time detection (30 FPS). The mAP@0.5:0.95 increased from the small to medium models by 2.87%, while the FPS decreased by 50.8%. The increase in mAP@0.5:0.95 from the medium to large models was by 3.80%, with the FPS decreasing further by 38.5%. The gain in mAP@0.5:0.95 from the large to extra-large models was by 1.59% at the cost of FPS decreasing further by 42.5%. To visualise the gain in detection performance, the newly trained m, l and x models were given the Experiment 4 task of detecting cones in the manually simulated and the real environments. Figure 28, Figure 29, Figure 30 and Figure 31 show the inference results of the YOLOv5 s, m, l and x models respectively. Part (a) of each of the figures demonstrates a sample of the inference output of simulated environments and part (b) demonstrates the real environment's inference output (see Appendix IV for the complete set of results), where the areas circled in red highlight any detection errors. A detection error was a model either not detecting a cone where a cone was present or detecting a cone where a cone was not present. Partial detections were also taken as detection errors.

(a) A sample of simulated environment inference results.

(b) Real environment inference result.

Figure 28. YOLOv5s inference output.

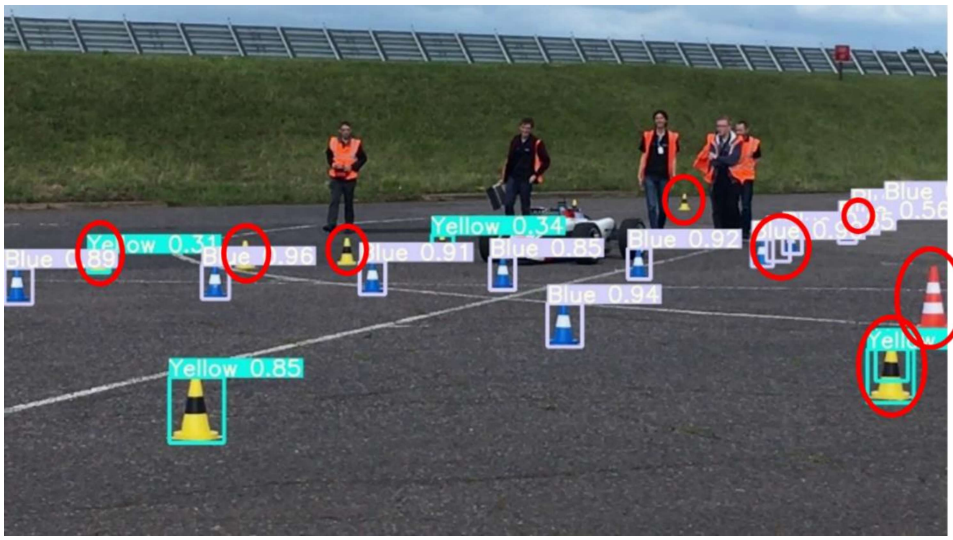
(a) A sample of simulated environment inference results.



(b) Real environment inference result.

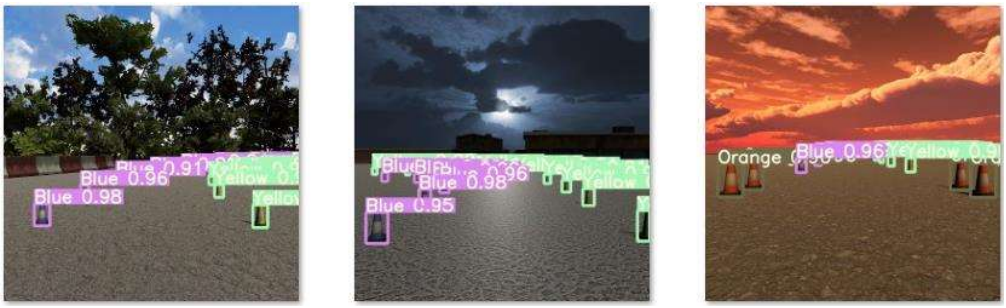
Figure 29. YOLOv5m inference output.

(a) A sample of simulated environment inference results.

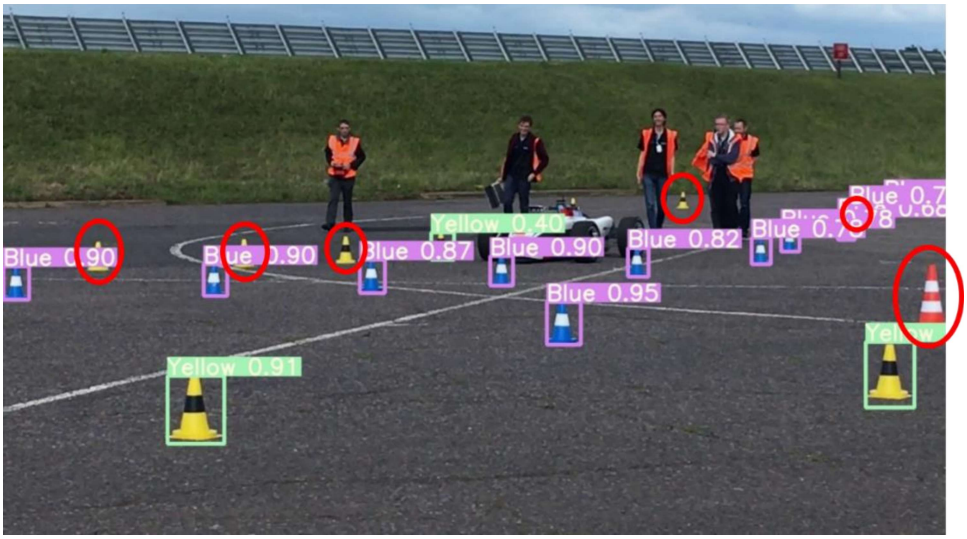


(b) Real environment inference result.

Figure 30. YOLOv5l inference output.



(a) A sample of simulated environment inference results.



(b) Real environment inference result.

Figure 31. YOLOv5x inference output.

The visual analysis of parts (a) of each figure shows that all models perform exceptionally well in the simulated environments. The visual analysis of parts (b) confirms that the larger models suffered more from overfitting, as model l did not detect all and model x did not detect any of the three yellow cones in the top left quadrant of the image. The number of errors for the small, medium large and extra-large models are 6, 9, 8 and 6 respectively. As model x did not achieve real-time detection speed, models m and l made more errors on the real image and showed only marginal gain in mAP@0.5:0.95 and no gain in mAP@0.5 at great costs of FPS, model s was concluded to be the most efficient detector.

4.6 Summary

All models have demonstrated exceptional results on the respective validation datasets. However, analysing the numerical test results and the qualitative test results, the best performing model was established to be YOLOv5s, first trained on dataset C, then tuned with transfer learning to the specific scenario, on dataset E. The detector's mAPs in the GIoU at 0.5 and in the range of 0.5 to 0.95 were 0.979 and 0.767 respectively, with the detection speed of 132 FPS. This model also demonstrated the best performance on the inference dataset.

5 Conclusions

Aiming to create an object detection system for an AV, this project first looks into what object detection is and explores the various ways that efficient localisation and classification of objects can be achieved. It then focuses on object detection for track-based vehicles, using traffic cones as detection targets and YOLO version 5 as the detection tool. Virtual environments are then adopted to generate labelled image datasets using the Unity game engine.

The two steps of labelled synthetic image data construction are introduced in detail. Five different training datasets were generated and the detection performances from various trainings was evaluated. Lastly, the four YOLOv5 models (small, medium, large and extra-large) were systematically compared. Based on the testing and the detection results, several conclusions can be summarised as follows.

1. The training methodology that produced the best detection results consisted of two steps. First, training on a large dataset of randomised cones, props and backgrounds. Second, tuning the trained model with transfer learning on a smaller dataset, where the randomisation is more controlled by placing the cones in a track grid on an asphalted surface, whilst other props are placed outside of the asphalt.
2. All YOLOv5 models trained in the above manner could automatically localise and classify cones in the simulated and the real environments.
3. The highest overall detection performance was shown by the smallest YOLOv5 model (YOLOv5s). The mAPs of this model were marginally lower than of the larger YOLOv5 models. However, it showed the highest detection speed and the best detection performance in the real environment.
4. YOLOv5s' mAPs in the GIoU at 0.5 and in the range of 0.5 to 0.95 were 0.979 and 0.767 respectively, with the detection speed of 132 FPS.

5.1 Recommendations for further work

The potential improvements of this system could include the following.

1. To achieve better detection performance on FS-AI racetracks, the virtual models of the traffic cones can be modified to more closely resemble those used by FS.
2. Adding more randomisation effects when generating the images. These include varying image contrast, blur and sharpness, as well as adding the effects of rain or fog.

3. Gaining an access to the FSOCO dataset would provide the real-world images of the FS-AI racetracks, which can then be used to train detectors that perform much better in the real environment.
4. Lastly, researching and proposing the computer hardware configuration to be installed on the FS AV, which will provide efficient cone detection at the minimum power requirements.

5.2 Final remarks

Constraining the task of object detection for AVs to racetrack environments resulted in narrowing of the project's scope down to detecting various traffic cones. Virtual environments were generated in the Unity game engine to synthesise labelled images of the cones in different settings. These images were then used to train a YOLOv5 based detector to successfully localise and classify the cones in the real environment.

7 References

[1] National Highway Traffic Safety Administration, “Traffic Safety Facts,” U.S. Department of Transportation, Washington D.C., 2013.

[2] National Highway Traffic Safety Administration, “National Motor Vehicle Crash Causation Survey,” U.S. Department of Transportation, 2008.

[3] B. Lawrence, T. Miller, Z. Eduard and B. Lawrence, “The Economic and Societal Impact of Motor Vehicle Crashes, 2010 (Revised),” U.S. Department of Transportation, Washington D.C., 2014.

[4] D. J. Fagnant and K. Kockelman, “Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations,” *Transportation Research Part A: Policy and Practice,* vol. 77, pp. 167-181, 2015.

[5] N. Agatz, A. Erera, M. Savelsbergh and X. Wang, “Optimization for dynamic ride-sharing: A review,” *European Journal of Operational Research,* vol. 223, no. 2, pp. 295-303, 2012.

[6] P. Tientrakool, Y.-C. Ho and N. F. Maxemchuk, “Highway Capacity Benefits from Using Vehicle-to-Vehicle Communication and Sensors for Collision Avoidance,” in *2011 IEEE Vehicular Technology Conference (VTC Fall)*, San Francisco , 2011.

[7] J. Kabzan, M. d. l. I. Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari and S. Brits, “AMZ Driverless: The Full Autonomous Racing System,” *Journal of Field Robotics,* vol. 37, no. 7, pp. 1267-1294, 2020.

[8] J. Redmon, “YOLOv3,” YouTube, 25 March 2018. [Online]. Available: https://www.youtube.com/watch?v=MPU2HistivI. [Accessed 10 March 2021].

[9] C. Szegedy, A. Toshev and D. Erhan, “Deep Neural Networks for Object Detection,” in *26th International Conference on Neural Information Processing Systems - Volume 2*, 2013.

[10] Warwick Racing, “Warwick Racing Home Page,” University Of Warwick, 2021. [Online]. Available: https://warwickracing.org/. [Accessed 11 March 2021].

[11] FORMULA STUDENT, “FS-AI - FORMULA STUDENT ARTIFICIAL INTELLIGENCE,” Institution of Mechanical Engineers, [Online]. Available: https://www.imeche.org/events/formula-student/team-information/fs-ai. [Accessed 13 March 2021].

[12] D. Dodel, M. Schötz and N. Vödisch, “FSOCO: The Formula Student Objects in Context Dataset,” 2020.

[13] P. N. Druzhkov and V. D. Kustikova, “A Survey of Deep Learning Methods and Software Tools,” Nizhni Novgorod, 2016.

[14] J. Shotton, A. Blake and R. Cipolla, “Contour-based learning for object detection,” in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, Beijing, 2005.

[15] P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan, “Object Detection with Discriminatively Trained Part-Based Models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 32, no. 9, pp. 1627-1645, 2009.

[16] C. Hilario, J. Collado, J. Armingol and A. d. l. Escalera, “Pyramidal image analysis for vehicle detection,” in *Intelligent Vehicles Symposium*, Las Vegas, 2005.

[17] Y. Amit, 2D Object Detection and Recognition: Models, Algorithms, and Networks, MIT Press, 2002.

[18] M. Sonka, V. Hlavac and R. Boyle, Image Processing, Analysis and Machine Vision, Springer US, 1993.

[19] A. Krizhevsky, I. Sutskever and G. Hinton, “Imagenet classification with deep convolutional,” in *Advances in Neural Information Processing Systems 25*, 2012.

[20] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs and H. Lipson, “Understanding Neural Networks Through Deep Visualization,” 2015.

[21] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” 2013.

[22] M. Jaderberg, K. Simonyan, A. Zisserman and K. Kavukcuoglu, “Spatial Transformer Networks,” 2016.

[23] R. Girshick, “Fast R-CNN,” in *IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, 2015.

[24] S. Ren, K. He, R. Girshick and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” in *Advances in Neural Information Processing Systems 28*, 2015.

[25] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” 2016.

[26] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, “SSD: Single Shot MultiBox Detector,” 2016.

[27] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017.

[28] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama and K. Murphy, “Speed/accuracy trade-offs for modern convolutional object detectors,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[29] P. Soviany and R. T. Ionescu, “Optimizing the Trade-off between Single-Stage and Two-Stage Object Detectors using Image Difficulty Prediction,” 2018.

[30] R. Deepa, E. Tamilselvan, E. Abrar and S. Sampath, “Comparison of Yolo, SSD, Faster RCNN for Real Time Tennis Ball Tracking for Action Decision Networks,” in *International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Sathyamangalam, India, 2019.

[31] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” 2016.

[32] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” 2018.

[33] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” 2020.

[34] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen and J.-W. Hsieh, “CSPNet: A New Backbone that can Enhance Learning Capability of CNN,” in *IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop)*, 2020.

[35] G. Jocher, “YOLOv5,” Ultralytics, 5 January 2021. [Online]. Available: https://github.com/ultralytics/yolov5. [Accessed 21 February 2021].

[36] J. Redmon, “darknet,” GitHub, 10 December 2020. [Online]. Available: https://github.com/pjreddie/darknet. [Accessed 11 March 2021].

[37] A. Paszke, S. Gross, S. Chintala and G. Chanan, “PyTorch Home Page,” Facebook's AI Research lab (FAIR), [Online]. Available: https://pytorch.org/. [Accessed 11 March 2021].

[38] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, “Focal Loss for Dense Object Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 42, no. 2, pp. 318-327, 2018.

[39] S. Liu, L. Qi, H. Qin, J. Shi and J. Jia, “Path Aggregation Network for Instance Segmentation,” 2018.

[40] I. Goodfellow, Y. Bengio and A. Courville, “Convolutional Networks,” in *Deep Learning*, MIT Press, 2016, pp. 326-339.

[41] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” in *32nd International Conference on Machine Learning*, 2015.

[42] X. Zhang, Y. Zou and W. Shi, “Dilated convolution neural network with LeakyReLU for environmental sound classification,” in *22nd International Conference on Digital Signal Processing (DSP)*, London, 2017.

[43] S. Li, X. Gu, X. Xu, D. Xu, T. Zhang, Z. Liu and Q. Dong, “Detection of concealed cracks from ground penetrating radar images based on deep learning algorithm,” *Construction and Building Materials,* vol. 273, 2021.

[44] K. He, X. Zhang, S. Ren and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” 2015.

[45] AMZFormulaStudent, “AMZ Driverless: The Full Autonomous Racing System,” 15 May 2019. [Online]. Available: https://www.youtube.com/watch?app=desktop&v=hYpToBIMpVQ. [Accessed 4 March 2021].

[46] Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series,” in *The handbook of brain theory and neural networks*, Cambridge, U.S., MIT Press, 1998, pp. 255-258.

[47] Q. V. Le, “Building high-level features using large scale unsupervised learning,” in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, 2013.

[48] Y. Bengio, “Learning Deep Architectures for AI,” in *Foundations and Trends in Machine Learning*, 2009, pp. 1-127.

[49] Z.-Q. Zhao, P. Zheng, S.-T. Xu and X. Wu, “Object Detection With Deep Learning: A Review,” *IEEE Transactions on Neural Networks and Learning Systems,* vol. 30, no. 11, pp. 3212 - 3232, 2019.

[50] Y. Chen, C. Han, N. Wang and Z. Zhang, “Revisiting Feature Alignment for One-stage Object Detection,” arXiv:1908.01570, 2019.

[51] E. Badeka, T. Kalampokas, E. Vrochidou, K. Tziridis, G. Papakostas, T. Pachidis and V. Kaburlasos, “Real-time vineyard trunk detection for a grapes harvesting robot via deep learning,” in *Thirteenth International Conference on Machine Vision*, Rome, 2021.

[52] Y. Chen, C. Zhang, T. Qiao, J. Xiong and B. Liu, “Ship detection in optical sensing images based on YOLOv5,” in *Twelfth International Conference on Graphics and Image Processing (ICGIP 2020)*, Xi'an, 2021.

[53] W. Li, W. Li, F. Yang and P. Wang, “Multi-Scale Object Detection in Satellite Imagery Based On YOLT,” in *IGARSS IEEE International Geoscience and Remote Sensing Symposium*, Yokohama, 2019.

[54] Institution of Mechanical Engineers, “FS-AI Dynamic Events Setup and Cones Specification,” 7 March 2019. [Online]. Available: https://www.imeche.org/events/formula-student/team-information/forms-and-documents. [Accessed 4 March 2021].

[55] Y. Liu, B. Lu, J. Peng and Z. Zhang, “Research on the Use of YOLOv5 Object Detection Algorithm in Mask Wearing Recognition,” *World Scientific Research Journal,* vol. 6, no. 11, 2020.

[56] G. Jocher, “YOLOv5 - Train Custom Data,” Ultralytics, 27 February 2021. [Online]. Available: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data. [Accessed 7 March 2021].

[57] F. Zhou, H. Zhao and Z. Nie, “Safety Helmet Detection Based on YOLOv5,” in *IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*, Shenyang, 2021.

[58] H. Ma, Y. Liu, Y. Ren and J. Yu, “Detection of Collapsed Buildings in Post-Earthquake Remote Sensing Images Based on the Improved YOLOv3,” *Remote Sensing,* vol. 12, no. 1, p. 44, 2020.

[59] 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,* 2015.

[60] Formula Student UK, “Formula Student AI testing ahead of FS2019,” 2019. [Online]. Available: https://www.youtube.com/watch?v=Mt-aNqZqVmg. [Accessed 21 02 2021].

Appendix I

The full list of Unity assets used from the Unity Asset Store:

- Traffic cones - https://assetstore.unity.com/packages/3d/props/traffic-cones-34912
- Ground materials - https://assetstore.unity.com/packages/2d/textures-materials/floors/yughues-free-ground-materials-13001
- Concrete materials - https://assetstore.unity.com/packages/2d/textures-materials/concrete/yughues-free-concrete-materials-12951#content
- Asphalt materials - https://assetstore.unity.com/packages/2d/textures-materials/roads/asphalt-materials-141036
- Sky textures - https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014, https://assetstore.unity.com/packages/2d/textures-materials/sky/8k-skybox-pack-free-150926
- Trees - https://assetstore.unity.com/packages/3d/vegetation/trees/birch-tree-pack-vol-1-49093, https://assetstore.unity.com/packages/3d/vegetation/trees/realistic-tree-9-rainbow-tree-54622, https://assetstore.unity.com/packages/3d/vegetation/trees/realistic-tree-10-54724, https://assetstore.unity.com/packages/vfx/shaders/world-space-trees-free-shader-117088
- White/Red Barriers - https://assetstore.unity.com/packages/3d/props/exterior/old-plastic-barrier-182509
- Various road props - https://assetstore.unity.com/packages/3d/props/exterior/road-props-low-poly-123340
- Road blockers - https://assetstore.unity.com/packages/3d/props/exterior/road-blocker-663
- Industrial buildings - https://assetstore.unity.com/packages/3d/building-shed-1042, https://assetstore.unity.com/packages/3d/environments/industrial/storage-building-50430, https://assetstore.unity.com/packages/3d/environments/industrial/free-industrial-building-garage-123146
- Cars - https://assetstore.unity.com/packages/3d/vehicles/land/hq-racing-car-model-no-1203-139221, https://assetstore.unity.com/packages/3d/vehicles/low-poly-car-149312, https://assetstore.unity.com/packages/3d/vehicles/land/low-poly-sports-car-20-144253, https://assetstore.unity.com/packages/3d/vehicles/land/modern-touring-car-pack-demo-132820, https://assetstore.unity.com/packages/3d/vehicles/land/touring-race-car-pack-demo-130064
- Racetracks - https://assetstore.unity.com/packages/3d/environments/roadways/race-tracks-140501

Appendix II

Table 11 demonstrates the default set of hyperparameters used, where the hyperparameters are unspecified, *scratch* and the set of hyperparameters aimed at adjusting models to specific training cases, *finetune* [35].

Table 11. Hyperparameter values.

Model / Parameter	scratch	finetune
lr0	0.01	0.0032
lrf	0.2	0.12
momentum	0.937	0.843
weight_decay	0.0005	0.00036
warmup_epochs	3.0	2.0
warmup_momentum	0.8	0.5
warmup_bias_lr	0.1	0.05
box	0.05	0.0296
cls	0.5	0.243
cls_pw	1.0	0.631
obj	1.0	0.301
obj_pw	1.0	0.911
iou_t	0.2	0.2
anchor_t	4.0	2.91
fl_gamma	0.0	0.0
hsv_h	0.015	0.0138
hsv_s	0.7	0.664
hsv_v	0.4	0.464
degrees	0.0	0.373
translate	0.1	0.245
scale	0.5	0.898
shear	0.0	0.602
perspective	0.0	0.0
flipud	0.0	0.00856
fliplr	0.5	0.5
mosaic	1.0	1.0
mixup	0.0	0.243

Appendix III

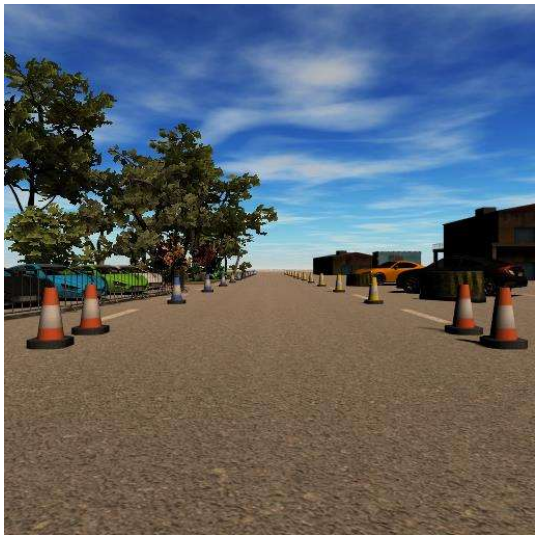
The full inference dataset is presented in Figure 32, where (a) is the real image and (b-p) are synthetic images manually created in Unity.

(a)

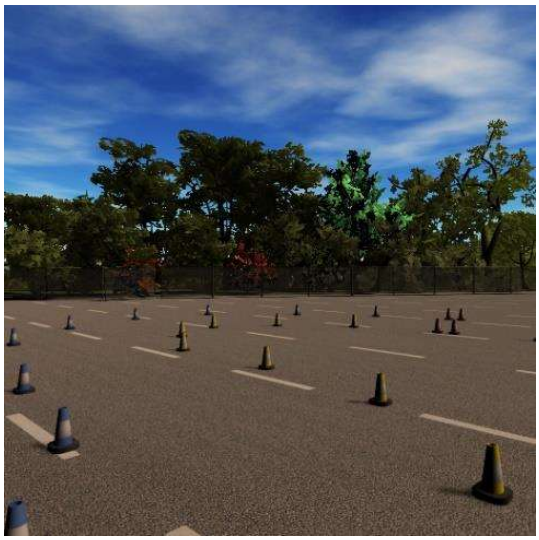
(b)

(c)

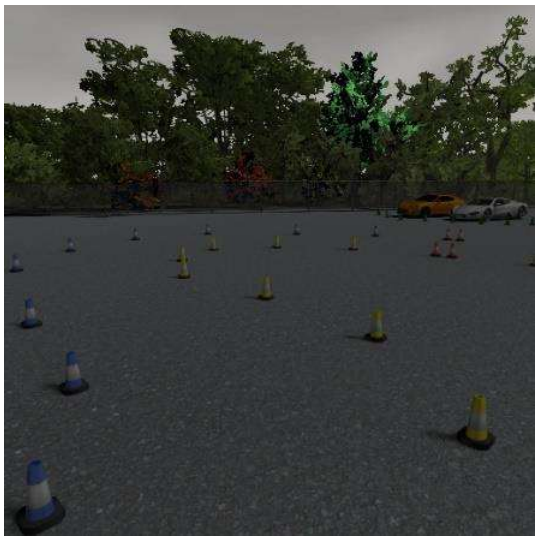
(d)



(e)

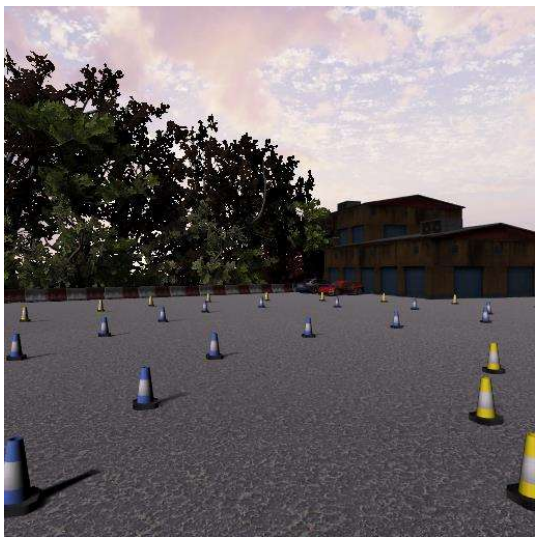


(f)



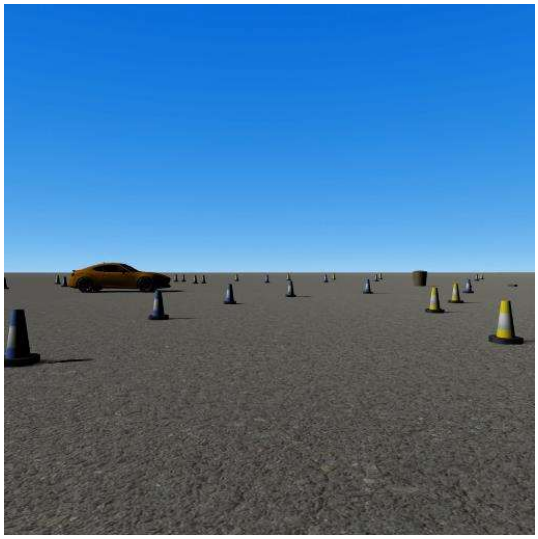
(g)

(h)



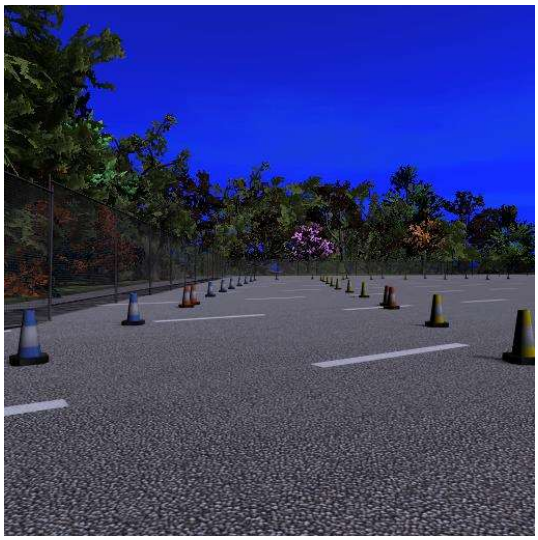
(i)

(j)



(k)

(l)



(m)

(n)

(o)

(p)

Figure 32. Full dataset for inference tests.

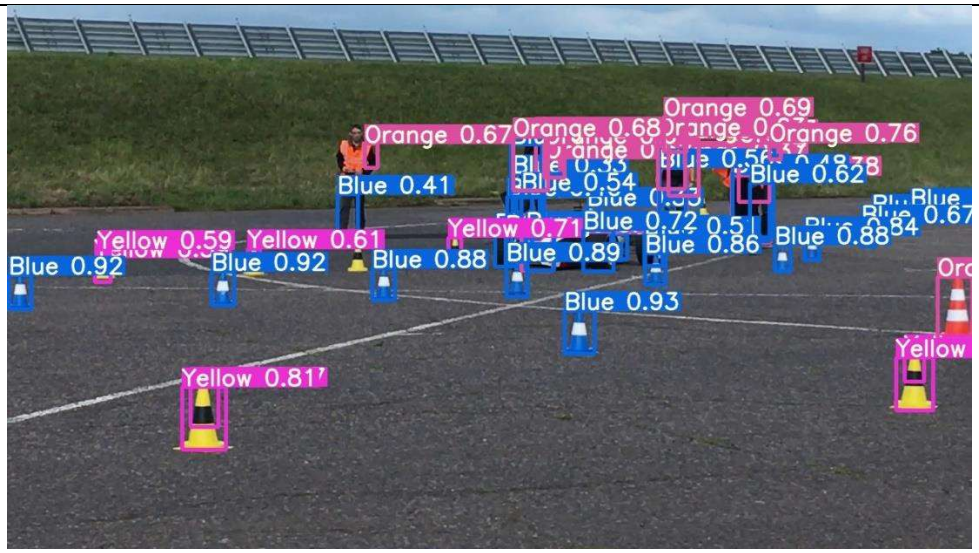
Appendix IV

The full synthetic and real image inference results for all models are presented in Table 12.

Table 12. Full set of inference test outputs.

Model	Inference results
YOLOv5s, Experiment 1 (a)	

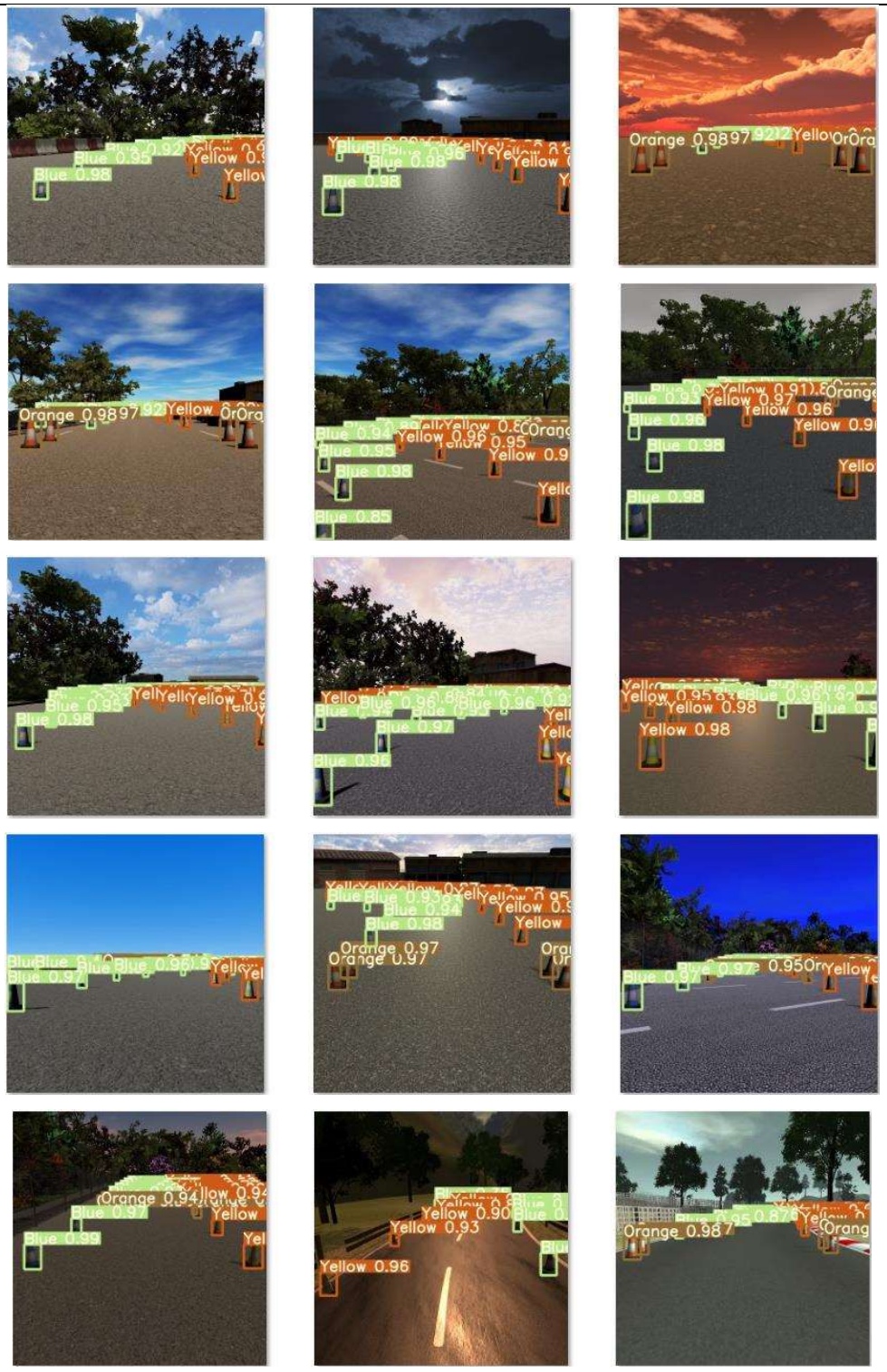
	
YOLOv5s, Experiment 1 (b)	

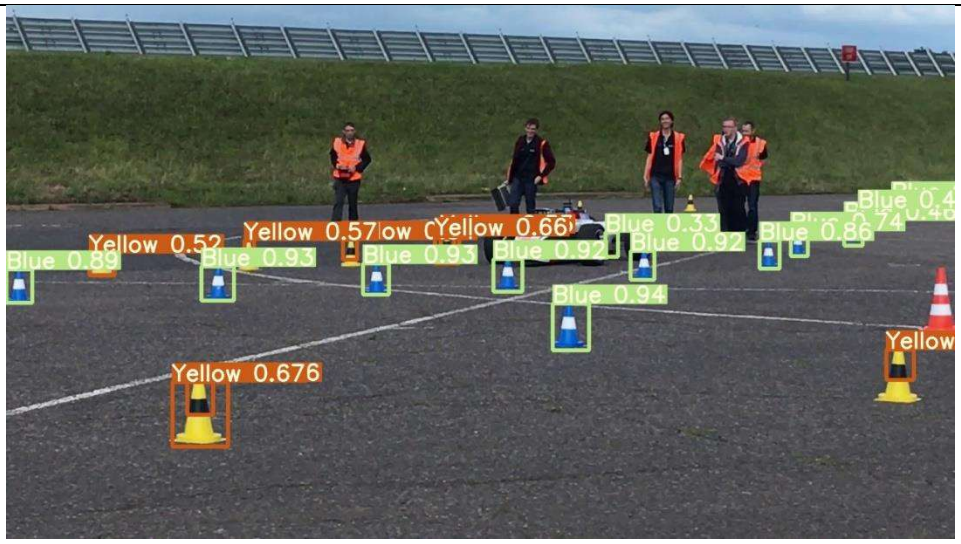
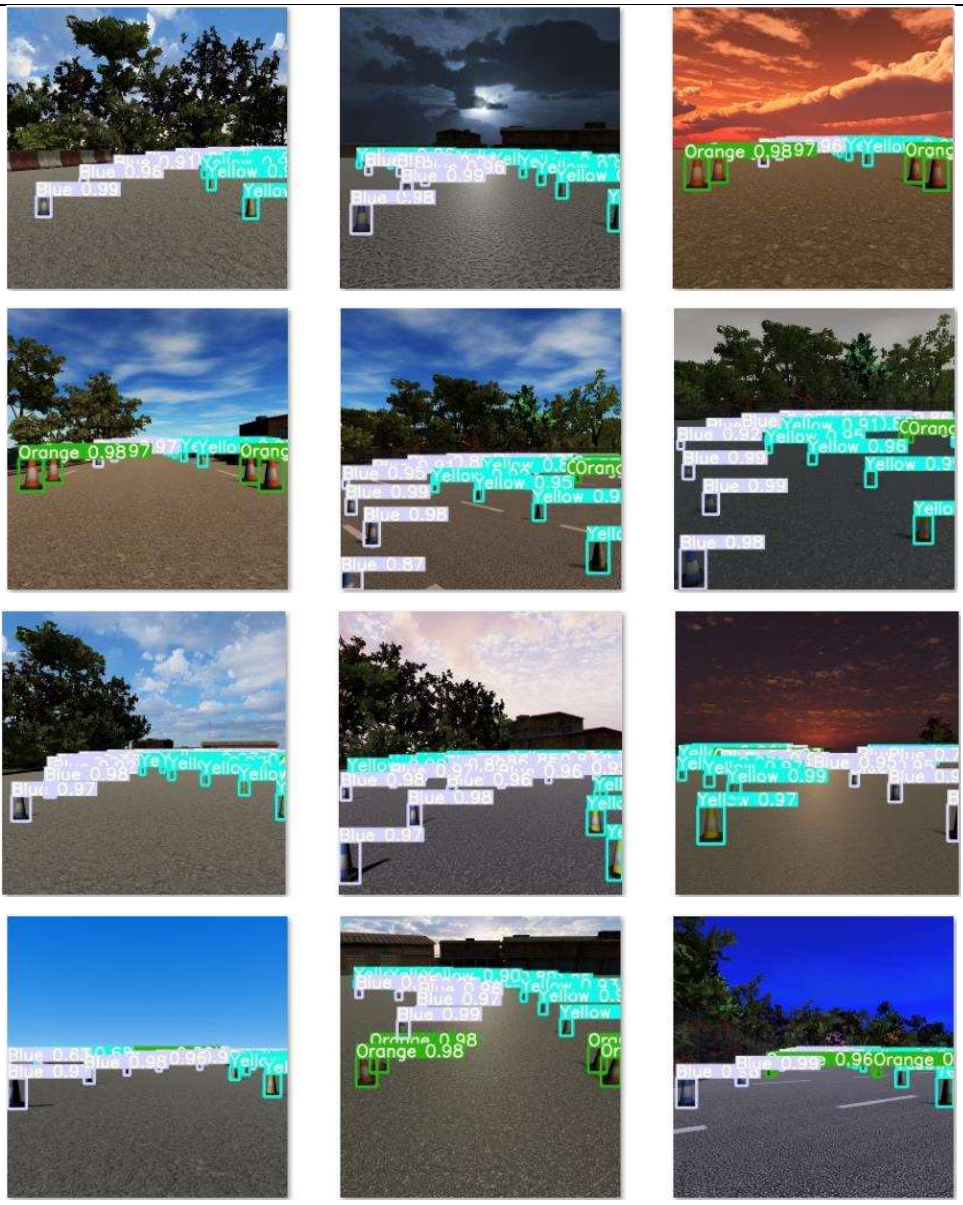
	
	
YOLOv5s, Experiment 1 (c)	

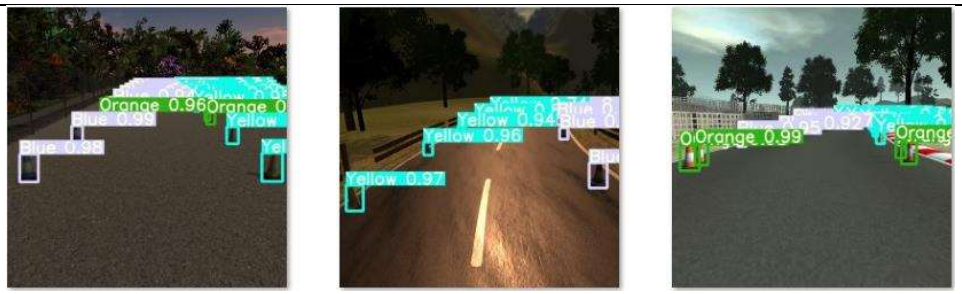
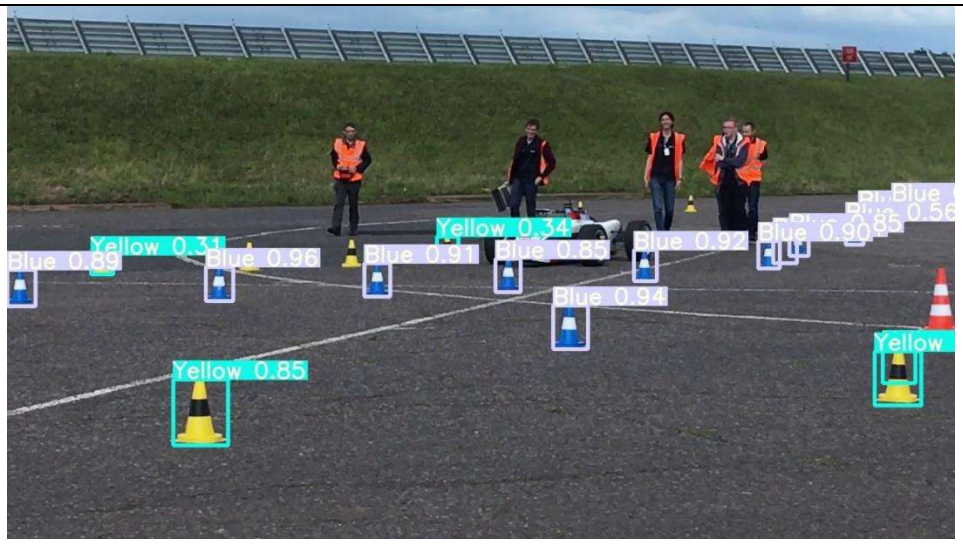
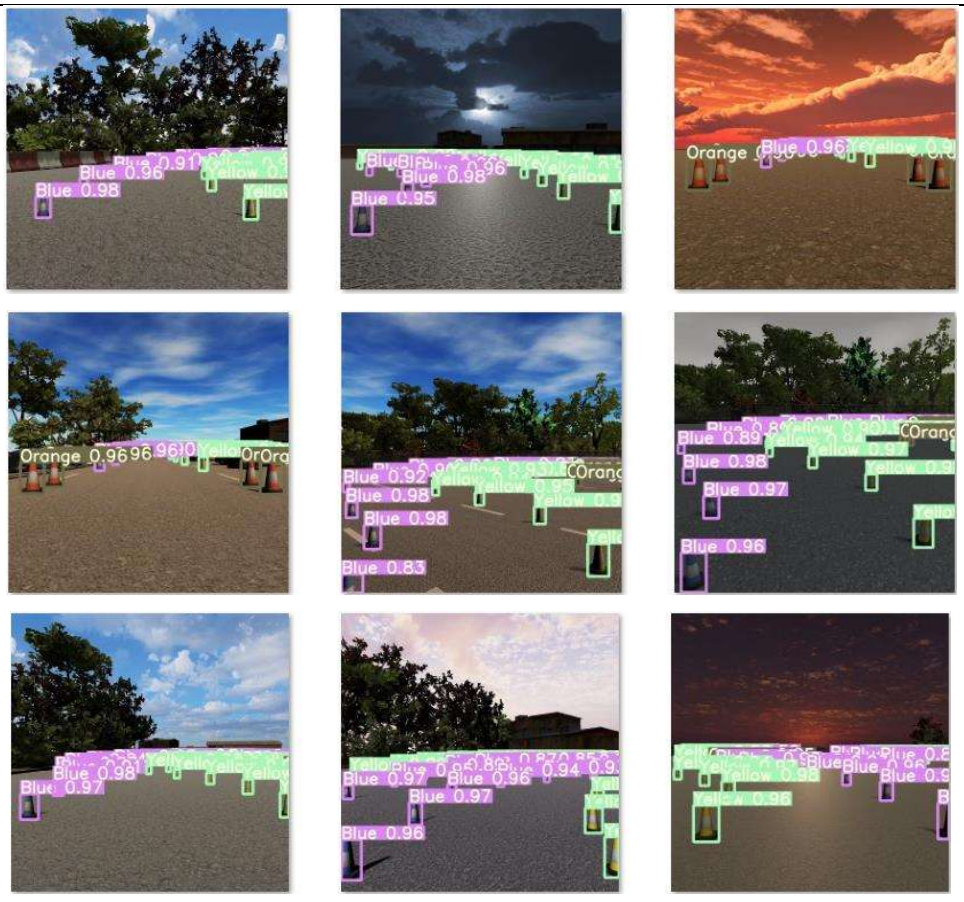
	
	
YOLOv5s, Experiment 2	

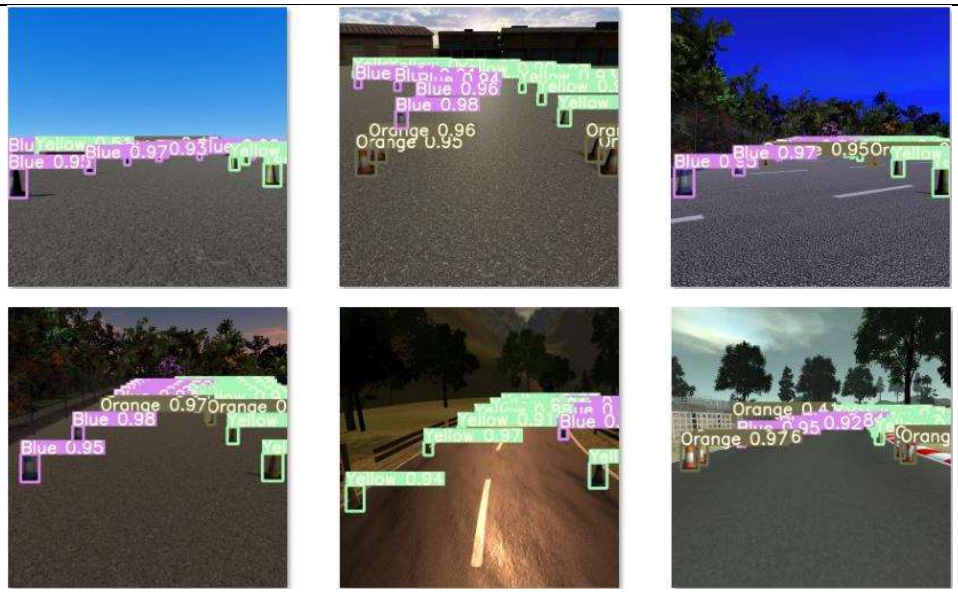
	
	
YOLOv5s, Experiment 3	

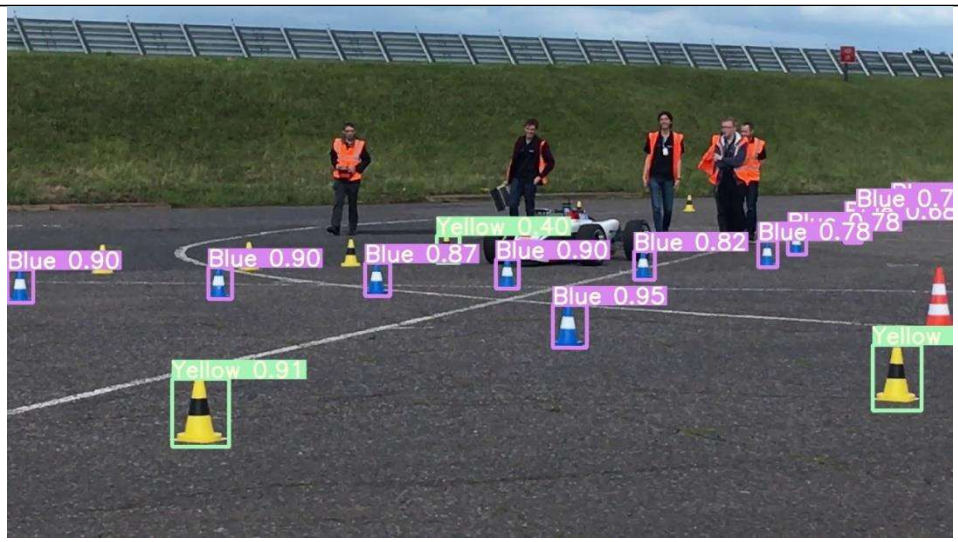

Blue 0.87
Blue 0.95
Blue 0.97
Blue 0.95
Yellow 0.96
Blue 0.97
Blue 0.97
Blue 0.96
Blue 0.97
Yellow 0.96
Blue 0.96
Blue 0.97
Yellow 0.92
Yellow 0.95
Blue 0.86
Yellow 0.49
Blue 0.87
Blue 0.87
Blue 0.81
Blue 0.89
Yellow
Yellow 0.75

YOLOv5m, Experiment 5	

	
YOLOv5l, Experiment 5	

	
	
YOLOv5x, Experiment 5	


Blue 0.98
Orange 0.96
Orange 0.95
Orange 0.97
Blue 0.98
Blue 0.95
Orange 0.976


Blue 0.90
Blue 0.90
Blue 0.87
Yellow 0.40
Blue 0.90
Blue 0.82
Blue 0.78
Blue 0.95
Yellow 0.91