DEBRIS DETECTION FOR FORMULA ONE

Name: William J B Newbould
Student Number: 200415855
Degree Program: BSc Computer Science
Supervisor: Zhichao Ma

Word Count: 14,961

Abstract

I present two methods for detecting on-track obstructions in real-time using deep learning in order to improve response times to incidents in Formula One. The suggested methods involve utilising YOLOv8 and YOLOS in a live Formula One environment in order to detect the presence of on-track debris, marshals, and recovery vehicles. This research aims to determine whether the produced real-time object detection models can feasibly be employed in such a vital health and safety role within the world of motorsport. Alongside this, this research also aims to compare two vastly different approaches to object detection that share both the same aim and the same YOLO principles but differ in almost every other aspect. The proposed solutions utilise the on-board footage from the various Formula One drivers and aim to identify any on-track obstructions observed through this footage. Having run both models in real-time on a given input video, the state-of-the-art medium sized variant of YOLOv8 achieved 0.877 mAP50 with a throughput of 29.5 FPS on a NVIDIA GeForce RTX 3070 GPU, while the YOLOS-Tiny model achieved 0.645 mAP50 with a throughput of 26.9 FPS.

Declaration

I declare that this document represents my own work except where otherwise stated.

Table of Contents

Table of Figures

Glossary

Debris: Debris is defined as pieces from something that has been destroyed or pieces of rubbish or unwanted material that are spread around.

In the context of Formula 1, debris refers to any foreign object on the track that could endanger the drivers' safety. Fragments of tire rubber, body work parts from cars, loose tyres, and other items that could come off during a given session all fall under the category of debris.

mAP: Mean Average Precision (mAP) is the current benchmark metric used in the computer vision community to evaluate the robustness of object detection models.

mAP50: The mean average precision calculated at intersection over union (IOU) threshold 0.5.

mAP50-95: The mean average precision calculated at intersection over union (IOU) thresholds from 0.5 to 0.95 with a step of 0.05 (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95)

Non-maximum Suppression: A post-processing step that selects one entity (bounding boxes) out of many overlapping entities.

1. Introduction

1.1 Motivation

Since its formation in 1950, Formula One has been the pinnacle of open-wheel single-seater racing in the world. In this time, Formula One has continuously been perceived as a decidedly glamorous sport given that its drivers travel the world, driving around in fast cars, all in the hope of standing on top of a podium to needlessly spray champagne at their competitors. The edge-of-seat entertainment that the sport provides along with the lavish luxuries that encapsulate it have helped to maintain this perception, however the true nature of the sport is much more somber.

In the sport's 73-year history a total of 52 drivers have lost their lives in accidents at FIA World Championship events or other events while driving Formula 1 cars. Of the 52 drivers that have died, thirty-two occurred at official World Championship Grand Prix races, while 7 occurred during tests and 13 occurred outside of official F1 events (Pretorius, n.d.). While the majority of these tragic events occurred in the sports infancy (from the 1950s through to the end of the 1970s), the passing of Jules Bianchi (1989-2015), David Ferrer (1955-2017) and Anthoine Hubert (1996-2019) provide stark reminders that the dark history of Formula One isn't as distant as we would like to believe.

To ensure that drivers, marshals, pit crews and fans are safe, the track must be clear of obstructions when racing at full speed. While it is impossible to ever guarantee this, officials must take prompt appropriate actions so to limit both the window and severeness of any risk that may materialise. Any unintentional obstructions such as debris must be removed quickly and safely, while intentional obstructions such as recovery vehicles, marshals or safety cars must be reported to the teams and drivers to ensure awareness of the situation.

The ability to detect and subsequently safely remove debris from a live track is of unspeakable significance. If performed correctly, the race can resume quickly, and the fans get to see their favourite drivers racing once again. Alternatively, if action is slow or inadequate, the lives of every person attending the event is put at risk.

No incident better illustrates the requirement for a clear track than the occasion when Brazilian driver Felipe Massa was knocked out by a loose spring while driving at full speed during the qualifying session for the 2009 Hungarian Grand Prix. The loose spring came off the car of Rubens Barichello and landed on the racing line heading up to turn 4, which is often taken at entry speeds of approximately 255kph. Felipe Massa, running not too far behind Barichello, ran over the spring causing it to be thrown upwards towards his face, hitting his helmet, narrowly missing his eye, and knocking him unconscious. With Massa unconscious in the cockpit, the car continued on its prior trajectory and collided into the tyre wall. The injuries that Massa sustained from this event can be seen in appendix A.

With recent pioneering innovations such as the introduction of the halo structure (2018), the head and neck support device (2003), wheel tethers (1999) and new fire-protective materials (2020), Formula One seemed to be learning from its chequered history.

However, a recent incident involving French driver Pierre Gasly at the 2022 Japanese Grand Prix once again caused severe concern regarding the safety of the sport. The incident involved Gasly having a high-speed near miss with an on-track recovery vehicle in wet, low-visibility conditions. Appendix B illustrates a snapshot of the incident. While those new to the sport may not think much of this incident, for many the event triggered flashbacks to the tragic death of Jules Bianchi who suffered fatal head injuries when he hit a recovery tractor at 126kph at the same track back in 2014.

Another recent Incident saw British Formula One fan, Will Sweet, receive injuries to his arm after being struck by a flying piece of debris while attending the 2023 Australian Grand Prix. The incident occurred when Danish driver Kevin Magnussen crashed into a wall causing his right-rear tyre to come off, while a large chunk of the accompanying wheel rim launched 20 metres into the air, over the catch-fencing and cut into Will Sweet's arm. Appendix C illustrates the extent of the arm injury that Will Sweet suffered. At the same event, British Formula One World Champion, Lewis Hamilton reported seeing crowds of fans making their way onto the track moments after the chequered flag while drivers were still lapping the circuit at high speeds. (The Herald, 2023)

These events reminded the world that despite Formula One's recent innovations, the sport's safety procedures as a whole remain archaic, inconsistent, and inadequate.

1.2 Project Aim

The aim of this project is to produce a robust real-time debris detection system for Formula One that detects the presence of on track debris, marshals, and recovery vehicles.

1.3 Project Objectives

To achieve this aim, I have dissected the project into the following six objectives:

1. Investigate what procedures are currently used to identify debris on a Formula One circuit.
2. Assess the advantages and limitations of the existing procedures used to identify debris on a Formula One circuit.
3. Collect and pre-process on-board video footage of instances where debris, marshals or recovery vehicles are on track.
4. Annotate instances of on-track debris, marshals and recovery vehicles using VoTT.
5. Explore and assess different methods and tools that are currently used to create real-time object detection systems.
6. Develop an object detection system capable of identifying debris on-track in real-time by employing and building upon at least one of the identified methods.

1.4 Objectives in Depth

Objective 1: Research the current procedures used in Formula One to identify debris and report the results in the research section of the dissertation report.

Objective 2: Analyse the finding of the report produced from Objective 1 and locate areas of weakness in the existing procedures. Report this analysis in the research section of the dissertation report.

Objective 3: Gather footage from F1TV of at least five Formula One Grand Prix sessions whereby debris, marshals or recovery vehicles are visibly on track. For each incident across the five sessions, select five drivers and capture their on-board perspective of the incident over a number of laps. Once gathered, use a video editing software to trim the collected footage into condensed clips of on track incidents and reduce quality to 720p to help inference speed later on.

Objective 4: Use VoTT, an image annotation tool designed by Microsoft, to draw bounding boxes around all objects of interest in each frame of every video collected. Use these annotations to form a training, testing and validation dataset.

Objective 5: Research the current technologies and methodologies used to detect the presence of objects in images and report the findings in the Background Research section of the dissertation report.

Objective 6: Using the knowledge gained from the report produced from Objective 5, implement at least one of the identified methods to detect objects in images in real-time. If time allows, implement two methods and contrast the performance of the two methods to see which method performs better.

1.5 Report Structure

This report is structured as follows:

Section 1, Introduction: An introduction to the project, its theme, motivation, aim and objectives.

Section 2, Project Management: A focused look into how the project was managed and how the project has changed from its initial design to its final outcome.

Section 3, Background Research: This section dives into key technical and non-technical background research that inspired this project and made the work done in this project possible. There is a large emphasis on object detection, various object detection models and use cases for object detection.

Section 4, Developing a Solution: A discussion of how two solutions were developed to tackle the issues highlighted in the introduction. This includes rationale for key design decisions, stating how the intended designs will work and declaring what I hope the designs will achieve.

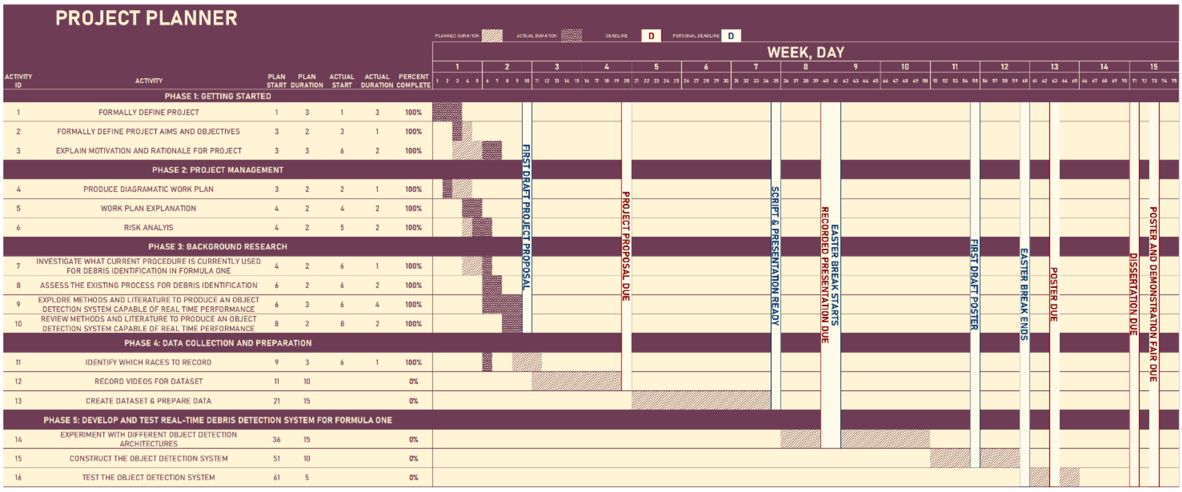
Section 5, Implementation: This section covers how I implemented my intended designs, how I decided to evaluate the solutions and any unexpected factors not anticipated in the design stage.

Section 6, Results and Evaluation: An analysis of the implemented solutions including a variety of common metrics in order to compare the two different solutions.

Section 7, Conclusion: This section reflects on the project and whether or not its aim and objectives were successfully met. It also covers my personal development and suggestions for future work and research.

2. Project Management

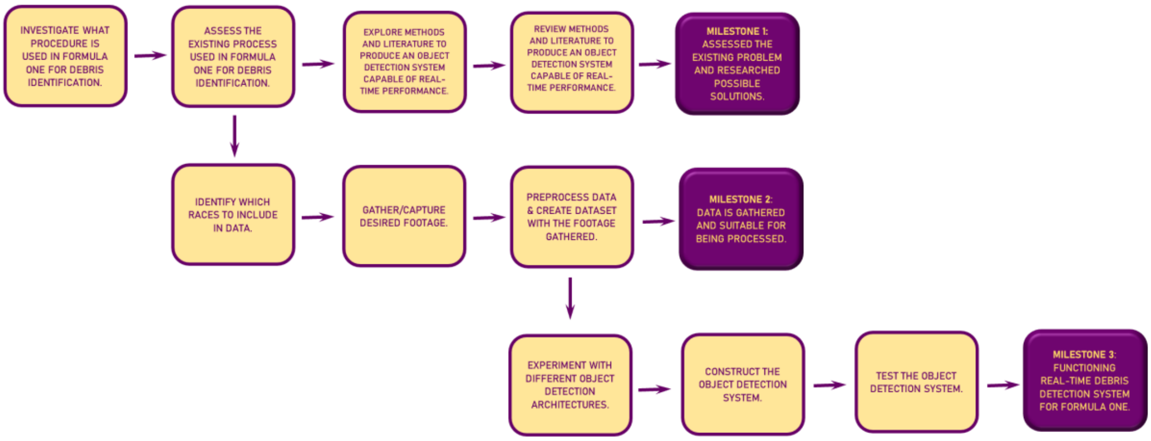
2.1 Initial Approach



PROJECT PLANNER

ACTIVITY ID	ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
	PHASE 1: GETTING STARTED					
1	FORMALLY DEFINE PROJECT	1	3	1	3	100%
2	FORMALLY DEFINE PROJECT AIMS AND OBJECTIVES	3	2	3	1	100%
3	EXPLAIN MOTIVATION AND RATIONALE FOR PROJECT	3	3	6	2	100%
	PHASE 2: PROJECT MANAGEMENT					
4	PRODUCE DIAGRAMATIC WORK PLAN	3	2	2	1	100%
5	WORK PLAN EXPLANATION	4	2	4	2	100%
6	RISK ANALYIS	4	2	5	2	100%
	PHASE 3: BACKGROUND RESEARCH					
7	INVESTIGATE WHAT CURRENT PROCEDURE IS CURRENTLY USED FOR DEBRIS IDENTIFICATION IN FORMULA ONE	4	2	6	1	100%
8	ASSESS THE EXISTING PROCESS FOR DEBRIS IDENTIFICATION	6	2	6	2	100%
9	EXPLORE METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL TIME PERFORMANCE	6	3	6	4	100%
10	REVIEW METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL TIME PERFORMANCE	8	2	8	2	100%
	PHASE 4: DATA COLLECTION AND PREPARATION					
11	IDENTIFY WHICH RACES TO RECORD	9	3	6	1	100%
12	RECORD VIDEOS FOR DATASET	11	10			0%
13	CREATE DATASET & PREPARE DATA	21	15			0%
	PHASE 5: DEVELOP AND TEST REAL-TIME DEBRIS DETECTION SYSTEM FOR FORMULA ONE					
14	EXPERIMENT WITH DIFFERENT OBJECT DETECTION ARCHITECTURES	36	15			0%
15	CONSTRUCT THE OBJECT DETECTION SYSTEM	51	10			0%
16	TEST THE OBJECT DETECTION SYSTEM	61	5			0%

Figure 1: Initial project Gantt chart.
Please see Appendix E for enlarged copy.



INVESTIGATE WHAT PROCEDURE IS USED IN FORMULA ONE FOR DEBRIS IDENTIFICATION.
ASSESS THE EXISTING PROCESS USED IN FORMULA ONE FOR DEBRIS IDENTIFICATION.
EXPLORE METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL-TIME PERFORMANCE.
REVIEW METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL-TIME PERFORMANCE.
MILESTONE 1: ASSESSED THE EXISTING PROBLEM AND RESEARCHED POSSIBLE SOLUTIONS.
IDENTIFY WHICH RACES TO INCLUDE IN DATA.
GATHER/CAPTURE DESIRED FOOTAGE.
PREPROCESS DATA & CREATE DATASET WITH THE FOOTAGE GATHERED.
MILESTONE 2: DATA IS GATHERED AND SUITABLE FOR BEING PROCESSED.
EXPERIMENT WITH DIFFERENT OBJECT DETECTION ARCHITECTURES.
CONSTRUCT THE OBJECT DETECTION SYSTEM.
TEST THE OBJECT DETECTION SYSTEM.
MILESTONE 3: FUNCTIONING REAL-TIME DEBRIS DETECTION SYSTEM FOR FORMULA ONE.

Figure 2: Initial project pert chart.

In the proposal for this project, the project plan laid out five basic phases for the structure of the project:

1. Getting Started
2. Project Management
3. Background Research
4. Data collection and preparation
5. Develop and test a real-time debris detection system for Formula One

An in-depth visualisation of this initial plan is illustrated by figure 1 and figure 2.

2.2 Management Approach

I followed the principle of Scrum management throughout the duration of this project by utilising the tasks listed in figure 1 to define a backlog of objectives and criteria that could be used to form sprints. I used the same figure to assign desired deadlines for when these sprints were to be completed.

I held weekly reviews regarding the progress of sprints. In these reviews, I assessed whether the objectives of the sprint would be met and whether the sprint would be completed in the desired timeframe. The outcome from the weekly reviews would go on to help form the objective and timeframe of the next sprint.

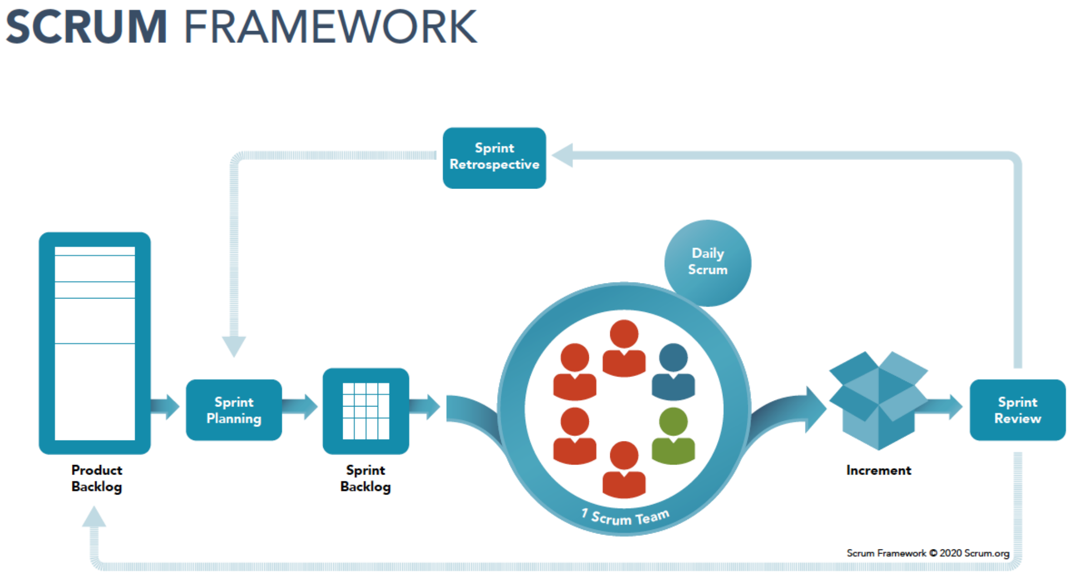


Figure 3: Graphical summary of the official Scrum framework (Scrum, 2020).

Overall, I believe this management style operated successfully because at every stage I knew what tasks had to be done by which date and what tasks were remaining and yet to commence. As I did not initially know what models I would be implementing, I found the backlog nature of Scrum to be rather practical as I could add to the backlog of tasks at any point, so long as it did not interfere with the objective or progress of the current sprint.

2.3 Changes Made to Initial Plan

The original plan for this project was to produce one CNN-based object detection model to detect the presence of debris on a Formula One track and if time allowed, implement another CNN-based object detection model so to allow for comparison between two different models.

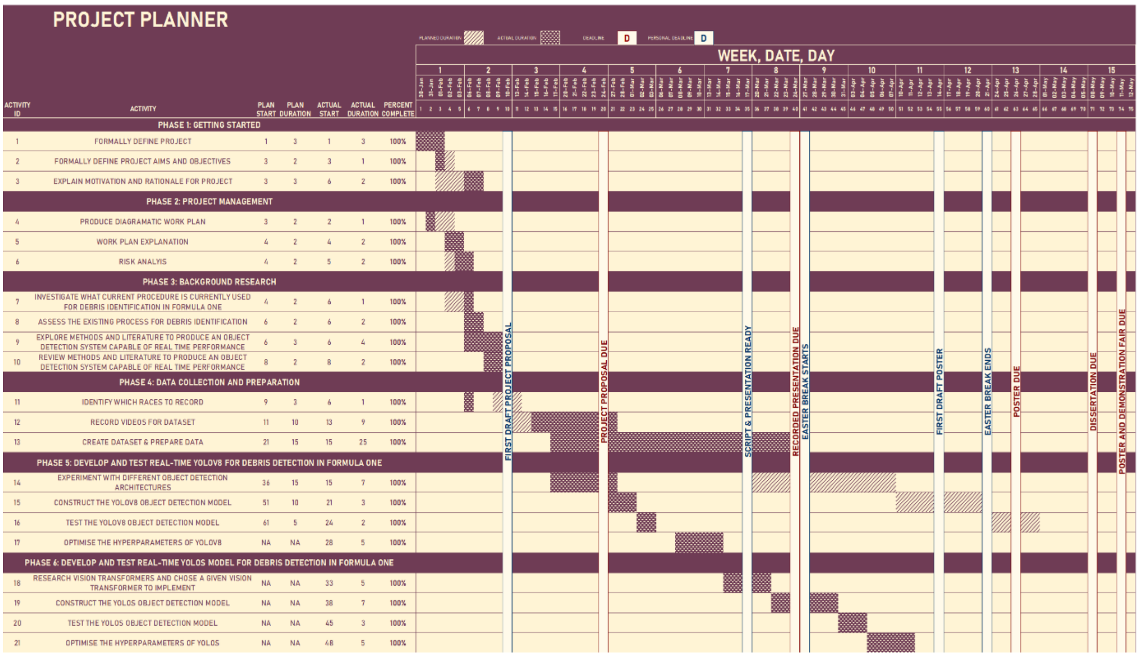
To my surprise, I was able to quickly implement the CNN-based YOLOv8 model and subsequently had plenty of time spare to add to what I had already accomplished. Having had discussions with my project supervisor regarding other potential models to implement, I came across the concept of vision transformers. Having researched the concept further, I decided that I would rather implement a vision transformer than another CNN-based object detection model such as Single Shot Detector (SSD) as it would provide an opportunity to contrast two entirely different approaches to tackle the problem of object detection in computer vision.

Another alteration made to this project since the submission of the project proposal is the objects of interest that I aimed to detect. Originally, I desired only to detect the presence of debris, however, having read Track Foreign Object Debris Detection based on Improved YOLOv4 Model (Song, et al., 2022), I realised that debris isn't the only objects that the drivers have to avoid on track and subsequently added marshals and recovery vehicles to the list of objects of interest.

The biggest change made to the initial plan for this research project was the decision to no longer display the post-model on-board footage (the live predictions of the model) from each of the 20 drivers. With the aim of reducing visual and observational overload, I found watching the live detections from all of the 20 drivers at once to be more observationally exhaustive than trying to watch the 20 video streams without the detections. With boxes constantly flickering on and off in various parts of the screen it was impossible to focus on one driver yet alone all the detections from the 20 drivers. Such a solution should ideally provide clarity and knowledge to its user, but instead I found myself to be confused and dazed. This implementation provided too much information and was subsequently overwhelming. As a result, I decided to drop this idea and switched my focus to the performance of the object detection models rather than the method of delivery to the users.

2.4 Schedule

The changes to the initial plan referred to in section 2.3 resulted in the alteration of the project schedule whereby a sixth stage was added in order to research, implement, test, and optimise the chosen vision transformer. This caused the subsequent change from figure 1 which depicts the initial project schedule to figure 4 which depicts the final project schedule having completed the project. Please see appendix F to see an enlarged version of the final project schedule.



PROJECT PLANNER

ACTIVITY ID	ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
	PHASE 1: GETTING STARTED					
1	FORMALLY DEFINE PROJECT	1	3	1	3	100%
2	FORMALLY DEFINE PROJECT AIMS AND OBJECTIVES	3	2	3	1	100%
3	EXPLAIN MOTIVATION AND RATIONALE FOR PROJECT	3	3	6	2	100%
	PHASE 2: PROJECT MANAGEMENT					
4	PRODUCE DIAGRAMATIC WORK PLAN	3	2	2	1	100%
5	WORK PLAN EXPLANATION	4	2	4	2	100%
6	RISK ANALYIS	4	2	5	2	100%
	PHASE 3: BACKGROUND RESEARCH					
7	INVESTIGATE WHAT CURRENT PROCEDURE IS CURRENTLY USED FOR DEBRIS IDENTIFICATION IN FORMULA ONE	4	2	6	1	100%
8	ASSESS THE EXISTING PROCESS FOR DEBRIS IDENTIFICATION	6	2	6	2	100%
9	EXPLORE METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL TIME PERFORMANCE	6	3	6	4	100%
10	REVIEW METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL TIME PERFORMANCE	8	2	8	2	100%
	PHASE 4: DATA COLLECTION AND PREPARATION					
11	IDENTIFY WHICH RACES TO RECORD	9	3	6	1	100%
12	RECORD VIDEOS FOR DATASET	11	10	13	9	100%
13	CREATE DATASET & PREPARE DATA	21	15	15	25	100%
	PHASE 5: DEVELOP AND TEST REAL-TIME YOLOV8 FOR DEBRIS DETECTION IN FORMULA ONE					
14	EXPERIMENT WITH DIFFERENT OBJECT DETECTION ARCHITECTURES	36	15	15	7	100%
15	CONSTRUCT THE YOLOV8 OBJECT DETECTION MODEL	51	10	21	3	100%
16	TEST THE YOLOV8 OBJECT DETECTION MODEL	61	5	24	2	100%
17	OPTIMISE THE HYPERPARAMETERS OF YOLOV8	NA	NA	28	5	100%
	PHASE 6: DEVELOP AND TEST REAL-TIME YOLOS MODEL FOR DEBRIS DETECTION IN FORMULA ONE					
18	RESEARCH VISION TRANSFORMERS AND CHOSE A GIVEN VISION TRANSFORMER TO IMPLEMENT	NA	NA	33	5	100%
19	CONSTRUCT THE YOLOS OBJECT DETECTION MODEL	NA	NA	38	7	100%
20	TEST THE YOLOS OBJECT DETECTION MODEL	NA	NA	45	3	100%
21	OPTIMISE THE HYPERPARAMETERS OF YOLOS	NA	NA	48	5	100%

Figure 4: Final Project Gantt Chart.
Please see Appendix F for enlarged copy.

3. Background Research

3.1 Situation Research

3.1.1 Introduction to Race Control

On any given Grand Prix weekend in Formula One, there is a team of officials, collectively known as Race Control, whose responsibility it is to oversee proceedings, uphold the rules of the sport and ensure the safety of all personnel at the event whether driver, marshal, fan or otherwise. The environment in which Race Control operates is illustrated by appendix D.

Race Control is composed of the following personnel:

The Race Director: A permanent appointee of the FIA (Fédération Internationale de l'Automobile, the governing body of Formula One), who has overriding authority over the majority of the decisions made by Race Control. The Race Director is responsible for stopping races with a red flag, the use of safety cars and virtual safety cars, and the use of black and white warning flags. The role also involves reporting incidents to the stewards for further investigation, communicating to the track workers via the Race Clerk, defining track limits at each Grand Prix weekend and reviewing and improving the methods of Race Control. The Race Director acts as the figure head of Race Control and is subsequently responsible for communicating decisions to relevant parties whether that is another official, one of the teams or even the media.

The Permanent Starter: A permanent appointee of the FIA, who is responsible for overseeing the start procedure at the beginning of each race.

The Race Clerk: Also known as the clerk of the course, they must be in constant communication with both the Race Director and all marshal posts throughout each session. They are the point of contact between Race Control and the track workers to inform them of what the Race Director has decided. This official is nominated by the organisers of the event (typically the owners of the given track in the case of Formula One).

The Stewards: Responsible for adjudicating racing incidents and apply reprimands for breaches of the sporting code and technical regulations if they have been reported to them by officials or delegates. For every Grand Prix, a panel of four stewards are appointed, three of which are nominated by the FIA. The other steward is an ex-racing driver who aids Race Control by offering the viewpoint from the perspective and mindset of a racer.

In addition to these roles, there is often a handful of additional FIA staff and experienced officials from the circuit itself who know the track well and understand the best practices to ensure safety given the layout of the track. Altogether Race Control can contain up to roughly 10 to 20 officials for any given Formula One Grand Prix session.

3.1.2 The Drawbacks of The Procedures Used by Race Control to Detect Debris

Race Control has access to hundreds of camera angles, live data, team radio messages and much more throughout the course of a given session and must use these resources to make decisions concerning on-track events in a matter of minutes. Subsequently, the existing structure heavily trusts the observation and attention capacity of the officials on duty and despite the mass of data made available to them, Race Control still largely relies on radio communication with track-side marshals in order to identify on-track debris, which can be very time inefficient.

The problem here is that although these officials are provided with a substantial amount of data, it is not possible for a small team to monitor every single video stream from each camera, observe every single data stream or watch every driver's on-board footage. Subsequently it is impossible for such a small team to visually observe the entire expanse of the track and be aware of all on track events at a given moment. Regardless of training, no human possesses the awareness, vision, and consistency of judgement to correctly identify potentially dangerous situations at any given point on the track and make these decisions in the split second that they require to be taken in so that safety is ensured. With responsibilities such as adjudicating and reprimanding sporting infringements on top of assessing the suitability and safeness of the track, it is hard to believe that officials are aware of incidents the instant that they happen.

3.2 Technical Research

3.2.1 The Field of Object Detection in Computer Vision

Object detection is a computer vision technique for locating instances of objects in images or videos. To do this, object detection algorithms draw bounding boxes around instances of detected objects, partnered with labels to respectively illustrate both where and what the object is.

The ultimate goal of object detection is to replicate, if not surpass the human ability to recognise and locate objects.

Examples of object detection in practice include face recognition, vehicle counting, pedestrian detection and self-driving cars. It can even be used for tracking objects across multiple frames in a video for example tracking the position of a football player throughout a match.

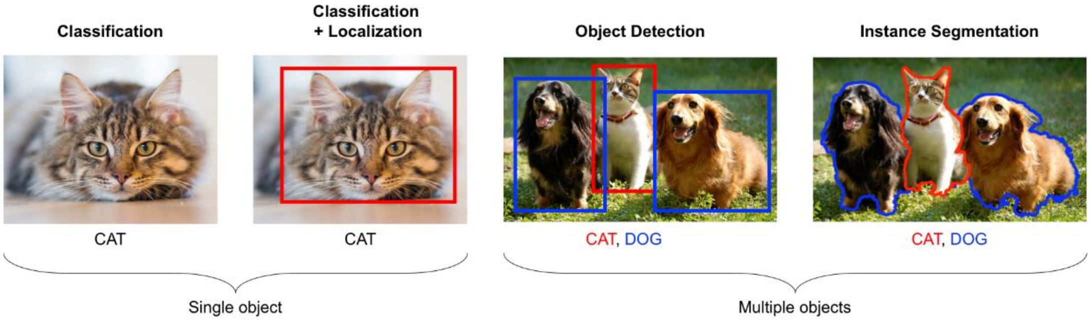
Mean Average Precision, commonly referred to as mAP is the current benchmark metric used in the computer vision community to evaluate the robustness of object detection models.

3.2.2 Object Detection vs Image Classification and Instance Segmentation

Object detection differs from other computer vision techniques such as image classification and instance segmentation through the way the image is labelled and annotated.

Image classification algorithms do not annotate the image at all and instead assign a singular label to the entire image to declare what the image contains. If multiple objects are present within an image, the classification algorithm will return a singular label that best represents that image but will not indicate where that object is located. In contrast to this, object detection algorithms draw bounding boxes around each detected object and assign a label to that bounding box. Subsequently, an image in object detection can have multiple bounding boxes and thus multiple labels. Image classification is known to be well-adapted to recognising non-physical features such as brightness but tend to be outperformed by object detection algorithms in their ability to detect physical objects.

Instance segmentation, like object detection, aims to detect instances of objects in images and videos, however, the two techniques differ in the way in which the detected objects are annotated. Instance segmentation combines semantic segmentation with object detection to first detect the instances of objects of interest, and then segment each object within the detected boxes. While object detection algorithms draw rectangular bounding boxes around detected objects, instance segmentation defines which pixels represent a given object and subsequently provides a tighter fitting bounding box that reveals the shape of the object in addition to its location within the image.



3.2.3 Types of Object Detection Models

Prior to the popularisation of deep learning, the field of object detection used classical machine learning techniques to detect a number of common features across a given image and classify their clusters using logistic regression, histograms, or random forests. Some famous classical approaches to object detection include:

- Viola–Jones object detection framework based on Haar features.
- Scale-invariant feature transform (SIFT).
- Histogram of oriented gradients (HOG) features.

Nowadays, modern deep learning approaches vastly outperform these classical techniques. The vast majority of these deep learning approaches make use of Convolutional Neural Networks (CNNs), however, having become vastly successful in the field of Natural Language Processing, Vision Transformers have become one of the most promising neural network architectures for object detection.

There are two classes of deep learning-based object detection algorithms: One-stage and two-stage.

One-stage object detection uses a single pass of the input image to make predictions. They process an entire image in one pass, making them computationally efficient, however, this efficiency comes at the cost of accuracy as the models tend to be generally less accurate than other methods and less effective at detecting small objects.

Meanwhile, two-stage object detection uses two passes of the input image to make predictions. The first pass is used to generate a set of potential object locations, and the second pass is used to refine these proposals and make final predictions. This approach is more accurate than one-stage object detection but is also more computationally expensive.

In most cases, one-stage object detection is better suited for real-time applications, while two-stage object detection is better for applications where accuracy is more important. Listed below are some examples of popular one-stage and two-stage object detectors.

One-Stage Object Detectors

- YOLO (You Only Look Once)
- SSD (Single Shot Detector)

Two-Stage Object Detectors

- RCNN
- Fast RCNN
- Faster RCNN
- RFCN
- Mask RCNN

3.2.4 Convolutional Neural Network-based Object Detection Models

3.2.4.1 One-Stage CNN-based Object Detection Models

3.2.4.1.1 YOLO

You Only Look Once (YOLO) is a family of object detection models named as such because of their ability to make predictions for a given image with just one forward pass. In this singular forward pass the network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

The original YOLO object detector was released in 2016 and was substantially faster than other object detectors at the time. Since then, multiple variants of YOLO have been released, with the latest variant, known as YOLOv8, being released on 10th January 2023 by Ultralytics. YOLOv8 has five versions ranging from YOLOv8n (the smallest and fastest model) with a 37.3 mAP score on COCO to YOLOv8x (the largest and slowest model), scoring a 53.9 mAP score on COCO.

YOLO's fame is attributable to its considerable accuracy while maintaining a small model size.

3.2.4.1.2 Single-Shot Detector (SSD)

SSD, like YOLO, takes only one shot to detect multiple objects present in an image and similarly to earlier versions of YOLO utilises anchor boxes to do so. SSD makes use of a base network, usually VGG-16, to act as a feature extractor. Prediction and confidence scoring of the bounding boxes is done by multiple feature maps of different sizes that represent multiple scales and as such SSD is typically able to detect a wide range of different sized objects.

SSD500 achieves a mAP50 score of 46.5% and a mAP50-95 score of 26.8% with a throughput 22 FPS on the COCO dataset (Liu, 2016).

3.2.4.2 Two-Stage CNN-based Object Detection Models

3.2.4.2.1 Faster RCNN

Faster RCNN is a state-of-the-art two-stage object detector that improves upon the work of R-CNN and Fast R-CNN by replacing the selective search algorithm in Fast R-CNN with a CNN called Regional Proposal Network (RPN). This change vastly improved computational efficiency and resulted in an inference speed of roughly 200ms and a throughput of ~5 FPS. (Ren, 2015)

While deemed 'fast' for a two-stage object detector, it still trails the likes of YOLO or SSD in terms of inference speed, likely due to the 300 generated proposals per image. Despite this, Faster R-CNN only achieve a mAP50 score of 42.7% and a MAP50-95 score of 21.9 on the COCO dataset.

3.2.5 Transformer-based Object Detection Models

3.2.5.1 YOLOS

You Only Look Once Sequence (YOLOS) aims to implement the principles of YOLO with a transformer as the backbone architecture instead of a CNN. Yuxin Fang et al, the creators of YOLOS, state that the model was not designed with the intention of being state-of-the-art, but merely to demonstrate the versatility and transferability of transformers from image recognition to object detection.

YOLOS has three versions ranging from YOLOS-Tiny (the smallest and fastest model) with a 28.7 mAP score on COCO to YOLOS-B (the largest and slowest model), scoring a 42.0 mAP score on COCO. Given the model does not aim to be state-of-the-art, YOLOS achieves a similar mAP score to that of DETR and more complex models such as Faster R-CNN.

3.2.5.2 DETR

Detection Transformer (DETR) is a set-based object detector that uses an encoder-decoder transformer on top of a convolutional backbone. A set-based object detector means that the problem of object detection is viewed as a direct set prediction problem. This differs from the classical thinking related to object detection which often involved repurposing classifiers to perform detection.

Variants of DETR make use of either ResNet-50 or ResNet-101 as the backbone architecture. DETR-R50 achieves a mAP score of 42.0% on COCO while DETR-R101 achieves a mAP score of 43.5%.

4. Developing a Solution

4.1 Proposed Solution

Having researched recent events such as Will Sweet's debris-related injury at the 2023 Australian Grand Prix and Pierre Gasly's near miss with a recovery vehicle at the 2022 Japanese Grand Prix, I believe that the consistency and accuracy of judgment within Race Control could be vastly improved. I believe that in its current format, the officials that make up Race Control have too many responsibilities, and as such are overloaded and thus unable to perform their jobs efficiently and effectively to the standard that is required.

In order to relieve these officials of one of their numerous tasks, I propose a robust real-time object detection system using one of YOLOv8 or YOLOS to detect the presence of debris, marshals and recovery vehicles on a given live track. I chose these three particular categories of objects so to improve safety for drivers, marshals, and fans in addition to hopefully preventing some of the situations I have already highlighted including Felipe Massa's injury and Pierre Gasly's near miss.

The work completed throughout this research project will focus on implementing two robust real-time object detection models using two different approaches. The deployment of the models along with a graphical user interface that the officials would hypothetically interact with in order to utilise the models' capabilities would be implemented in future work outside the realms of this project.

4.1.1 Intended Use for the Proposed Solution

For this solution, I wanted to introduce deep learning technology into the detection process at Race Control so to allow for faster and superior decision making concerned with on track safety. I believe it to be neither ethical nor sufficiently safe to design this system with the intention for the deep learning solution to be a complete alternative to Race Control whereby the system would be responsible for both the detection of foreign objects as well as the subsequent action. In accordance with this belief, the detection system that I propose is designed with the intention to be used alongside the existing structure already established at Race Control and should be used to further support the officials on duty. If granted access to all the live video footage that Race Control has access to, the proposed solution could make use of the stationary cameras scattered around the track in addition to the on-board cameras attached to the cars in order to visually cover as much of the track as possible. If granted access to information such as the positional data from the cars and stationary cameras, the system interface could not only alert Race Control of detections but inform them of where on the track the debris was located.

4.1.2 How the Proposed Solution Would Improve the Current Situation

I believe that the addition of an object detection system, as previously described, would be of great benefit to the current setup of Race Control. The intention of this solution would ultimately be to notify and inform the Race Director of the presence of debris, marshals or recovery vehicles on track and allow the Race Director to declare the appropriate responding action(s). As a result, the object detection system in practice would effectively remove the need to constantly monitor on-track cameras throughout a given session in search of on track obstructions. The video footage from the on-track cameras could then be used sparingly to provide confirmation before affirming the chosen action. This change in the detection process at Race Control would reduce the strain on the attention and vision capabilities of the officials as the information relating to the current state of the live track would be substantially condensed compared to the present procedure.

4.2 Design Decisions

4.2.1 Solution

There are a range of techniques in the field of computer vision that enable us to highlight the presence of an object within an image. Such techniques including, object detection, image classification and instance segmentation are discussed extensively in section 3.2.2 of this report. Of these techniques, I believed object detection best suited the problem at hand due to its ability to detect multiple instances of objects and illustrate the position of the detected objects within the image. In contrast, image classification fails to do either of these. While instance segmentation performs a similar role to that of object detection, I believed that the annotation process for instance segmentation would be too faffy given the nature of debris and the additional benefit of providing a tighter fitting bounding box would not add any particular value in this use case.

4.2.2 Data

For this research project, I was required to make a dataset of my own as no dataset existed that contained images of debris, marshals and recovery vehicles in a Formula One environment. In order to create this dataset, I chose to collect a number of videos from the 2021 Formula One season as I was aware of several incidents throughout this season in which debris, marshals or recovery vehicles were present on track. For each incident, I wanted to gather multiple on-board perspectives over a number of laps so to have footage that covered the entirety of the incident, from the cause of the incident through to the resolution and subsequent return to a clear track. Having multiple perspectives would allow for unique observations of incidents that some drivers may witnessed but other drivers may not have. I chose to use the drivers' on-board footage due to its availability to me. Ideally, I would have liked access to the various cameras scattered around the track as well, but I could not source this data.

Once gathered, this data was annotated with the intention to be used for training, testing and validation purposes when producing the proposed object detection models. I aimed to have at least 10,000 annotated images for the training dataset so to have a broad range of data for the models to learn from. To further broaden the data and improve robustness, I used data augmentation to generate additional images with varying brightness, saturation, hue, exposure, and blurriness. Data augmentation expands the breadth of examples for a model to learn from and subsequently is known to help improve a model's ability to generalise.

4.2.3 Chosen Architectures

4.2.3.1 YOLOv8

Launched on 10th January 2023, YOLOv8 is the latest state-of-the-art iteration of the You Only Look Once (YOLO) family of computer vision models, developed by Ultralytics. It can be used for object detection, image classification, instance segmentation and even object tracking thanks to a recent update.

4.2.3.1.1 Why YOLOv8?

I decided to use YOLOv8 as the architecture for one of my models because it is currently one of the most accurate one-stage CNN based object detectors available, making it easily able to perform in real-time while achieving good levels of accuracy. This is illustrated by the 50.2 mAP50-95 score that the medium variant of YOLOv8 achieves on the COCO dataset.

With a launch date of the 10th of January 2023, YOLOv8 became public just as this project commenced and subsequently provided a perfect opportunity to use up-to-date state-of-the-art technology.

I specifically chose YOLOv8 over other alternatives due to its well-packaged, well-documented, and well-publicised nature. The convenience of single line commands to perform tasks such as training and validation through the Ultralytics Python package made the model widely accessible while the existence of numerous publications and tutorials online further increased the exposure and appeal of the model. The aforementioned characteristics make the model widely accessible to a range of developers, allowing YOLOv8 to be vastly popular within the computer vision community.

In contrast to this, alternatives such as Faster RCNN failed to meet the requirement to run inference in real-time, with Faster RCNN only capable of a throughput of 5 FPS (an inference speed of ~0.2s). Meanwhile YOLOv8 substantially outperforms SSD500 which only achieved an mAP50-95 score of 26.8 on the COCO dataset.

4.2.3.1.2 Variants of YOLOv8

Model	size (pixels)	mAP$^{val}_{50-95}$	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 5: The performance metrics for the five variations of YOLOv8 (Ultralytics, 2023).

Ultralytics provide five variations of YOLOv8 ranging from YOLOv8n, the smallest and quickest model (with a 37.3 mAP50-95 score on COCO) through to YOLOv8x the largest and most accurate model (with a 53.9 mAP50-95 score on COCO). Altogether, the five variants provide a flexible range of models that can cater to requirements of most tasks and hardware limitations thanks to the subtle performance differences from one variant to the next. With appropriate hardware and hardware optimisation software such as NVIDIA's TensorRT or OpenCV's dnn module, all five variations of YOLOv8 are capable of running inference at real-time speeds.

Unable to successfully implement these hardware optimisation techniques, I found that the best balance between inference speed and model accuracy was found with the medium variant of YOLOv8, which is highlighted in figure 5. This variant achieved a throughput of ~30 FPS on an NVIDIA GeForce RTX 3070 GPU, matching the throughput of the intended input videos while providing similar mAP50-95 scores when compared to the larger variations available.

4.2.3.1.3 Limitations of YOLOv8

As with previous iterations of YOLO models, YOLOv8 can struggle with the detection of small objects, especially those in the background of images. In order to reduce the scale of this problem I plan on using a range of data augmentation techniques including cropping.

Another issue that YOLOv8 arguably suffers from is that its confidence scores can be somewhat cautious whereby some objects that the model has been trained on receive confidence scores of just 60-80%. This is not necessarily an issue, but it is a stark difference from the confidence score system used by YOLOS. In order to overcome this minute issue, I plan to use a lower confidence threshold for YOLOv8 than for YOLOS.

4.2.3.1.4 YOLOv8's Architecture

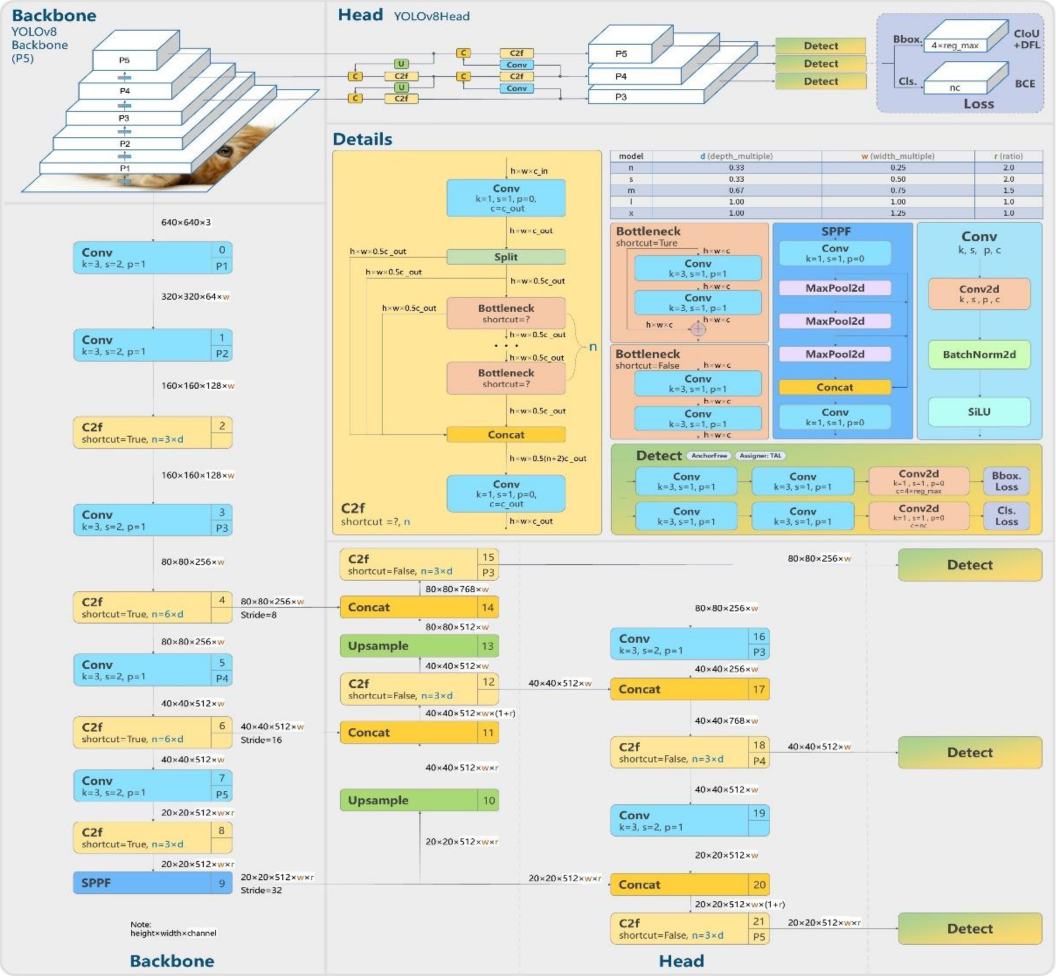


Figure 6: A depiction of the YOLOv8 architecture (Ultralytics, 2023).

As of the time of writing, there is currently no official paper for YOLOv8 and so it is not yet known as to what research methodologies were used to produce the architecture. What is known, however, is that YOLOv8 is an anchor-free model.

Previous iterations of YOLO made use of thousands of anchor boxes in order to allow the detection model to process an entire image at once, thus making real-time object detection possible. These anchor boxes were used to make thousands of detections which would later be methodically dismissed based on their IOU scores (typically through the use of Non-Maximum Suppression). Anchor free detection reduces the number of predictions made by the model and so subsequently speeds up the process of Non-Maximum Suppression (NMS) as there are fewer bounding boxes to calculate IOU scores for. This subsequently improves the model's inference speed and is also known to help generalisation ability.

4.2.3.2 YOLOS

YOLOS is an object detection model that was proposed in June 2021 in the research paper 'You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection' (Liu, 2021). YOLOS aimed to integrate transformer technology with YOLO principles within the field of object detection.

Note that YOLOS is a YOLO model purely through the shared principle of only looking once. Beyond this shared idea, the YOLOS network architecture shares nothing else with previous YOLO models.

4.2.3.2.1 Why YOLOS?

Ultimately, the choice of which vision transformer to use for the second model of this research project came down to one of either DETR or YOLOS. Neither model visibly stood out in terms of their performance metrics with YOLOS-B and DETR-R101 respectively achieving mAP scores of 42.0% and 43.5% on COCO.

The decision to choose YOLOS over DETR was partially influenced by the tempting opportunity to compare two YOLO models that share both the same aim and the same principles but differ in almost every other single way. The ultimate decision, however, was dictated by the inference speeds of the two models with DETR only capable of running at ~20 FPS while YOLOS-Tiny ran at ~30 FPS.

4.2.3.2.2 Variants of YOLOS

Model	Pre-train Epochs	Fine-tune Epochs	Eval Size	AP @ COCO val
YOLOS-Ti	300	300	512	28.7
YOLOS-S	200	150	800	36.1
YOLOS-S	300	150	800	36.1
YOLOS-S (dWr)	300	150	800	37.6
YOLOS-B	1000	150	800	42.0

Figure 7: The performance metrics of the different variants of YOLOS (Yuxin Fang, 2022).

YOLOS essentially provides three different variations: YOLOS-Tiny, TOLOS-S and YOLOS-B. Of these models, YOLOS-Tiny is the smallest and least accurate while YOLOS-B is the largest and most accurate. Unlike YOLOv8, YOLOS provides a much more distant range of models, meaning that there are large differences in both model accuracy and inference speed between each variation. This makes the decision of choosing the right model more difficult as the trade-offs from one model to the next are fairly substantial.

I experienced this issue first hand when I had to downgrade from YOLOS-S to YOLOS-Tiny in order to suitably meet the requirement for real-time inference speeds. This change allowed for much greater inference speeds from YOLOS-S which achieved ~12 FPS on average to YOLOS-Tiny which consistently provided a throughput of ~30 FPS. The downside of this substitution was the subsequent trade-off with accuracy which noticeably dropped.

4.2.3.2.3 Limitations of YOLOS

YOLOS typically achieves high recall scores while recording substantially lower precision scores. This means that it has successfully detected the majority of the 'ground truth' detections but has generated a lot of false positive predictions in doing so. In my opinion, this characteristic is attributable to its scattergun tactic of making 100 predictions per image and assigning overly high confidence scores to these predictions. In order to overcome this problem, I will set the confidence threshold at a high level so to filter out some of the false positive results.

YOLOS also suffers from having crowded bounding boxes that are attempting to encompass the same object. To reduce this issue, I will make use of Non-Maximum Suppression (NMS) with a low intersection over union (IOU) threshold so to reduce the number of overlapping bounding boxes.

4.2.3.2.4 YOLOS's Architecture

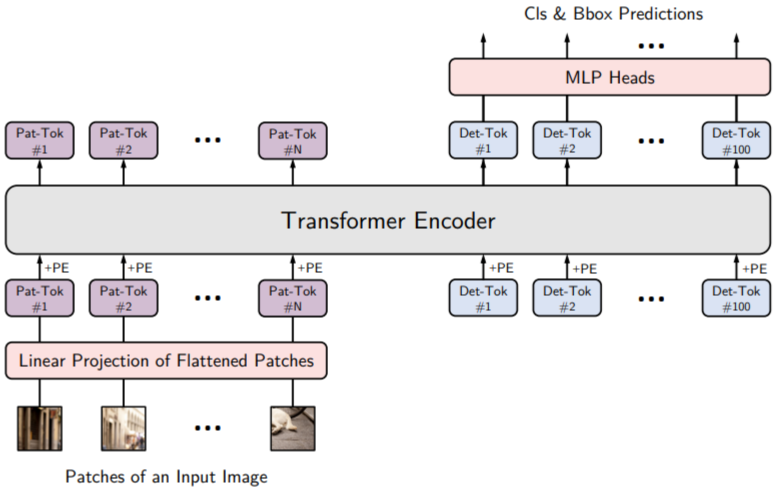


Figure 8: A depiction of the YOLOS architecture (Liu, 2021).

The design of YOLOS is deeply inspired by both DETR and the original Vision Transformer (ViT) architecture.

Essentially, the difference between the ViT and YOLOS architectures originates from the tasks that the architectures are trying to resolve. The ViT was designed to tackle the problem of image classification while YOLOS aimed to introduce transformer technology into the field of object detection. As a result, the subsequent differences in the architectures from ViT to YOLOS are made with the intention to optimise performance for object detection and not image classification.

To do so, YOLOS drops the CLS token for image classification and appends one hundred randomly initialized learnable detection tokens (DET tokens) to the input patch embeddings for object detection. A similar alteration is made during training, whereby, YOLOS replaces the image classification loss in ViT with the bipartite matching loss to perform object detection in a set prediction manner.

While YOLOS is inspired by DETR, YOLOS does differ from DETR in a number of areas:

- DETR adopts a Transformer encoder-decoder architecture, while YOLOS chooses an encoder-only architecture.
- DETR only employs pre-training on its CNN backbone but leaves the Transformer encoder & decoder to be trained from random initialization, while YOLOS naturally inherits representations from any pre-trained Vision Transformer (ViT).
- DETR applies cross-attention between encoded image features and object queries with auxiliary decoding losses deeply supervised at each decoder layer, while YOLOS always looks at only one sequence for each encoder layer, without distinguishing PATCH tokens and DET tokens in terms of operations (Liu, 2021).

4.3 Criteria for Success

As with any computer vision task, the ideal aim for the object detection models that I implement is to replicate if not outperform the human ability to detect debris, marshals, and recovery vehicles. In tangible terms, this means achieving high mAP scores across the three categories of objects.

Given that most object detection models are benchmarked on the COCO dataset, which contains images of humans and a range of vehicles, I expect that the models that I implement will perform well in terms of the detection of marshals and recovery vehicles, achieving a mAP50 score of 0.8 or more for these categories. As debris is a much broader concept than that of marshals and recovery vehicles, I anticipate that the models will have the greatest struggle detecting debris correctly. In accordance with this belief, I hope that the models can achieve a mAP50 score of at least 0.5 for the debris category.

The ultimate success of this project will be determined by whether the produced object detection models adequately perform in the environment in which Formula One operates. The debris detection models must therefore:

- Function in real-time.
- Function in a range of weather conditions both dry and wet.
- Function in a range of different lighting conditions (sunny, cloudy, night/dark with flood lights, etc.).
- Function in the occurrence of blurred objects caused by high speed.
- Be capable of detecting fragments of debris that vary in size and shape.

The models' ability to run in real-time will be judged by the inference speed and frames per second rate that they achieve when running on my laptop's NVIDIA GeForce RTX 3070 GPU. The other conditions listed above will be judged by the mAP scores that the models achieve.

5. Implementation

5.1 Data

5.1.1 Collection

As intended, I captured the on-board perspective of multiple drivers across multiple laps for a variety of different incidents in which debris, marshals or recovery vehicles were present on the track. To source this footage, I used F1TV's archive which stores full race replays from every season dating back to the 1981 season in addition to full replays of every Grand Prix session from 2018 to the current day. From this archive, I gathered footage from nine different incidents spanning across five different races from the 2021 Formula One season. For each incident, I captured the on-board perspective of five different drivers from the causation of the incident through to the return to a clear track. This process was very simple but somewhat time consuming.

5.1.2 Pre-process

Once collected, I trimmed the gathered footage into condensed clips, using a video editing software called Clipchamp, so to remove the sections of the lap where no objects of interest were present. I used the same software to reduce the quality of the resulting video so to further improve inference speeds later on.

5.1.3 Annotation

After pre-processing the data, the next step was to annotate the data so to produce a set of ground truth detections for the models to be trained, tested, and validated on. To do this, I made use of an application called VoTT (Visual Object Tagging Tool). VoTT is an image annotation and labelling software tool developed by Microsoft used for building end to end object detection models from image and videos assets for computer vision algorithms. I used this software tool to draw bounding boxes around instances of debris, marshals, and recovery vehicles that I was able to detect in each frame.

This task was challenging in a number of ways, with the biggest issue, that I suffered from being consistency. From one image to the next, it was extremely difficult to determine the threshold for when an object was too far out of sight and subsequently unrecognisable and a what stage the object was close enough so to be deemed recognisable. The issue with distant objects was persistent and caused me to question whether I should annotate an object because in several frames time it is revealed to be an object of interest or whether I should leave it as it could potentially confuse the model.

I found this task to be the most challenging aspect of this project as the process of drawing the bounding boxes was extremely faffy and the standard to which I performed this task ultimately dictated the quality of the data and thus determined how well the models performed.

Over the course of this research project, I would say that I attributed ~60% of my time to this task alone. While I referenced a survey completed by CloudFlower in 2016 (Press, 2016), in the proposal for this project, I did not anticipate that findings of that survey, that stated that the average data scientist spends 60% of their time cleaning and organising data and 19% of their time collecting data, would hold so true.

In total, I successfully annotated 5,777 images consisting of 7,716 instances of debris, 5,701 instances of marshals and 1,435 instances of recovery vehicles.

Category	No. Annotations
Debris	7,716
Marshals	5,701
Recovery Vehicles	1,435
Total	14,852

5.1.4 Creation of Dataset

After making a total of 14,852 unique annotations across 5,777 images, I exported the annotations from VoTT and uploaded the annotations to a website called Roboflow. By uploading the annotations to Roboflow, I could:

- Split the data into training, testing and validation datasets with any split I so desired.
- Specifically select which data went in which dataset.
- Go back over the uploaded annotations and add/remove any annotations.
- Expand the training dataset by up to 3 times by adding additional generated images using data augmentation.
- Easily import the dataset to Google Colab with a generated code snippet.
- Export the dataset to a wide range of common image annotation formats.

On top of this, the platform allowed the generation of multiple versions of datasets whereby I could experiment with different split ratios and different augmentations. Overall, the use of this platform made the creation and usage of the dataset incredibly easy.

5.1.5 Data Augmentation

One particularly useful feature that Roboflow provided was its data augmentation feature that allowed the expansion of the training dataset by a maximum of 3-fold through the addition of several slightly modified copies of existing data within the training dataset. As part of this feature, I could select which augmentations I wanted to apply in addition to determining the extent to which each augmentation was applied. The augmentations chosen are illustrated on the following page.

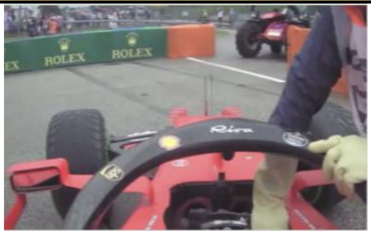
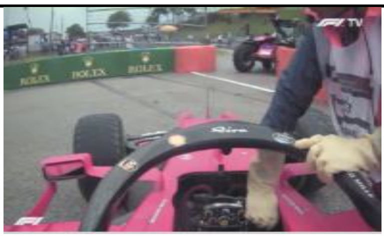
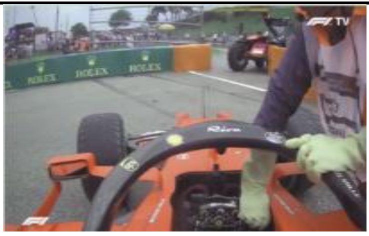
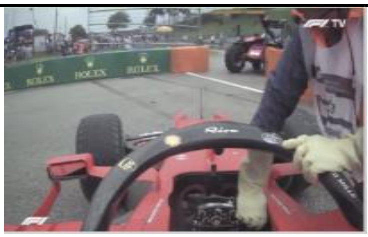
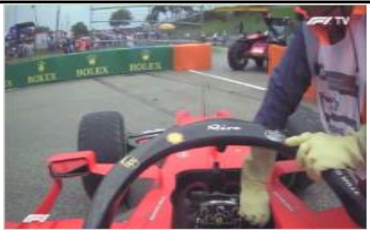
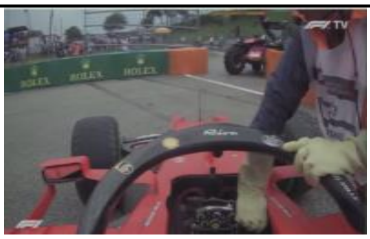
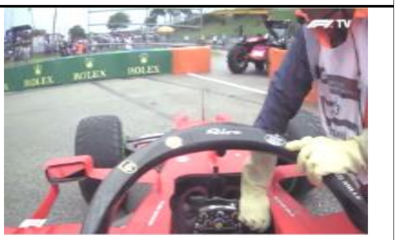
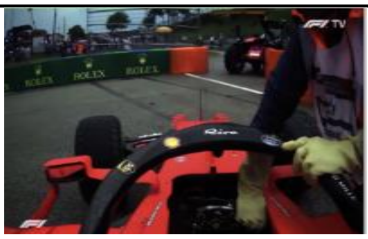
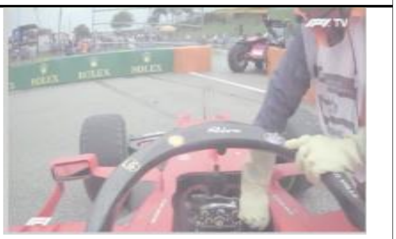
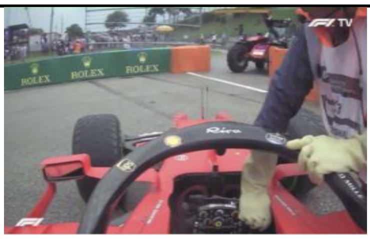
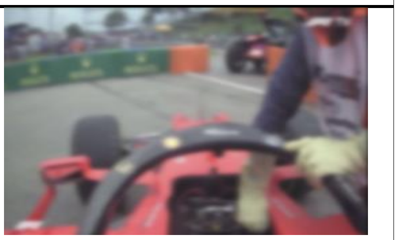
Augmentation	Min	Max
Crop	 0% Zoom	 20% Zoom
Hue	 -20%	 +20%
Saturation	 -25%	 +25%
Brightness	 -25%	 +25%
Exposure	 -25%	 +25%
Blur	 0px	 2px
Bounding Box: Blur	0px	2px

Figure 9: Applied data augmentations.

5.2 Architectures

Both the YOLOv8 and YOLOS architectures were initialised, trained, and evaluated in a Google Colab environment so to utilise its superior storage and GPU capacities. The two models make use of the exact same data previously referred to by importing the created dataset via the generated code snippet provided by Roboflow. The importation of this data is the first phase of implementation in both notebooks having successfully setup the environment with all required libraries installed.

5.2.1 YOLOv8

YOLOv8 Training.ipynb trains, evaluates, validates, and optimises the medium-sized variant of the YOLOv8 model range.

For each variant of YOLOv8, Ultralytics provide both a pre-trained (.pt) and base (.yaml) version. The pre-trained variant has been trained on the COCO dataset and as such is trained to detect the various 80 object categories within the COCO dataset. The base version as the name suggests has not been trained and is used for situations such as this when using a custom dataset.

Through the use of the Ultralytics Python package, the process of initialising the YOLOv8m model was incredibly simple, just requiring a single line of code and a reference to the desired variant of YOLOv8, which in this case is 'YOLOv8m.yaml'.

5.2.1.1 Training YOLOv8

As with the initialisation of the model, the process required for training was equally simple to execute. To train the model, I called the train function on the initialised YOLOv8m model and specified any specific hyperparameters that I desired.

At this early stage in the process of creating a YOLOv8 model to detect instances of debris, marshals, and recovery vehicles, I used the default hyperparameters over 100 epochs to set a benchmark performance for later on. To ensure reproducibility of this phase, I used a seed.

After each epoch in training, the model is evaluated on the validation dataset and performance metrics such as box loss, mAP50, and mAP50-95 are calculated to illustrate the progress of the training.

YOLOv8 Training Hyperparameters	
Hyperparameter	Value
Epochs	100
Batch Size	16
Optimiser	SGD
Initial Learning Rate	1e-2
Final Learning Rate	1e-2
Weight Decay	5e-4

Figure 10: Default YOLOv8 hyperparameters.

5.2.1.2 Evaluating YOLOv8

To evaluate the trained YOLOv8 model, I initialised a new model with the best weights from training before calling the val function on the trained model. The val function takes a model and evaluates it on a validation set to measure its accuracy and generalisation performance. In addition to the metrics provided after each epoch, the val function calculates the mAP50 and mAP50-95 scores for all object categories in addition to an average across all three categories.

The same val function generates additional evaluation metrics such as performance curves from training and a confusion matrix among others. The ability to generate fundamental evaluation metrics is very simple through the use of this function, however, there is no option to customise or tweak any of these metrics to your preferred format or style.

5.2.1.3 Validating YOLOv8

Another feature stemming from the val function that I made use of was the ability to take a batch of images and display both the ground truth detections and the corresponding model's predicted detections. Through this approach, it was easy to determine on an observational level as to how well the model had performed by comparing the two batches (that contained the same images).

To further test the extent to which the model had performed sufficiently, I ran inference on a random image from the validation dataset to compare and contrast the original plain image with no annotations against the image containing the model's predicted detections.

5.2.1.4 Hyperparameter Tuning YOLOv8

Having fully trained, evaluated and validated the YOLOv8m model using the default hyperparameters, I now wanted to optimise the performance of the model and subsequently find the best hyperparameters to achieve this. This task is called hyperparameter tuning and involves using either a random or selective choice of possible values for hyperparameters and trialling various combinations, noting the values that achieve the best results.

To implement this, I used an automatic hyperparameter tuning software framework called Optuna that is specifically designed for machine learning and can be used alongside other frameworks such as PyTorch.

To implement hyperparameter tuning with Optuna, an objective function is fed into a study for optimisation with the goal of finding the optimal set of hyperparameter values through several trials. Note that a trial is simply a single execution of the objective function. Here the objective function returns metrics that the study can use to evaluate the various trials. In this case, I used mAP50 and mAP50-95 as the corresponding metrics. The direction of the study is then used to determine whether the study aims to maximise this objective or minimise it, which in this case was to be maximised.

The objective function is also used to initialise the hyperparameters to be tuned and define the range of values that are to be trialled for each hyperparameter. For the YOLOv8m model, I chose four fundamental hyperparameters to tune with the following range of values:

- Batch Size (min: 8, max: 32)

- Optimiser ("SGD", "Adam")
- Initial Learning Rate (min: 1e-5, max: 1e-1)
- Final Learning Rate (min: 1e-5, max: 1e-2)

The objective function that I created for this task trains an initialised YOLOv8m model with generated hyperparameters that are chosen from the defined ranges above and returns the evaluation metrics mAP50 and mAP50-95 after 20 epochs of training on a smaller-scale dataset (containing just 2504 images across the training, testing and validation datasets).

The resulting study aimed to find the combination of hyperparameters that achieved the best mAP50 and mAP50-95 scores after 20 trials (20 executions of the objective function). After 20 trials, the trial that achieved the highest mAP50 and mAP50-95 scores is noted and the hyperparameters used for that trial can be retrieved by calling *study.best_trials*.

After completing the trials, I used some of the visualisation tools provided by Optuna to visualise the significance of the various hyperparameters. This included a hyperparameter importance plot, a slice plot, a pareto-front plot and an optimisation history plot.

After the best hyperparameters were found, these values were then used to train an initialised YOLOv8m model over 100 epochs so to compare the performance of the model with default hyperparameters against the model with optimised hyperparameters. Once this was completed, the same evaluation and validation process as referred to in section 5.2.1.2 and 5.2.1.3 were carried out.

5.2.2 YOLOS

YOLOS Training.ipynb trains, evaluates, validates, and optimises the tiny-sized variant of the YOLOS model range. Unlike YOLOv8, YOLOS was not so neatly packaged and subsequently required a lot of work in order to implement simple tasks.

The suggested way to implement YOLOS was by creating a PyTorch dataset and corresponding dataloaders with an added feature extractor so to transform the data (which is in COCO format) into the format and size that YOLOS expects (DETR format). The result of this transformation was the creation of the training, validation and testing datasets that were in the required format. A collate_fn is then defined to batch images together before the dataloaders are created making use of the collate_fn. Note collate_fn is a function used to process the batch you want to return from your dataloader.

In this preparation phase, the image processor and feature extractor must be defined and to do so AutoImageProcessor and AutoFeatureExtractor are respectively called both with the checkpoint to the chosen model which in this case is 'hustvl/YOLOS-Tiny'.

With the completion of these tasks, the majority of the awkward and confusing code is out of the way.

5.2.2.1 Training YOLOS

The training process was much simpler and required just four steps to implement. The first step was to define the model and load the YOLOS-Tiny model by calling AutoModelForObjectDetection using the same 'hustvl/YOLOS-Tiny' checkpoint as before. After this, the training hyperparameters are defined in TrainingArguments with the following values:

YOLOS Training Hyperparameters	
Hyperparameter	Value
Epochs	15
Batch Size	16
Optimiser	AdamW
Learning rate	1e-5
Weight Decay	1e-4

Figure 11: Common hyperparameters used to train initial YOLOS model.

The training arguments are then passed to the trainer along with the model, dataset, and data collator. Calling trainer.train() commences the process of training.

5.2.2.2 Evaluating YOLOS

Existing suggestions online to evaluate transformer models were once again needlessly complex involving downloading pre-defined evaluators that did not produce the metrics I desired and so I decided to implement the evaluation from scratch.

To do this I took the validation dataset and stored each image's actual detections in a list. This list would represent the ground truth. I then somewhat replicated this with predicted detections over a range of IOU values so to have accompanying lists of predicted detections. With a list of ground truth detections and predicted detections for the validation dataset, I could then compare the two lists and calculate mAP values.

The evaluation iteratively loops through each IOU threshold from 0.95 to 0.5 (with a step of 0.05) and displays the predicted labels above the actual labels before printing the precision and recall for all object categories including an overall average across the categories. Once completed, the evaluation takes the precisions and recalls across all the IOU thresholds and calculates an average so to calculate mAP50-95.

I then implemented a confusion matrix by iteratively looping through each image in the validation dataset and individually compared the actual detections against the predicted detections. I viewed this as a set problem and deemed matching detections from the list of actual detections and predicted detections to be true positives. Once matching detections were paired off, I then compared the remaining detections and paired them with a fourth category referred to as 'background'. Any remaining detections found in the actual list of detections represented false negatives that the model had not predicted, while any remaining detections found in the predicted list of detections represented false positives that the model had predicted but were not actual detections.

As I had no input into the design and layout of the confusion matrix produced for YOLOv8, I tried to design the confusion matrix for YOLOS in a similar style so to easily be able to compare the two diagrams later on.

5.2.2.3 Validating YOLOS

To validate the performance of the YOLOS model, I selected a random image from the validation dataset and displayed the original image, the actual detections, and predicted detections for that image, using a confidence threshold of 0.7 and an IOU threshold of 0.5 to produce this visualisation.

In addition to this, I thought I would illustrate the effect of changing the confidence threshold and subsequently displayed four different predictions for the same image using confidence thresholds of 0.0, 0.5, 0.7 and 0.9 using an IOU threshold of 0.5. I replicated this visualisation with IOU thresholds to illustrate the effect that the IOU threshold has on the outcome.

5.2.2.4 Hyperparameter Tuning YOLOS

Having fully trained, evaluated and validated the YOLOS-Tiny model using common hyperparameters, I now wanted to optimise the performance of the model and subsequently find the best hyperparameters to achieve this. Once again, I made use of Optuna to implement this, only this time, aiming to find the best values for just the learning rate and batch size.

The process of hyperparameter tuning YOLOS differed slightly from the process involved for hyperparameter tuning YOLOv8 but essentially involved the same steps. To implement this, I defined the following hyperparameters to be tuned and defined the range of values to be trialled for each hyperparameter:

- Batch Size (8, 16, 32)
- Learning Rate (min: 1e-6, max: 1e-4)

I then defined a function to return an initialised YOLOS model before defining the training arguments excluding the hyperparameters to be tuned. I fed both these into the trainer along with a small-scale dataset and the data collator. To commence hyperparameter tuning, I called trainer.hyperparameter_search and defined the objective for the search. Here, I decided to run 20 trials with aim of finding the trial with the smallest validation loss after 5 epochs.

Once again, after the hyperparameter search had finished, I reused the values from the best trial and trained an initialised YOLOS-Tiny model over 15 epochs using those values. The same evaluation and validation methods were used excluding the illustration of how the confidence and IOU thresholds affected the results.

5.3 Setting Up Hardware Optimised Environment

Every aspect of this project's implementation was completed in Python. To allow for detections in real-time, the Python environment used had to be optimised so to make the best possible use of the hardware available. This meant ensuring that the Python scripts created ran on my laptop's GPU whenever appropriate. To do this, I installed PyTorch with CUDA. CUDA is a parallel computing platform and programming model developed by NVIDIA for computing on graphical processing units (GPUs). Computing applications that make use of CUDA, and subsequently harness the power of available GPUs, observe substantially quicker execution times (NVIDIA, 2023).

To further optimise the Python environment for faster inference speeds, I did attempt to install OpenCV's dnn module, which supports deep learning inference on images and videos, in addition to NVIDIA's TensorRT, a high-performance inference optimiser and runtime that delivers high throughput for deep learning inference applications. However, I was ultimately unable to utilise these tools due to the complex nature of the respective installation processes. This issue highlighted the significant roles that both hardware and hardware optimised code play in order to achieve real-time environments for deep learning.

In the end, PyTorch with CUDA proved to be sufficient enough by itself to run a number of prospective models at real-time or close to real-time inference speeds when running on an NVIDIA GeForce RTX 3070 GPU. The inability to utilise the additional hardware optimisation tools is likely one of several reasons as to why certain prospective models were unable to achieve real-time inference speeds and subsequently as to why they were ultimately not selected. It is likely that models that I ruled out due to insufficient inference speeds could, with the right hardware and hardware optimisation tools, run at the desired throughput, however, I was unable to implement and subsequently prove this throughout this research project.

5.4 Real-Time Inference

Having successfully created a Python environment suitable for real-time detections, I constructed a Python script for each model to run real-time inference on a given input video. To do this, I used OpenCV to read the given input video file frame by frame and iteratively fed the current frame into the models for inference before displaying the final result with any detections in a separate window.

The model's predictions are displayed in the separate window as labelled annotations with accompanying confidence scores overlayed on top of the original frame. The ability to translate the models' predictions into the vibrant annotations is owing to the supervision Python package.

In order to record the performance of each model, I used a number of timers and counters throughout both scripts to make note of the models' throughput (Frames Per Second) and inference speed.

As a default, the confidence thresholds for YOLOv8 and YOLOS are respectively set at 0.5 and 0.9 due to the vastly different nature in which confidence scores are calculated as referenced in section 4.2.3.2.3 of this report.

6. Results and Evaluation

6.1 Evaluation Methodology

While the implementations of YOLOv8 and YOLOS differ in a number of areas, both ultimately produce results that evaluate the model using the same metrics and confusion matrix in addition to providing an example of inference. The evaluation metrics that will be focused on are the mAP50 and mAP50-95 scores in addition to the confusion matrices. Any visual observations made from validation will be noted and will act as a qualitative metric.

6.2 YOLOv8 Results

6.2.1 Default YOLOv8 Model

Through using the default hyperparameters as referred to in section 5.2.1.1 the YOLOv8m model achieved respective mAP50 and mAP50-95 scores of 0.877 and 0.583 across all three categories of objects. The detailed breakdown of each category can be seen in the table below.

YOLOv8m Performance (Default)						
Class	Images	Instances	Box (P	R	mAP50	mAP50-95
Debris	976	1398	0.776	0.631	0.694	0.304
Marshal	976	959	0.924	0.931	0.96	0.654
Recovery Vehicle	976	191	0.958	0.969	0.978	0.792
All	976	2548	0.886	0.844	0.877	0.583

Figure 12: Performance of the default YOLOv8 model.

When compared to the criteria for success laid out in section 4.3, the YOLOv8m model using default hyperparameters surpasses initial expectations and as such can be deemed to be vastly successful. The performance of the
debris category, in particular, substantially outperforms what I believed to be possible given the somewhat undescriptive nature of debris.

In terms of performance, it appears as if the model could have further improved if allowed to continue training for more epochs. The improvement that could have been experienced, however, would be minimal given the nature of the various curves in figure 13 that clearly illustrate the various curves approaching asymptotes.

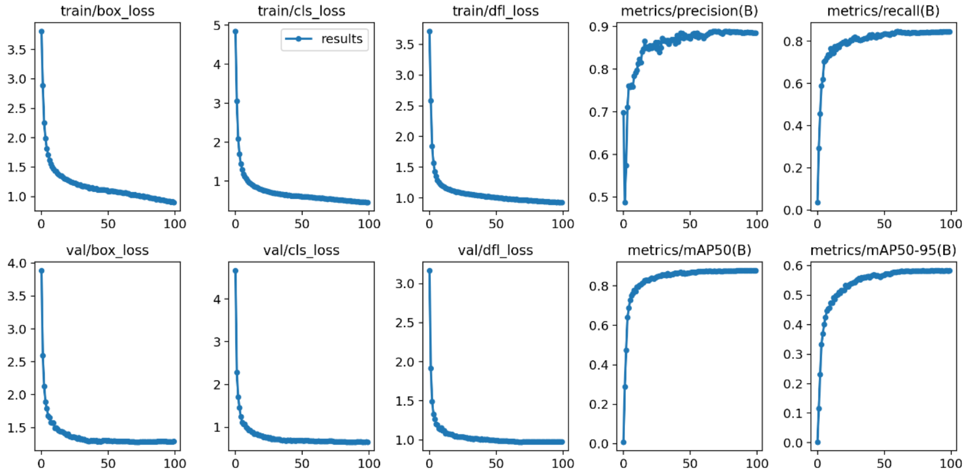


Figure 13: Performance curves from training the default YOLOv8.

The confusion matrix illustrated in figure 14 epitomises the success of the YOLOv8m model. The confusion matrix illustrates that 97% of all actual recovery vehicle detections were detected by the model with only a 2% error in incorrectly predicting objects to be recovery vehicles when in fact not. A similar story is told in the marshal category, however seemingly suffering from a higher frequency of predicting objects to be marshals when in fact not (27%).

The most notable observation from this visualisation is found with the debris category. Here, the model seemingly performs well, detecting 71% of the actual debris detections, however the model equally predicts a similar number of objects to be debris that are in fact not debris. The imprecision witnessed here is of mild concern, however, can be mitigated by using a high confidence threshold when the model performs inference so to limit the number of false positive results. I believe that the collection of more data and the inclusion of images where no objects of interest are present could be potential solutions to resolve this issue.

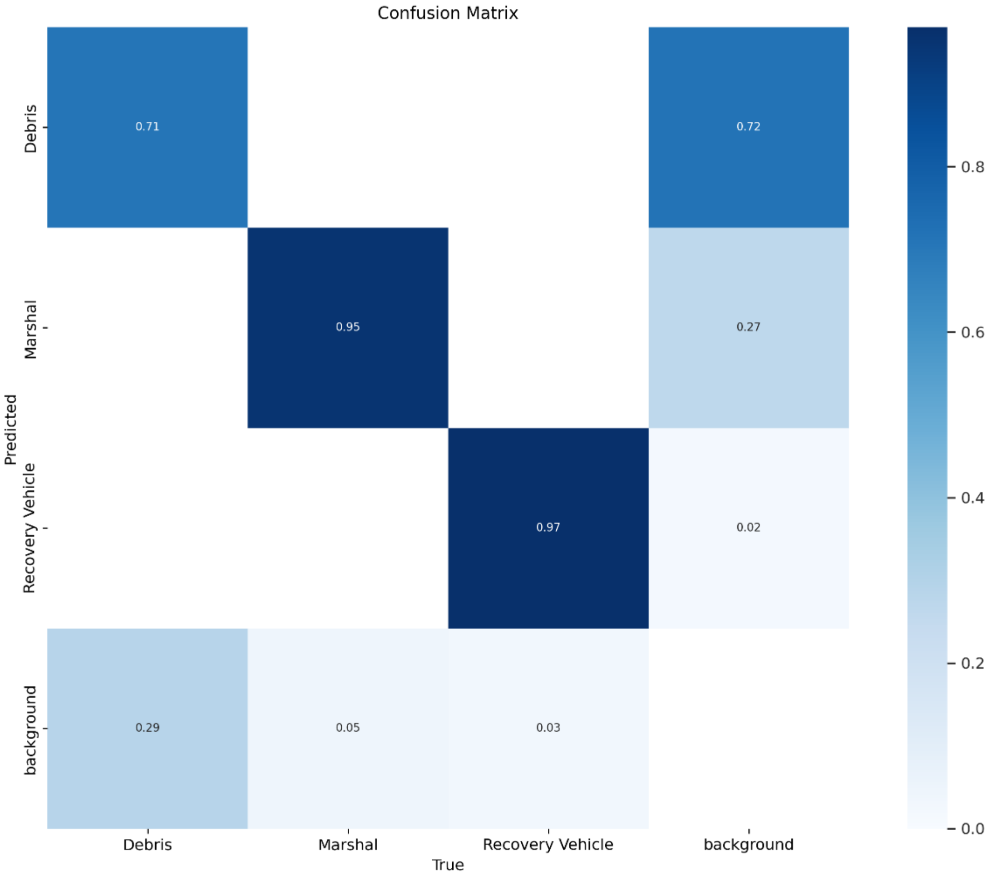
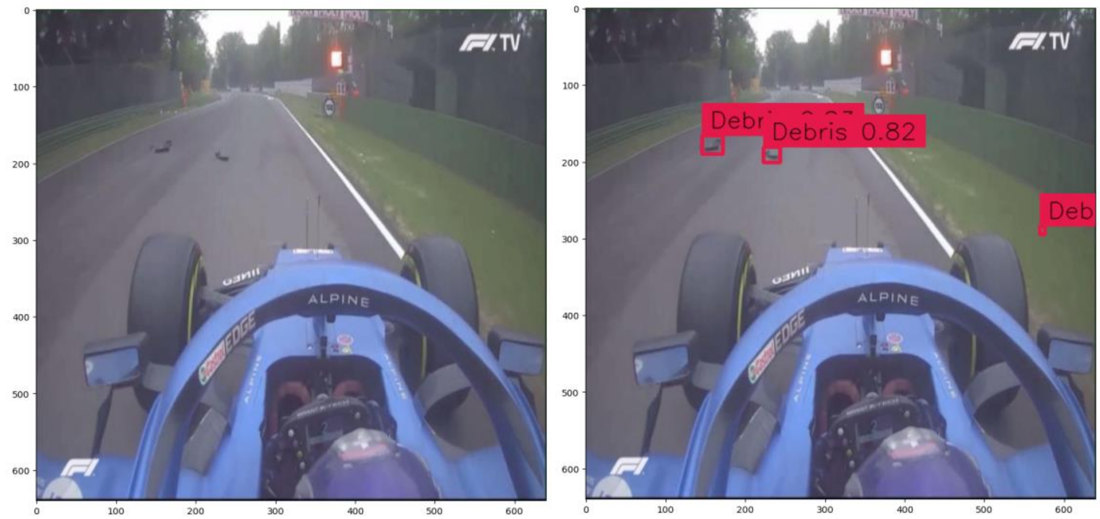


Figure 14: Confusion matrix for the default YOLOv8 model.

Figure 15 provides an insight as to the outcome of inference on a given image. Here, the main expectation is that the model is able to identify the two large chunks of debris on the track in the foreground, which the YOLOv8m model confidently detects. The model, in addition to this, picks up a substantially smaller piece of debris to the side of the track, which I find to be very impressive, however fails to detect some of the pieces of debris that can be found further down the road. This small imperfection is not of large concern as in practice the debris located further down the road would likely be detected when the car moved closer in that direction.

The bounding boxes are tight fitting and encircle the detected objects appropriately while the detections are made with a large level of confidence. As such, the results of inference are very promising.



Original Image

YOLOv8m's Detections

Figure 15: Running inference with the default YOLOv8 model.

Figure 16 provides an insight into the struggle the model experiences with marshals given the model's detection of an off-track marshal in the top left image while failing to identify a marshal stood directly in front of the car in the image second from the left on the bottom row. The debris, however, is generally detected to a good degree with exception of the top right image which seemingly misses a handful of detections. This could be a result of having comparatively few images in the training dataset being set in floodlit conditions.

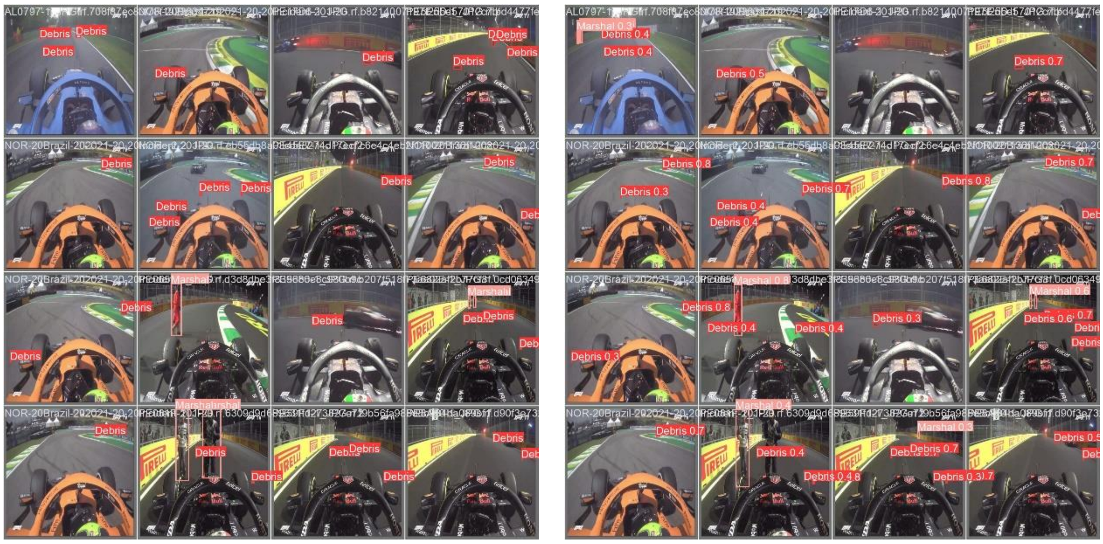


Figure 16: Default YOLOv8 validation.

6.2.2 Optimised YOLOv8 Model

The performances of the 20 different trials from the process of hyperparameter tuning can be seen in the optimisation history plot illustrated in figure 17. From this plot, it is clear to see that the process of optimising the hyperparameters did not provide continuous improvement to the mAP50 score, but instead illustrates a process of trial and error, taking on-board what worked and what did not. This is illustrated by the nine trials that achieved mAP50 scores close to zero after 20 epochs of training.

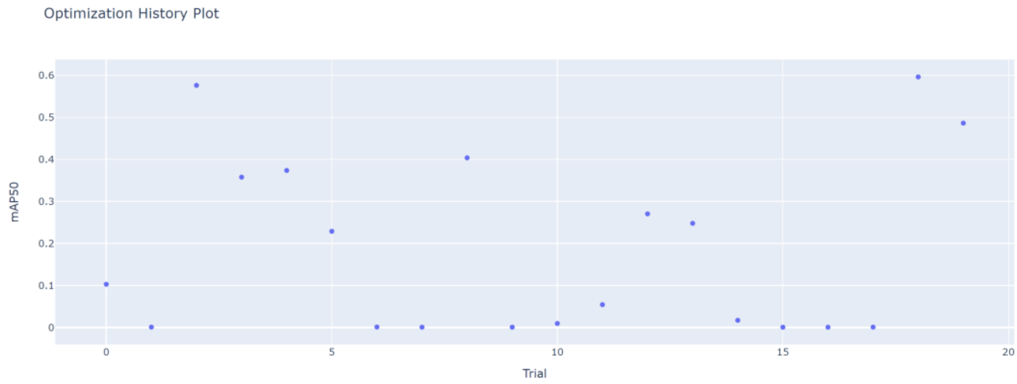


Figure 17: Optimisation history plot for hyperparameter tuning YOLOv8.

After 20 trials of hyperparameter tuning, the best hyperparameters were as follows:

YOLOv8 Training Hyperparameters	
Hyperparameter	Value
Batch Size	12
Optimiser	Adam
Initial Learning Rate	3.21e-4
Final Learning Rate	1.27e-3

Figure 18: The best hyperparameters for YOLOv8 after 20 trials of hyperparameter tuning.

One issue with these 'optimised' hyperparameters is that the final learning rate is larger than the initial learning rate which should not really be the case. Ideally the model should take large initial steps and then gradually take smaller and smaller steps as training progresses. By having a final learning rate larger than the initial learning rate, the model could converge too quickly and result in a suboptimal solution.

The importance of each hyperparameter and their impact on the resulting mAP50 score throughout the 20 trials is illustrated in figure 19. The larger the importance, the more significant impact changing that hyperparameter will have on the final mAP50 result. Here, we can see that the initial learning rate and the chosen optimiser are by some substantial distance the most significant hyperparameters. Figure 20 builds on this and provides an indication as to which hyperparameter values resulted in the best performance.

I believe that in order for these plots to truly provide a deep insight regarding the best hyperparameters for the YOLOv8m model more trials would be required to provide a clearer and more defined picture as to how to optimally setup the YOLOv8m model. While I do not believe the optimal solution can be found after just 20 trials, I was unable to run more trials due to limited time and computational resources.

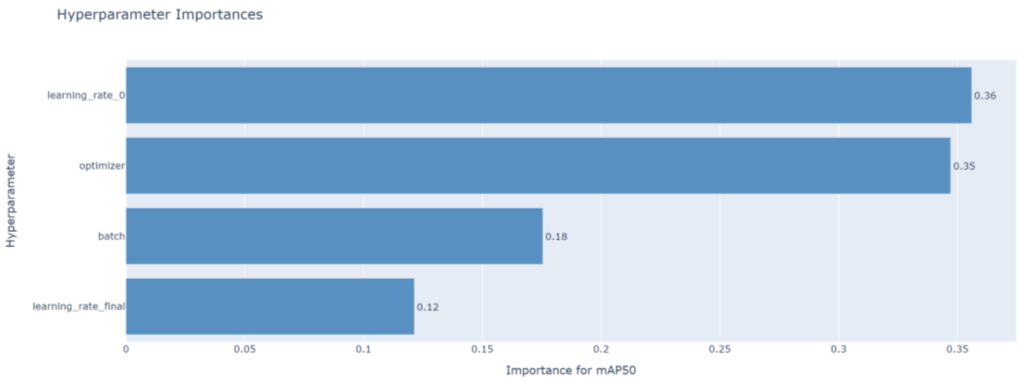


Figure 19: Hyperparameter importance plot for YOLOv8.

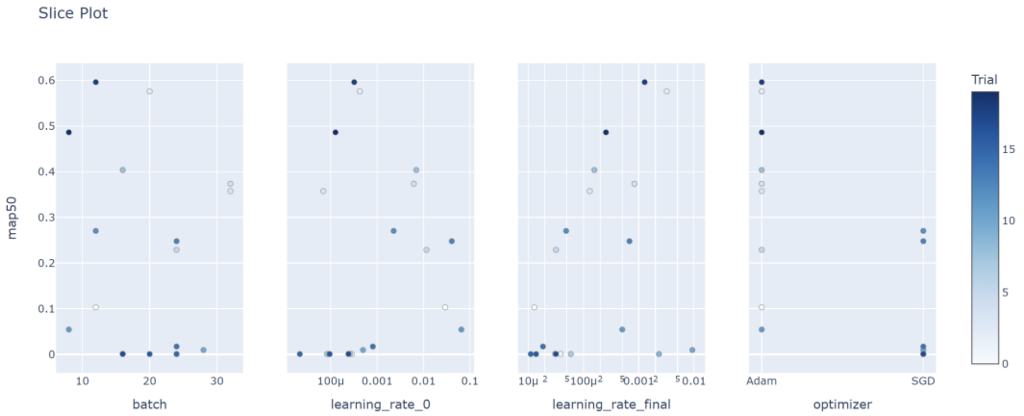


Figure 20: Slice plot for YOLOv8

Having completed 20 trials of various combinations of different optimisers, batch sizes and learning rates, the best trial performed ever so slightly worse than the default hyperparameters. Subsequently, I believe that more trials with perhaps a stricter range of possible learning rate values could likely outperform the produced 'optimised' model.

YOLOv8m Performance ('Optimised')						
Class	Images	Instances	Box (P	R	mAP50	mAP50-95
Debris	976	1398	0.771	0.609	0.686	0.299
Marshal	976	959	0.917	0.926	0.959	0.655
Recovery Vehicle	976	191	0.937	0.963	0.98	0.793
All	976	2548	0.875	0.833	0.875	0.582

Figure 21: performance of optimised YOLOv8 model.

The overall mAP50 score drops by 0.002 in comparison to the default model while the overall mAP50-95 drops by just 0.001. This slight drop in precision is experienced in every object category to very similar degrees. The drop off is not substantial at all, but the lack of improvement either indicates that the default hyperparameters are the best way to set the YOLOv8m model up, that the hyperparameter tuning did not manage to find the best combination of hyperparameters after 20 trials or that potentially there is some other limiting factor that limits the mAP50 score to ~0.875 after 100 epochs of training.

The performance curves of the 'optimised' model illustrated in figure 22 show comparable results to that of the YOLOv8m model using the default hyperparameters with no additional observations of note.

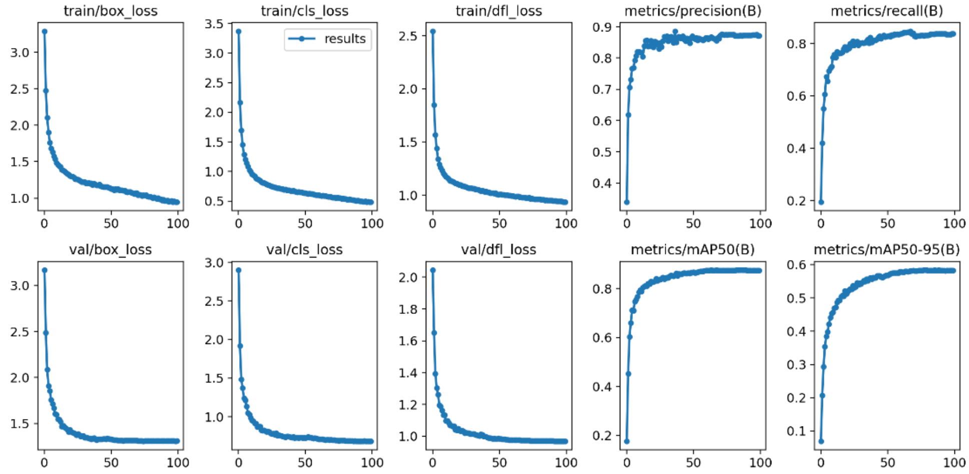


Figure 22: Performance curves for optimised YOLOv8 model.

The confusion matrix likewise shows little difference in the performance of the 'optimised' YOLOv8m model in comparison to the equivalent using the default hyperparameters. Once again, the small difference observed is a minute reduction in precision all-round.

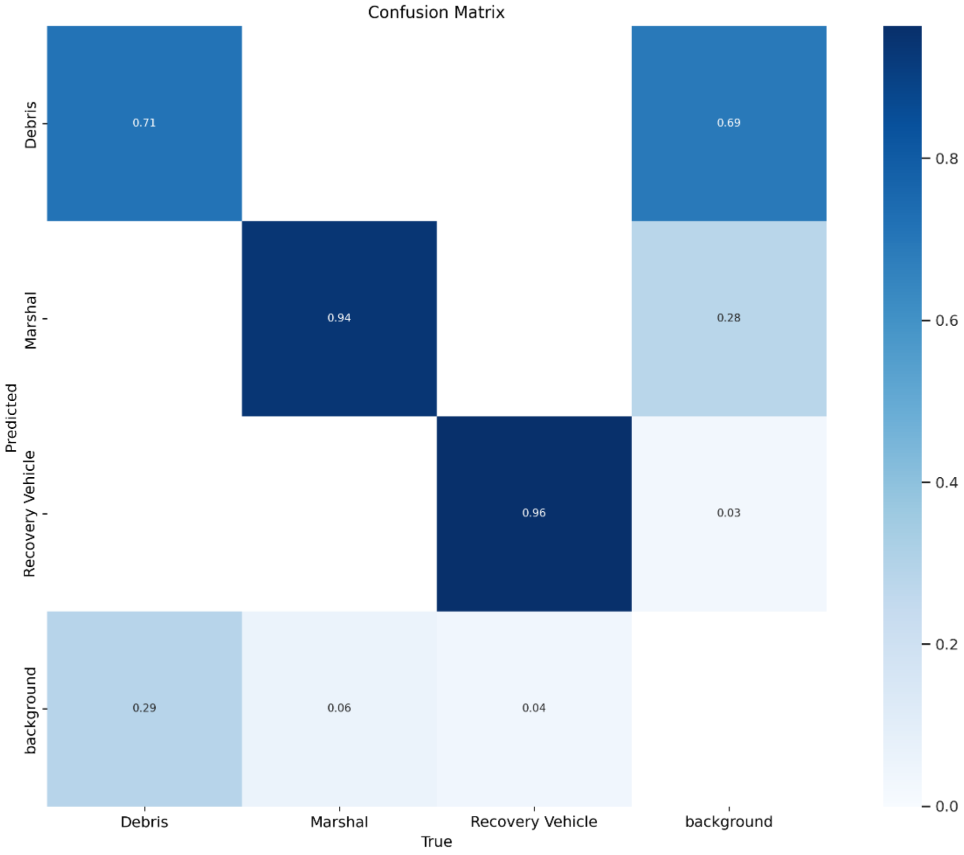


Figure 23: Confusion matrix for optimised YOLOv8 model.

As with the default model, the optimised model suffers from the same issues, as mentioned in section 6.2.1, when comparing the predictions of a batch of images against the corresponding actual detections.

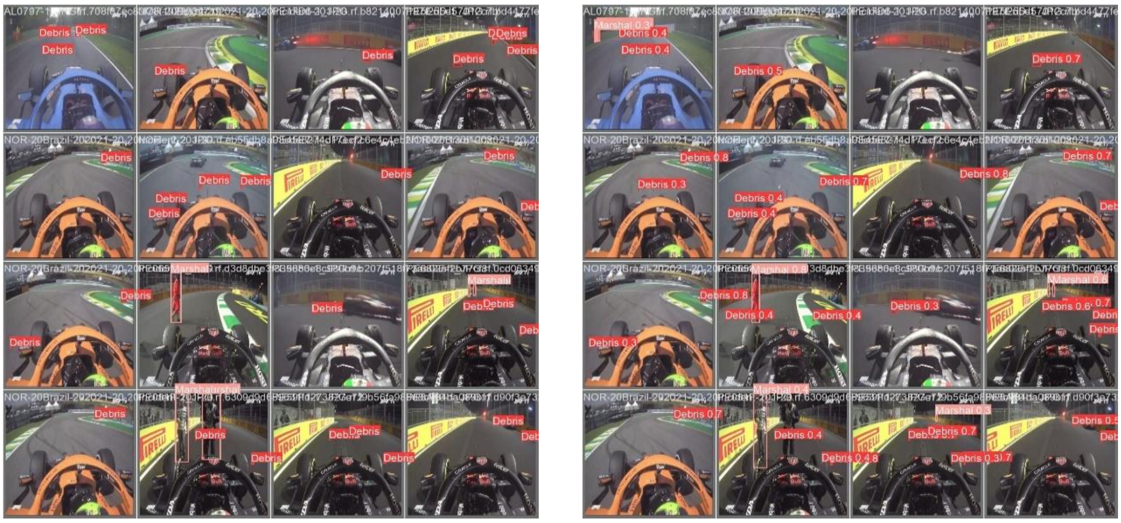
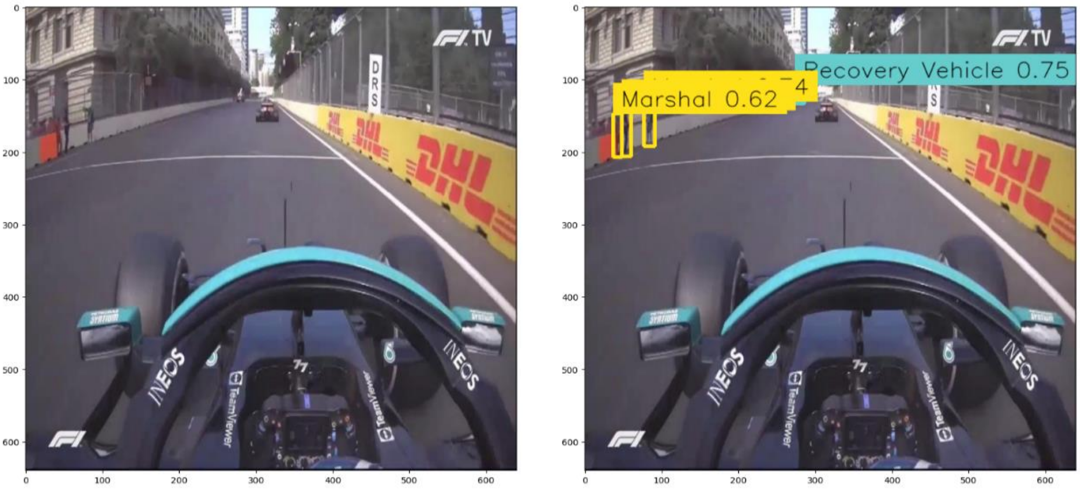


Figure 24:Optimised YOLOv8 validation.

The example of inference on an image as depicted in figure 25, highlights the continuing success of the YOLOv8m model as it successfully detects all 4 objects present in the image. Here the precision of the recovery vehicle bounding box is unknown due to the large text that occludes it, but from what is visible, it does appear to be accurately indicating the presence of a recovery vehicle in that region with a high confidence score. As a result, the tuned YOLOv8m model can be deemed to have successfully performed inference to a strong level.



Original Image

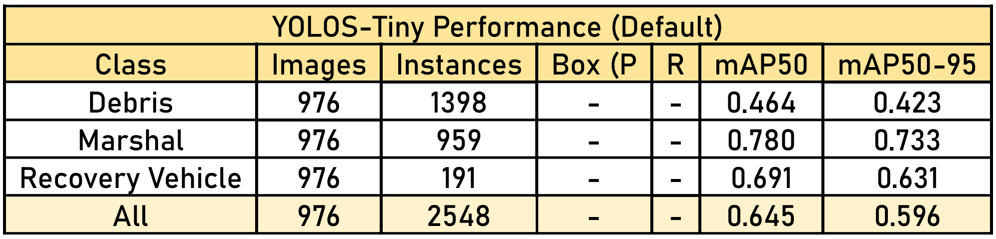
YOLOv8m's Detections

Figure 25: Running inference with the optimised YOLOv8 model.

6.3 YOLOS Results

6.3.1 Default YOLOS Model

Through using a combination of common hyperparameter values as referred to in section 5.2.2.1 the YOLOS-Tiny model achieved respective mAP50 and mAP50-95 scores of 0.645 and 0.596 across all three categories of objects. The detailed breakdown of each category can be seen in the table below.



YOLOS-Tiny Performance (Default)						
Class	Images	Instances	Box (P	R	mAP50	mAP50-95
Debris	976	1398	-	-	0.464	0.423
Marshal	976	959	-	-	0.780	0.733
Recovery Vehicle	976	191	-	-	0.691	0.631
All	976	2548	-	-	0.645	0.596

Figure 26: Performance of the default YOLOS model.

Based on the criteria for success laid out in section 4.3, the YOLOS-Tiny model using common hyperparameters fails to meet initial expectations and as such can be deemed to be somewhat unsuccessful. The performance of the recovery vehicle category is particularly disappointing given the more consistent appearance of recovery vehicles in comparison to the other two categories. This could be due to an unhealthy balance in the dataset that means there are 7,716 instances of debris across the three datasets but only 1,435 instances of recovery vehicles. Perhaps with more examples of recovery vehicles throughout the entirety of the dataset, the YOLS-Tiny model might perform better in this category.

One observation of note here is that the mAP50 and mAP50-95 scores are not too different from one another, indicating that the performance of the model is not very sensitive to changes to the IOU threshold.

The confusion matrix illustrated by figure 27 epitomises how YOLOS performs. The unmissable observation here is found in the debris category whereby the model detected 92% of all actual debris detections but predicted an additional 1,731 instances of debris compared to the actual number of detections. This illustrates how hypersensitive the model is and illustrates its scattergun approach. Similar levels of imprecision are found in the marshal and recovery vehicle object categories but to much lesser degrees. While using a high confidence threshold for inference could help reduce the extent of this issue, it is likely that a large number of false positive results could still be detected.

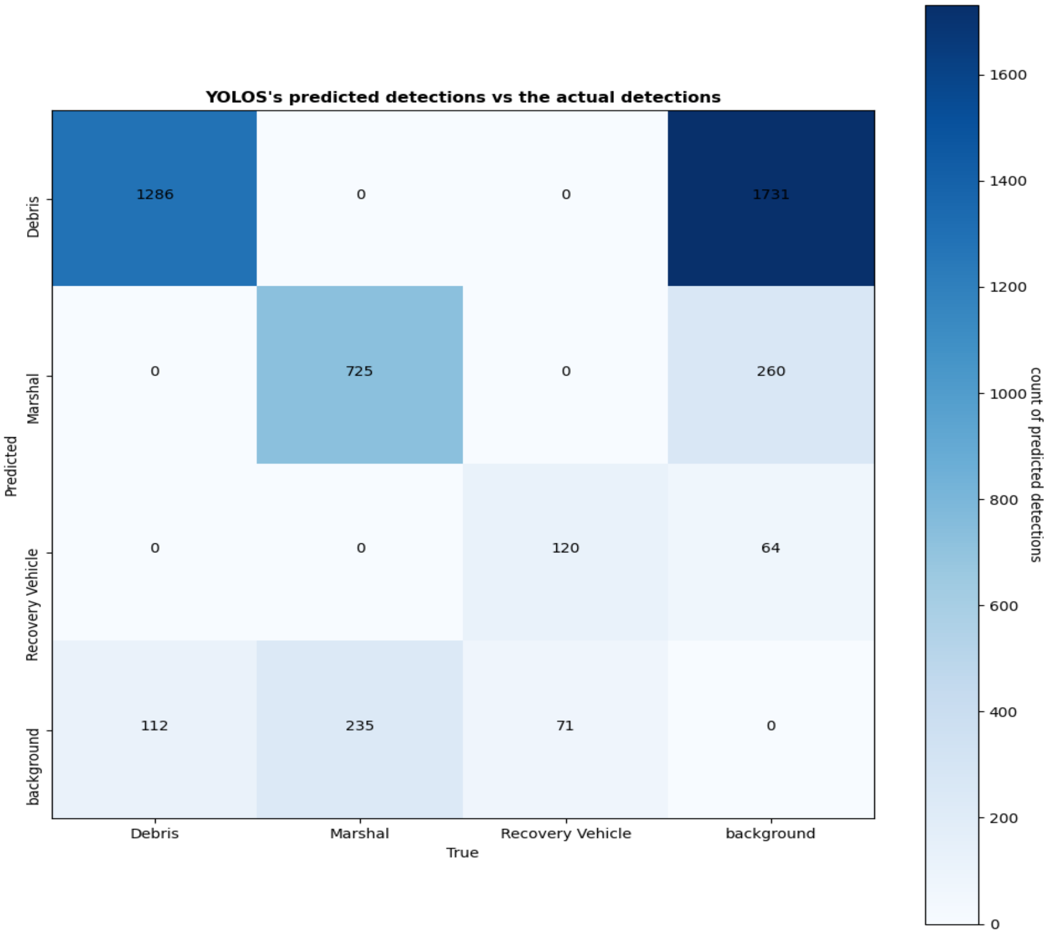
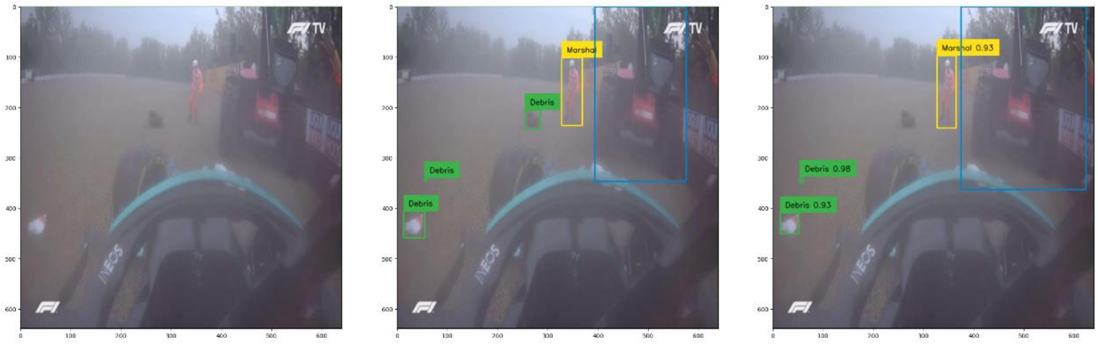


Figure 27: Confusion matrix for the default YOLOS model.

From figure 28, which illustrates YOLOS inference using a confidence threshold of 0.7 and IOU threshold of 0.5, it is clear to see the model's capability to make accurate detections. The YOLOS-Tiny model detects four of the five actual detections and does not detect any additional objects and subsequently can be deemed to have performed very well. The only slight concerns that arise from this validation is the failure to detect the piece of debris in the centre of the image and the somewhat loose-fitting bounding box that surrounds the recovery vehicle.



Original Image Actual Detections YOLOS's Detections

Figure 28: Running inference with the default YOLOS model.

It is worth noting that the piece of debris located in the centre of the image can be detected by the YOLOS model when the confidence threshold is reduced to 0.5. In contrast to this, when the threshold is increased to 0.9, the model no longer detects the recovery vehicle.

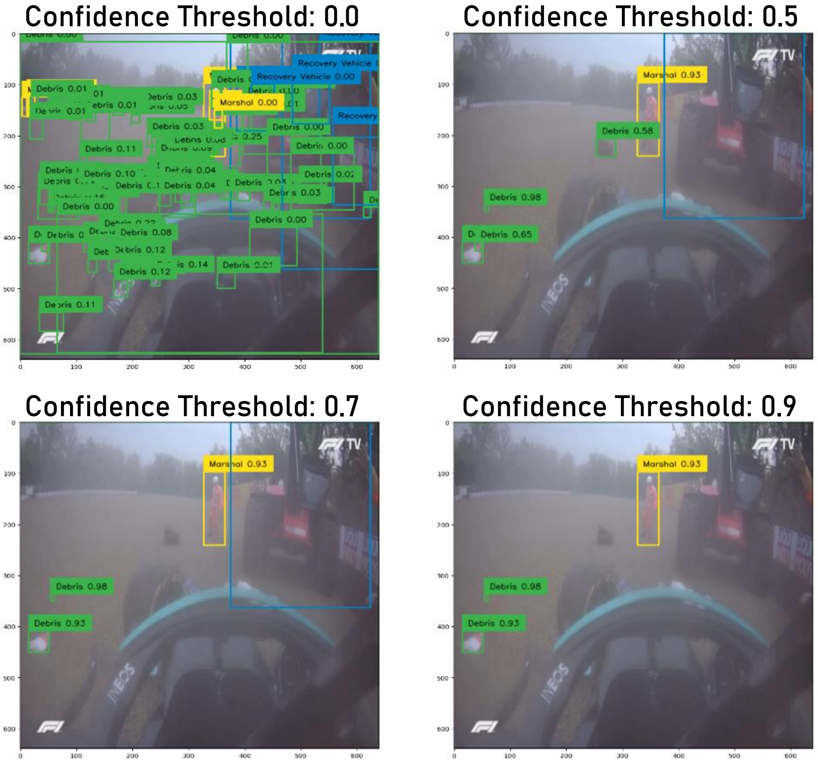


Figure 29: The effect of changing the confidence threshold on the results of YOLOS.

The effect of changing the IOU threshold can also be observed in *YOLOS Training.ipynb*.

6.3.2 Optimised YOLOS Model

After the completion of 20 trials the best hyperparameters were as follows:

YOLOS Training Hyperparameters	
Hyperparameter	Value
Batch Size	8
Learning Rate	3.536 e-05

Figure 30: Best hyperparameters for YOLOS after 20 trials of hyperparameter tuning.

The above values achieved the smallest validation loss of the 20 trials after 5 epochs of training and subsequently was deemed to be the best performing trial.

Having completed 20 trials of various combinations of different batch sizes and learning rates, the best trial performed substantially better than the YOLOS-Tiny model using

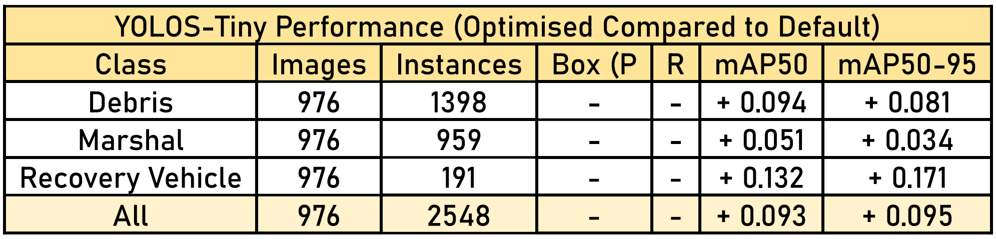
YOLOS-Tiny Performance (Optimised)						
Class	Images	Instances	Box (P	R	mAP50	mAP50-95
Debris	976	1398	-	-	0.558	0.504
Marshal	976	959	-	-	0.831	0.767
Recovery Vehicle	976	191	-	-	0.823	0.802
All	976	2548	-	-	0.738	0.691

Figure 31: Optimised YOLOS-Tiny performance.

common hyperparameters. If access to greater hardware was available, I would have liked to have optimised other hyperparameters, but this was not possible given the RAM usage required for the hyperparameter search. The detailed breakdown of each category can be seen in the table below.
Here, the optimised YOLOS Tiny model achieves an overall mAP50 score of 0.738 and an overall mAP50-95 score of 0.691. When compared to the criteria for success, the optimised YOLOS-Tiny model adequately surpasses the targeted mAP50 values for each category, and so can be deemed to be successful.

Compared to the default YOLOS-Tiny model, the overall mAP50 score improves by 0.093 while the overall mAP50-95 improves by 0.095. This significant improvement in precision is particularly experienced in the recovery vehicle object category whereby the mAP50 score improves by 0.132 and the mAP50-95 score improves by 0.171. This significant improvement in precision is remarkable and illustrates the effect that hyperparameter tuning can have on results.

The improvements from the default model to the optimised model can be seen in the table below.



YOLOS-Tiny Performance (Optimised Compared to Default)						
Class	Images	Instances	Box (P	R	mAP50	mAP50-95
Debris	976	1398	-	-	+ 0.094	+ 0.081
Marshal	976	959	-	-	+ 0.051	+ 0.034
Recovery Vehicle	976	191	-	-	+ 0.132	+ 0.171
All	976	2548	-	-	+ 0.093	+ 0.095

Figure 32: Comparing the performances of the default YOLOS and optimised YOLOS models.

The confusion matrix depicted in figure 33 further illustrates the differences between the two iterations of the YOLOS-Tiny model. Here, the optimised model surprisingly detects 102 fewer actual detections than the default model, but conversely detects 628 fewer false positive detections across all three object categories. This is a rather interesting trade-off to witness and asks the questions which of the two scenarios is preferable? While the mAP50 and mAP50-95 scores indicate that the optimised model is the superior of the two models, the confusion matrix somewhat indicates the reverse given the inferior recall ability. The choice as to which model is preferred is ultimately down to the various stakeholders, however, as the implementor I would in this case suggest that the default iteration of YOLOS-Tiny is superior.

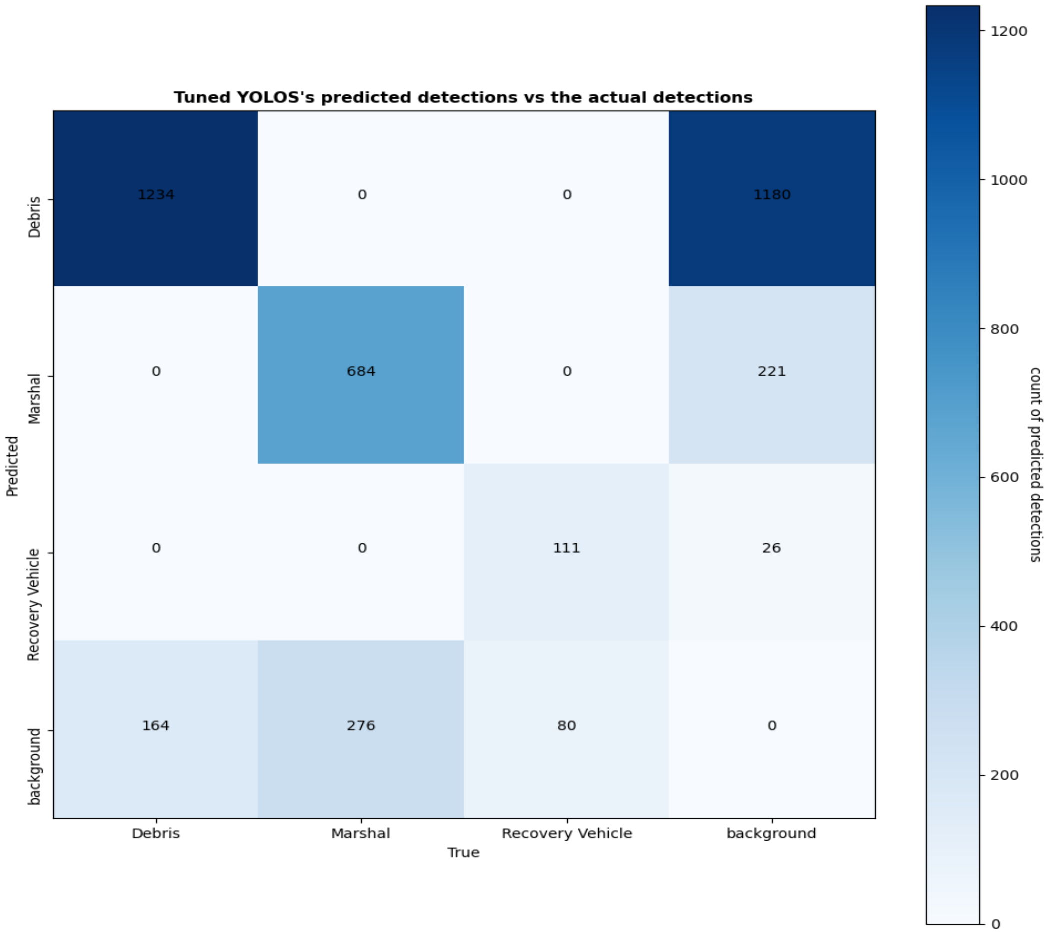
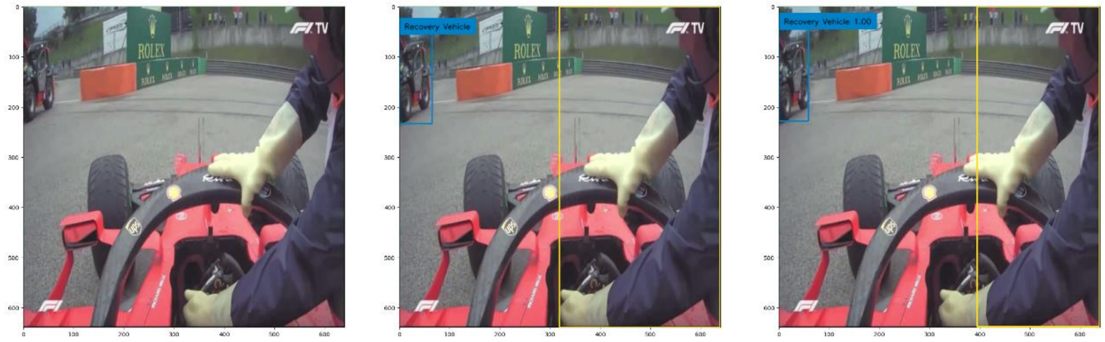


Figure 33: Confusion matrix for the optimised YOLOS model.

The example of inference below provides similar promising results as the default YOLOS-Tiny model, with the model making both detections with strong levels of confidence. The only concern here is the poor fitting bounding box around the marshal whereby the marshal's left hand is not encapsulated by the bounding box. Other than this, inference appears to perform remarkably well.



Original Image

Actual Detections

YOLOS's Detections

Figure 34: Running inference with the tuned YOLOS model.

6.4 Comparing YOLOv8 to YOLOS

Having established that the default models for both YOLOv8 and YOLOS achieved superior results in comparison to their optimised counterparts, it is now appropriate to compare and contrast the two different approaches.
To do this I will compare their evaluation metrics, compare the results of running inference on the same image and finally take a look at how both models perform when running inference in real-time on a video.

6.4.1 Evaluation

In terms of mAP50 scores, YOLOv8 vastly outperforms YOLOS as YOLOS achieves an overall mAP50 score of just 0.645 while YOLOv8 scores 0.877 in the same metric. Conversely, YOLOS seems to achieve a superior overall mAP50-95 score, achieving a score of 0.596 compared to YOLOv8 which only scored 0.583. The various performance metrics of both the YOLOv8 and YOLOS model are illustrated in the table below.

YOLOv8 vs YOLOS Performance						
			YOLOv8 (Default)		YOLOS (Default)	
Class	Images	Instances	mAP50	mAP50-95	mAP50	mAP50-95
Debris	976	1398	0.694	0.304	0.464	0.423
Marshal	976	959	0.96	0.654	0.780	0.733
Recovery Vehicle	976	191	0.978	0.792	0.691	0.631
All	976	2548	0.877	0.583	0.645	0.596

Figure 35: Comparing the performances of YOLOv8 and YOLOS.

With the overall mAP50 and mAP50-95 scores providing an inconclusive result as to which of the two models performed better, I will now evaluate both models' performances in each category and will base my final judgement on this evaluation.

The results depicted in the table above illustrate that YOLOv8 outperforms YOLOS indisputably in terms of the detection of recovery vehicles based on the substantially higher mAP50 and mAP50-95 scores achieved for the recovery vehicle object category by YOLOv8 in comparison to YOLOS. The confusion matrices in figure 36 confirm this theory as YOLOv8 produces more true positive results and fewer false negative and false positive results compared to YOLOS.

Conversely, the evaluation metrics for the debris and marshal object categories, depicted in the same table above, are inconclusive given that YOLOv8 achieves higher mAP50 scores in both categories while YOLOS achieves superior corresponding mAP50-95 scores.

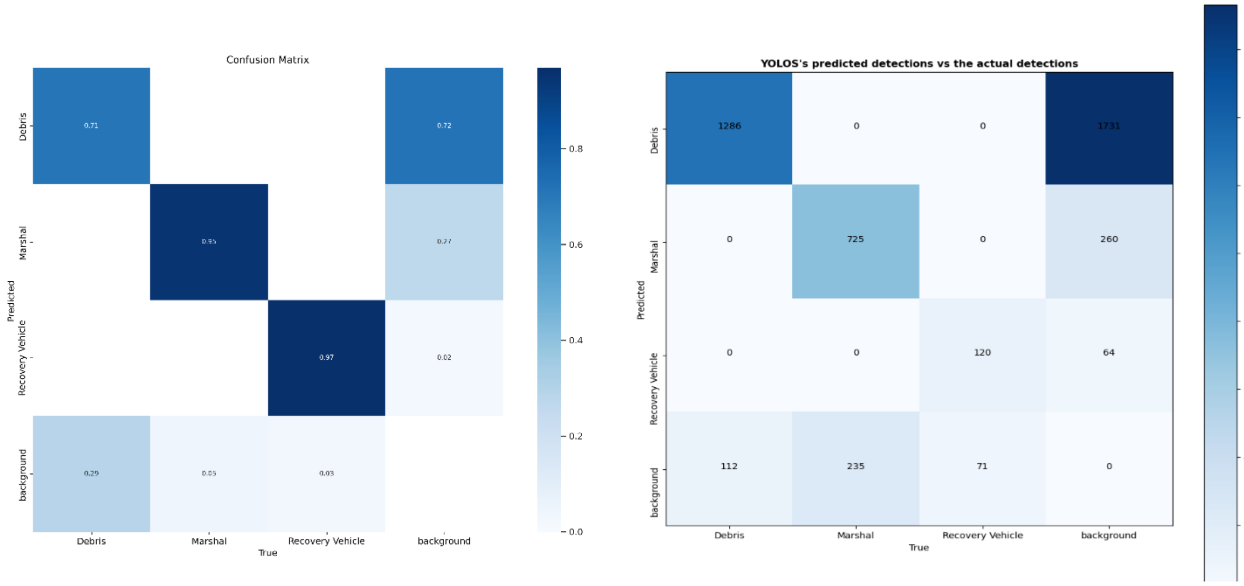


Figure 36: Comparing the confusion matrices of YOLOv8 and YOLOS.

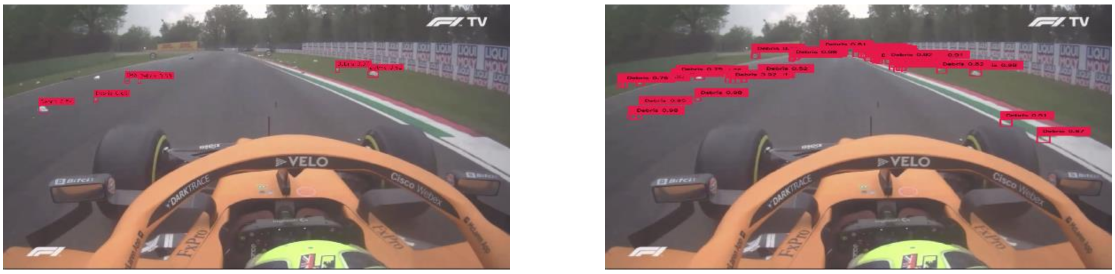
While the formats of the two confusion matrices in figure 36 are slightly different, it is clear to see straight away that YOLOS produces more false positive results than YOLOv8, particularly in the debris object category. YOLOS, however, detects more true positive results for the debris object category compared to YOLOv8, detecting 91% of all actual detections compared to just 71% for YOLOv8. This begs the question which of these two performances in the debris category is deemed to be superior? Is it better to have a model with high recall and low precision or a model with lower recall but higher precision? In my opinion the latter feels more suitable as it does not attempt a scattergun tactic and as such, I believe that YOLOv8 outperforms YOLOS in the debris category due to the substantial number of false positive results produced by YOLOS.

For the marshal category the two models predict the same number of false positive (27%) results but YOLOv8 successfully detects 95% of all actual instances of marshals while YOLOS only detects 76%. Subsequently, it can be suggested that YOLOv8 also outperforms YOLOS in the detection of marshals.

Based on the aforementioned evaluation metrics, it appears that YOLOv8 is superior to YOLOS in all object categories and thus is deemed to be better suited to the task of object detection in a Formula One environment.

6.4.2 Inference on Image

If we compare the two models using the same confidence threshold of 0.5 and an IOU threshold of 0.5, we see that YOLOS arguably over detects the debris while YOLOv8 under detects it.

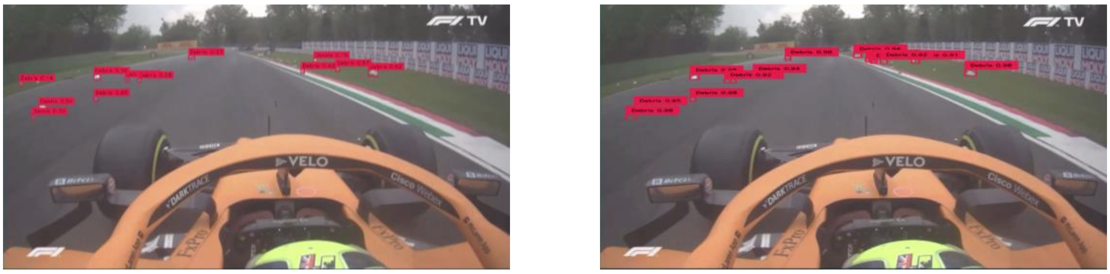


YOLOv8
Confidence threshold: 0.5 | IOU threshold: 0.5

YOLOS
Confidence threshold: 0.5 | IOU threshold: 0.5

Figure 37: Running inference with YOLOv8 and YOLOS on the same image.

To fix this, we can simply adjust these thresholds so to find a better balance that suits each model. Subsequently, changing YOLOv8's confidence threshold to 0.3 and YOLOS's confidence threshold to 0.9 results in a much more balanced outcome.



YOLOv8
Confidence threshold: 0.3 | IOU threshold: 0.5

YOLOS
Confidence threshold: 0.9 | IOU threshold: 0.5

Figure 38: Running inference with YOLOv8 and YOLOS with more suitable confidence thresholds.

This example of inference epitomises the models in terms of how they function and how the need to be setup. Finding the right IOU and confidence threshold is an art and takes several trials to find a good trade-off between achieving true positive detections and false positive detections. No threshold will likely ever provide the perfect balance, but it is nonetheless significant to find the thresholds that are appropriate for both the model and the task at hand.

With more appropriate confidence thresholds we still come to a significant dilemma: Which model performs better? YOLOS detects more instances of debris compared to YOLOv8 while neither model appears to detect any false positive results. This suggests that YOLOS is perhaps the superior model.

6.4.3 Real-time Inference on Video

To test the real-time inference capabilities of both models, I produced the python scripts *YOLOv8 Object Detection.py* and *YOLOS Object Detection.py* as referred to in section 5.4 and generated a demonstration video by compiling a handful of on-board perspectives of incidents that neither model had yet witnessed. The demonstration video runs for 2 minutes and 13 seconds and depicts several challenging scenarios including a vast mass of debris, water droplets on the camera, marshals interacting with debris and on-board perspectives with drivers who have crashed.

The results from feeding this demonstration video into both the YOLOv8 and YOLOS model can be found in *Demonstration.mp4*. *Demonstration.mp4* runs the original video alongside the YOLOv8 and YOLOS output providing a useful insight as to how the models would compare in a real-time environment.

The demonstration video runs at 30 FPS and thus sets the objective for the two object detection models to match or surpass this throughput in order to illustrate their capability and suitability to run inference in real-time.

The YOLOv8m model was the closest to achieving this, running at 29.5 FPS with an average inference speed of 24.9ms. While not quite matching the 30 FPS target threshold, a feat of 29.5 FPS is not far off and so for me still illustrates its capability to run in real-time.

On the other hand, the YOLOS-Tiny model only achieved a throughput of 26.1 FPS with an average inference speed of 33.2ms. While the achieved throughput is only 3.9 FPS off the target threshold, the subsequent delay observed in *Demonstration.mp4* is substantial, with the YOLOS-Tiny model taking 2 minutes and 27 seconds to run the 2 minutes and 13 seconds long demonstration video. Over the course of a whole race, this delay could become several minutes long making the detections made by the YOLOS model useless/irrelevant. Once again, the distance to achieving the target throughout is not substantial and as such the YOLOS model demonstrates its potential to run in real-time.

A workaround for these minute insufficiency in throughput performance could be to utilise superior hardware or make use of hardware optimised code so to reduce the inference speed by a couple milliseconds more. Another potential solution for the YOLOv8 model could also be to downgrade to the slightly faster YOLOv8s variant. This would improve inference speeds but also reduce accuracy. Given the 0.5 FPS shortfall in throughput, this option seems a tad extreme.

In terms of the accuracy of the models throughout the inference on the demonstration video, there are no quantitative metrics and so a qualitative analysis is required. An overall assessment of both models in this real-time demonstration would be that YOLOS correctly detects more objects than YOLOv8. However, even while using a confidence threshold of 0.9 YOLOS tended to make lots of false positive predictions. In contrast YOLOv8 was very cautious and was generally accurate with its detections, but too often did not detect obvious instances of objects even while using a confidence threshold of 0.5.

7. Conclusion

Throughout this project, two object detection models using conflicting approaches were produced to detect the presence of on track debris, marshals, and recovery vehicles in real-time Formula One environments. YOLOS provides surprising competition to YOLOv8 in spite of the intension for YOLOS to not be state-of-the-art but to illustrate the versatility of transformers. Both models produced promising results that at one point or another detect the majority of observed object instances in the demonstration video *Demonstration.mp4.*

While the results are promising, neither model, in my opinion, is yet reliable or accurate enough in their current form to be deployed for real-world usage and if done so, could perhaps add more issues and confusion to the decision-making process.

7.1 Assessment of Initial Aim and Objectives

As defined in section 1.2 of this report, the aim of this research project was to produce a robust real-time debris detection system for Formula One that detects the presence of on track debris, marshals, and recovery vehicles. Based on the results from the YOLOv8m and YOLOS-Tiny object detection models, as referred to in section 6, I would deem this aim to be achieved given that both models can successfully detect all the object categories to some degree of success. The success of the work done in this project is further illustrated by both YOLOv8 models and the optimised YOLOS model surpassing the expectations laid out in section 4.3.

While I do not believe that either model is sufficiently reliable or accurate to deploy and utilise in real-world scenarios, I do believe that the results show a vast amount of potential especially given the surprisingly decent performance of the detection of debris across the YOLOv8 and YOLOS detection models. With future research in this field, more data and superior resources for training and inference, I believe that this concept could very realistically be implemented in the near future.

7.2 What Could Have Been Done Better?

While the results of the YOLOv8 and YOLOS object detection models are promising, I do believe given the limited resources and time available to me for this project that I could have produced better results.

I believe that the data gathered for the created dataset was insufficient in volume for the task and, as referred to in section 5.1.3, struggles with the annotation of the data could have further reduced the quality of the produced dataset. I am now more familiar with the task of annotating images having annotated 5,777 images over the course of this project, however during the early stages of this task I was drawing loose fitting bounding boxes and subsequently reducing the quality of the data. In future, this task needs to be performed to a higher level of accuracy and consistency and could potentially benefit from the addition of a quality control phase.

Another aspect of this project I could have improved is the comparisons between the two contrasting object detection models. I believe that the two models are implemented in somewhat different approaches and subsequently the results are in somewhat different formats. If I could do this project again, I would try to implement the two models in as similar way as possible so to produce the same evaluation metrics and validation visualisations in the exact same format as one another.

7.3 Skills Developed

Prior to commencing this project, I had no experience working with object detection algorithms and knew nothing of the problem, its existing solutions, or its limitations. Over the course of this project, I have developed a deep understanding regarding all aspects of object detection from image annotation tools to architecture decisions. I am now able to understand how models have evolved over time and why these evolutions occurred.

In addition to the technical skills and knowledge that I have developed, I would say that my project management skills have also seen vast improvement as a result of this challenging self-led research project. Throughout the course of this project, I was required to define the aim of the project, determine what steps were required to achieve this aim and set a number of sprint objectives throughout so to ensure the aim was completed before the given deadline. While my project management ability has not been perfect, I am able to walk away from this project knowing where I can improve in the future.

7.4 Future Developments

In order for this proposed solution to become a realistic option, a number of features and improvements are required.

The first steps are to improve on the results of the object detection models produced throughout this project and produce a more reliable, accurate and robust object detection model that can be deemed sufficiently safe to implement into a real-world health and safety role. To do this, I would gather much more data from a much wider range of incidents, include other foreign objects such as f1 cars, safety cars, animals, helicopters and more so that the model better understands the context of the image and is less likely to misleadingly identify one of the aforementioned objects as debris and finally experiment with using subcategories for the debris object category given its broad description.

Once a sufficiently reliable object detection model is produced, the next step is to successfully integrate this technology into the current practices of Race Control. This means deploying the produced model and designing a graphical user interface to plainly communicate detections from the object detection model to the officials. To do this, I believe principles and techniques from the fields of Human Computer Interaction (HCI) and Data Visualisation could be used to further improve the human ability to recognise the model's detections. The resulting solution could even integrate the positional/location data of the cars and any other cameras used so to inform Race Control as to where exactly on the track the object can be found, allowing for much faster response times to incidents when they occur.

7.5 Future Research

When trying to implement the object detection system as initially intended by feeding all 20 driver's on-board footage into the models for inference, I came across the problem of object co-detection as referred to by Sid Yingze Bao, Yu Xiang and Silvio Savarese in their paper called Object Co-detection (Sid Yingze Bao, 2012). Given multiple images, each of which may contain instances of objects from different viewpoints, the goal of co-detection is to detect objects in all images and recognise whether or not the objects are different.

In practice this would essentially assign an ID number to each detected object and would mean that in this case of this project that an instance of debris observed across multiple laps would be regarded as a singular instance of debris and not multiple. Further research into this field would provide exciting avenues for further development.

7.6 Final Thoughts

This project has illustrated the potential for deep learning technology to assist and support health and safety procedures in the world of motorsport. The use of deep learning technology for tedious observation-based tasks such as this could alleviate huge amounts of pressure off officials and allow them to focus on the finer rules of the sport while allowing fans to experience the sport they know and love with minimal delays.

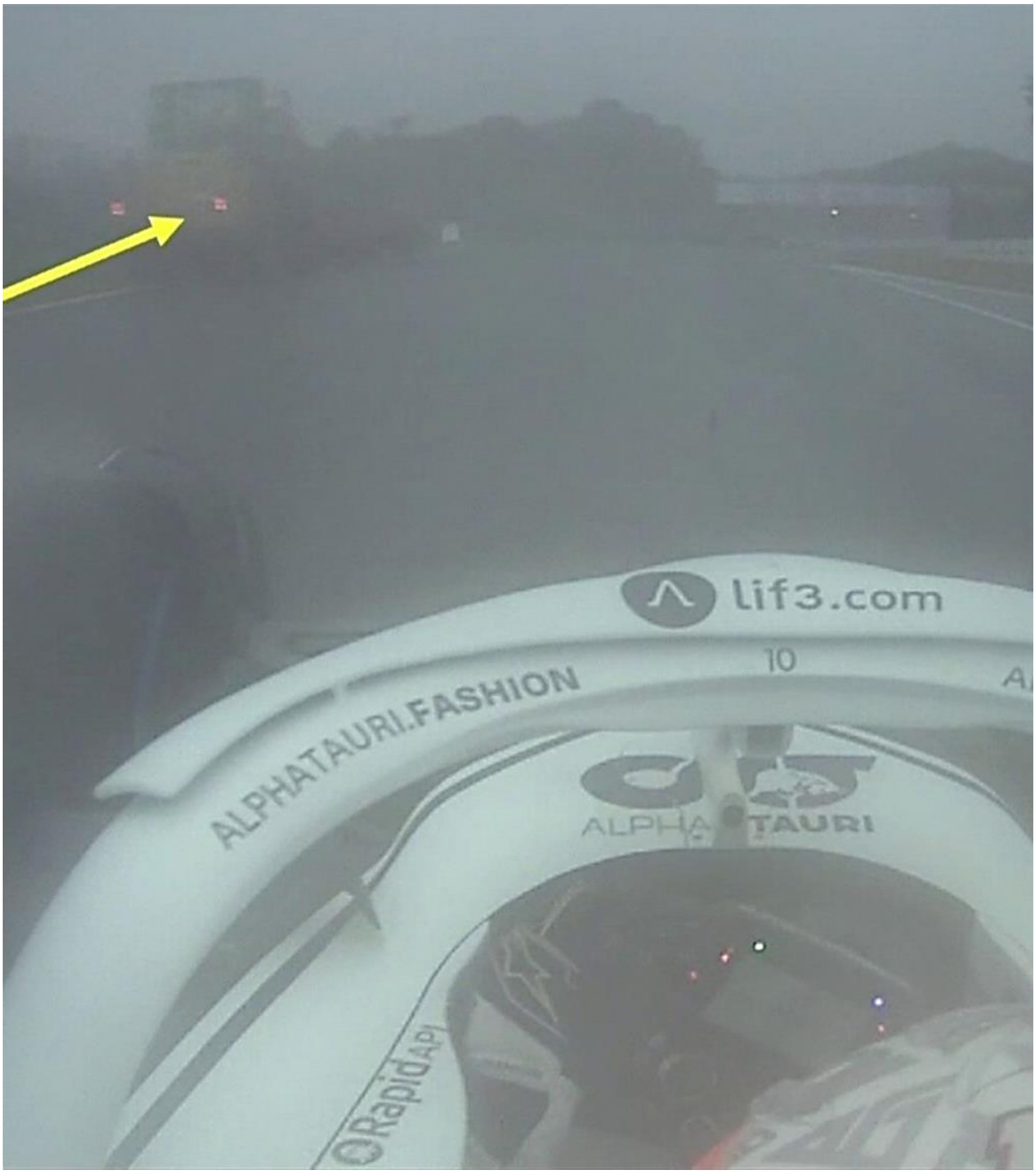
I believe that this is the future of sport and for this future to come about, the decision-making procedures in sport must improve with the technology. It is no good making use of the technology if the resulting procedures perform to a lower standard than that currently implemented. As a result, technology is just one component in the potential development for automated detection systems in sport. The other component is how the human officials interact with the technology and the extent to which this interaction is seamless and frictionless.

Appendices

Appendix A:

Felipe Massa's injuries from being hit by a loose spring at the 2009 Hungarian Grand Prix *(George, 2019)*

Appendix B:



Pierre Gasly's near miss with a recovery vehicle at the 2022 Japanese Grand Prix *(Sky Sports, 2022)*

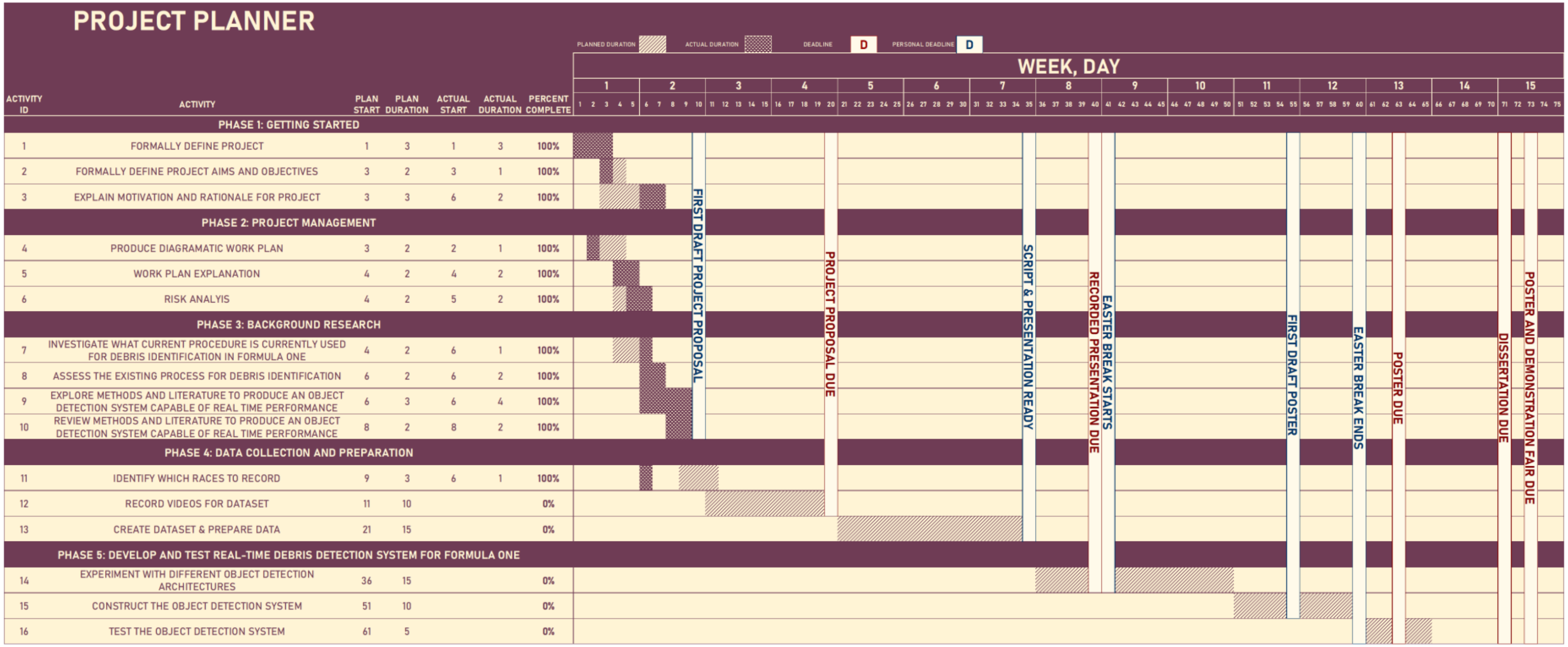
Appendix C:

Formula One fan, Will Sweet holding the piece of debris that cut his arm at the 2023 Australian Grand Prix *(ITVX, 2023)*

Appendix D:

Race Control at the 2019 Australian Grand Prix during Practice Session 2 *(SBG Sports Software, 2020)*

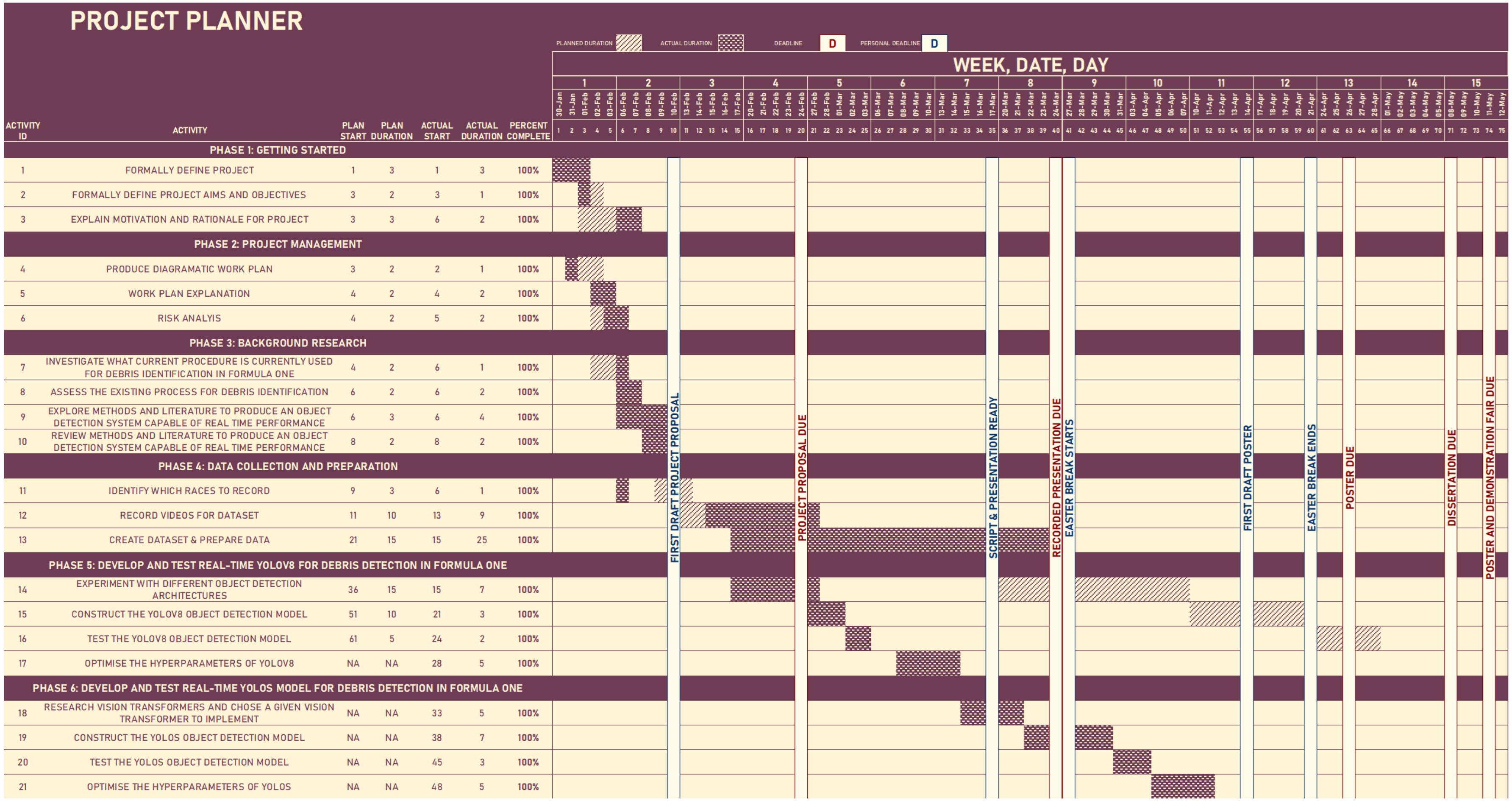
Appendix E:



ACTIVITY ID	ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
	PHASE 1: GETTING STARTED					
1	FORMALLY DEFINE PROJECT	1	3	1	3	100%
2	FORMALLY DEFINE PROJECT AIMS AND OBJECTIVES	3	2	3	1	100%
3	EXPLAIN MOTIVATION AND RATIONALE FOR PROJECT	3	3	6	2	100%
	PHASE 2: PROJECT MANAGEMENT					
4	PRODUCE DIAGRAMATIC WORK PLAN	3	2	2	1	100%
5	WORK PLAN EXPLANATION	4	2	4	2	100%
6	RISK ANALYIS	4	2	5	2	100%
	PHASE 3: BACKGROUND RESEARCH					
7	INVESTIGATE WHAT CURRENT PROCEDURE IS CURRENTLY USED FOR DEBRIS IDENTIFICATION IN FORMULA ONE	4	2	6	1	100%
8	ASSESS THE EXISTING PROCESS FOR DEBRIS IDENTIFICATION	6	2	6	2	100%
9	EXPLORE METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL TIME PERFORMANCE	6	3	6	4	100%
10	REVIEW METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL TIME PERFORMANCE	8	2	8	2	100%
	PHASE 4: DATA COLLECTION AND PREPARATION					
11	IDENTIFY WHICH RACES TO RECORD	9	3	6	1	100%
12	RECORD VIDEOS FOR DATASET	11	10			0%
13	CREATE DATASET & PREPARE DATA	21	15			0%
	PHASE 5: DEVELOP AND TEST REAL-TIME DEBRIS DETECTION SYSTEM FOR FORMULA ONE					
14	EXPERIMENT WITH DIFFERENT OBJECT DETECTION ARCHITECTURES	36	15			0%
15	CONSTRUCT THE OBJECT DETECTION SYSTEM	51	10			0%
16	TEST THE OBJECT DETECTION SYSTEM	61	5			0%

Initial Project Gantt Chart

Appendix F:



ACTIVITY ID	ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
	PHASE 1: GETTING STARTED					
1	FORMALLY DEFINE PROJECT	1	3	1	3	100%
2	FORMALLY DEFINE PROJECT AIMS AND OBJECTIVES	3	2	3	1	100%
3	EXPLAIN MOTIVATION AND RATIONALE FOR PROJECT	3	3	6	2	100%
	PHASE 2: PROJECT MANAGEMENT					
4	PRODUCE DIAGRAMATIC WORK PLAN	3	2	2	1	100%
5	WORK PLAN EXPLANATION	4	2	4	2	100%
6	RISK ANALYIS	4	2	5	2	100%
	PHASE 3: BACKGROUND RESEARCH					
7	INVESTIGATE WHAT CURRENT PROCEDURE IS CURRENTLY USED FOR DEBRIS IDENTIFICATION IN FORMULA ONE	4	2	6	1	100%
8	ASSESS THE EXISTING PROCESS FOR DEBRIS IDENTIFICATION	6	2	6	2	100%
9	EXPLORE METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL TIME PERFORMANCE	6	3	6	4	100%
10	REVIEW METHODS AND LITERATURE TO PRODUCE AN OBJECT DETECTION SYSTEM CAPABLE OF REAL TIME PERFORMANCE	8	2	8	2	100%
	PHASE 4: DATA COLLECTION AND PREPARATION					
11	IDENTIFY WHICH RACES TO RECORD	9	3	6	1	100%
12	RECORD VIDEOS FOR DATASET	11	10	13	9	100%
13	CREATE DATASET & PREPARE DATA	21	15	15	25	100%
	PHASE 5: DEVELOP AND TEST REAL-TIME YOLOV8 FOR DEBRIS DETECTION IN FORMULA ONE					
14	EXPERIMENT WITH DIFFERENT OBJECT DETECTION ARCHITECTURES	36	15	15	7	100%
15	CONSTRUCT THE YOLOV8 OBJECT DETECTION MODEL	51	10	21	3	100%
16	TEST THE YOLOV8 OBJECT DETECTION MODEL	61	5	24	2	100%
17	OPTIMISE THE HYPERPARAMETERS OF YOLOV8	NA	NA	28	5	100%
	PHASE 6: DEVELOP AND TEST REAL-TIME YOLOS MODEL FOR DEBRIS DETECTION IN FORMULA ONE					
18	RESEARCH VISION TRANSFORMERS AND CHOSE A GIVEN VISION TRANSFORMER TO IMPLEMENT	NA	NA	33	5	100%
19	CONSTRUCT THE YOLOS OBJECT DETECTION MODEL	NA	NA	38	7	100%
20	TEST THE YOLOS OBJECT DETECTION MODEL	NA	NA	45	3	100%
21	OPTIMISE THE HYPERPARAMETERS OF YOLOS	NA	NA	48	5	100%

Final Project Gantt Chart

References

George, D., 2019. *Felipe Massa Proves How Far he has Come Since his 2009 Crash.* [Online]
Available at: Felipe Massa Proves How Far he has Come Since his 2009 Crash
[Accessed 20 April 2023].

ITVX, 2023. *F1 fan struck by debris in Australian Grand Prix crash escaped 'horrendous' injury.* [Online]
Available at: https://www.itv.com/news/anglia/2023-04-03/british-f1-fan-hit-by-debris-in-australian-grand-prix-crash
[Accessed 20 April 2023].

Liu, W. a. A. D. a. E. D. a. S. C. a. R. S. a. F. C.-Y. a. B. A. C., 2016. *SSD: Single Shot MultiBox Detector.* Amsterdam, European Conference on Computer Vision.

Liu, Y. F. a. B. L. a. X. W. a. J. F. a. J. Q. a. R. W. a. J. N. a. W., 2021. You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection. *Advances in Neural Information Processing Systems,* 34(9781713845393), pp. 26183--26197.

NVIDIA, 2023. *CUDA Zone.* [Online]
Available at: https://developer.nvidia.com/cuda-zone#:~:text=CUDA%C2%AE%20is%20a%20parallel,harnessing%20the%20power%20of%20GPUs.
[Accessed 30 April 2023].

Press, G., 2016. *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says.* [Online]
Available at: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=315ac4456f63
[Accessed 16 february 2023].

Pretorius, L., n.d. *How Many F1 Drivers Have Died?.* [Online]
Available at: https://onestopracing.com/how-many-f1-drivers-have-died/
[Accessed 20 April 2023].

Ren, S. a. H. K. a. G. R. a. S. J., 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in Neural Information Processing Systems,* Volume 28.

SBG Sports Software, 2020. *SBG Sports Software is Appointed as FIA Exclusive Official Supplier of Race Control Systems.* [Online]
Available at: https://sbgsportssoftware.com/news/sbg-sports-software-is-fia-exclusive-official-supplier-of-race-control-systems
[Accessed 19 February 2023].

Scrum, 2020. *The Scrum Framework Poster.* [Online]
Available at: https://www.scrum.org/resources/scrum-framework-poster
[Accessed 17 April 2023].

Sid Yingze Bao, Y. X. a. S. S., 2012. Object Co-detection. In: S. L. P. P. Y. S. C. S. Andrew Fitzgibbon, ed. *Computer Vision -- ECCV 2012.* 978-3-642-33718-5 ed. s.l.:Springer Berlin Heidelberg, pp. 86-101.

Sky Sports, 2022. *Japanese GP: Pierre Gasly feared for life in truck near-miss on track, FIA penalises him for speeding.* [Online]
Available at: https://www.skysports.com/f1/news/12433/12716198/japanese-gp-pierre-gasly-feared-for-life-in-truck-near-miss-on-track-fia-penalise-him-for-speeding
[Accessed 14 April 2023].

Song, D., Yuan, F. & Ding, C., 2022. *Track Foreign Object Debris Detection based on Improved YOLOv4 Model.* Beijing, China, IEEE.

The Herald, 2023. *British F1 fan lucky to avoid 'horrendous' injury after flying debris.* [Online]
Available at: https://www.heraldscotland.com/sport/23432546.british-f1-fan-lucky-avoid-horrendous-injury-flying-debris/
[Accessed 29 April 2023].

Ultralytics, 2023. *ultralytics GitHub.* [Online]
Available at: https://github.com/ultralytics/ultralytics
[Accessed 27 April 2023].

Ultralytics, 2023. *Ultralytics x Paperspace: Advancing Object Detection Capabilities Through Partnership.* [Online]
Available at: https://ultralytics.com/article/Ultralytics-Paperspace-Advancing-Object-Detection-Capabilities-Through-Partnership
[Accessed 27 April 2023].

Yuxin Fang, B. L. X. W. J. F. J. Q. R. W. J. N. W. L., 2022. *YOLOS GitHub.* [Online]
Available at: https://github.com/hustvl/YOLOS
[Accessed 30 April 2023].