ggh

OBJECT DETECTION FOR AN AUTONOMOUS RACE CAR

SYSTEMS DEVELOPMENT GROUP PROJECT COURSEWORK

BY

**Group 8**

|  |  |
| --- | --- |
| Members | Contribution |
| Yousef Ali (22002656) | 19.1 % |
| Asser Aldardiri (210665616) | 25.5 % |
| Meshina Ushie (22039488) | 18 % |
| Taha Seghayer (21065689) | 18.1 % |
| Abdulrahman Ali (21065685) | 19.3 % |

Contents

[Introduction: 2](#_Toc133241926)

[Aims and objectives: 2](#_Toc133241927)

[Literature Review: 3](#_Toc133241928)

[Requirements: 6](#_Toc133241929)

[Project Planning and Team Roles: 8](#_Toc133241930)

[Design and Test Plan: 9](#_Toc133241931)

[Implementation and Testing: 10](#_Toc133241932)

[Evaluation and lessons learned: 15](#_Toc133241933)

# Introduction

Our team has developed an autonomous object detection program using Convolutional Neural Network (CNN) architecture, which we have trained on a large dataset to accurately detect objects in various conditions. The computer vision field has advanced considerably within the past decade, with it being used in the automotive industry, surveillance image, and video analysis. The main objective of the program is to identify multiple cones of various sizes and colors that need to be classified as blue, yellow or orange. The CNN was developed using TensorFlow – a framework used for developing CNN where it contains a large range of tools for creating, training, and testing CNNs- with the aid of Numpy –a python library that provides a powerful array data structure, that is commonly used to preprocess image data including; resizing, normalization, and conversions to array- , while FSCOCO data set is used to train the CNN to detect the cones. In this report, we will detail how our team designed and implemented the CNN architecture, along with testing and evaluation of the program’s performance.

# Aims and objectives:

The aim of this project is to develop a machine learning-based object detection model that can accurately detect cones in RGB images and output bounding boxes and class labels for all cones. This model will be specifically designed for the autonomous racing system of the Formula Student Artificial Intelligence (FS-AI) event, where the racing vehicle will use the ZED camera to detect cones laid out across the track. The detection of cones is crucial for the AI system to make decisions on how to drive during the dynamic events of the competition. The developed object detection model will be evaluated for accuracy and speed using a dataset of RGB images containing cones and will be tested for its ability to handle real-time processing of the stereo camera input. The model should be capable of detecting cones in various lighting and environmental conditions and should be relevant not only to autonomous racing but also to other object detection applications. By completing this project, the team aims to gain expertise in machine vision and machine learning techniques for object detection and evaluation.  
  
The objectives for this project are as follows:

**Specific**: Develop a machine learning-based object detection model using publicly available datasets that contain cones of the Formula Student design, such as the FSOCO dataset.

**Measurable**: Evaluate the accuracy and speed of the object detection model by testing it on a dataset of RGB images containing cones and measuring its precision, recall, and F1 score.

**Achievable**: Ensure that the model is capable of accurately detecting cones in various lighting and environmental conditions and can handle real-time processing of the stereo camera input.

**Relevant**: Develop a model that is specifically designed for detecting cones in an autonomous racing system but can also be applied to other object detection applications.

**Time-bound**: Complete the development and evaluation of the object detection model within a specific timeframe, taking into account the project timeline and deadline for the submission of the final model as well as the FS-AI event submission deadline.

(337)

# Literature Review:

Object detection is a fundamental task in computer vision and is crucial for many real-world applications, including robotics, autonomous vehicles, and surveillance systems. In recent years, deep learning-based object detection models have achieved significant progress in terms of accuracy and speed, making them a popular choice for many applications (Ren et al., 2015; Redmon et al., 2016; Liu et al., 2016). However, the development of object detection models that can accurately detect cones in RGB images under various lighting and environmental conditions remains a challenging task. The Formula Student Artificial Intelligence (FS-AI) event is a competition that challenges students to design and build autonomous racing vehicles capable of completing dynamic events, such as obstacle avoidance, slalom, and acceleration (Institution of Mechanical Engineers, 2022). One of the key tasks for the perception system of these vehicles is to accurately detect cones laid out across the track, which mark the boundaries and the start line. The detection of cones is crucial for the AI system to make decisions on how to drive during the competition. In recent years, several research studies have focused on the detection of cones for autonomous racing systems, using various approaches such as color-based segmentation, template matching, and deep learning-based object detection. This project aims to develop a deep learning-based object detection model for the detection of cones in RGB images, specifically for the autonomous racing system of the FS-AI event. To achieve this aim, publicly available datasets containing cones of the Formula Student design, such as the FSOCO dataset, will be used for model training and evaluation (Niclas Vödisch et al., 2020). While there has been significant research in the field of object detection using machine learning algorithms (Ren et al., 2015; Redmon & Farhadi, 2018), there are still gaps in the existing literature when it comes to applying these techniques to autonomous racing vehicles. Most of the existing work has focused on object detection in more traditional settings such as surveillance, face recognition, and medical imaging. In the context of autonomous racing vehicles, there is a need to accurately detect and classify cones in RGB images in real-time. While there are some datasets available, such as the FSOCO dataset (FSOCO Dataset, n.d.), which contains images of cones in the context of Formula Student competitions, there is a lack of research on the specific challenges and limitations of object detection in this context. For example, the stereo camera used in the ADS-DV racing vehicle has a limited field of view, and the images obtained can be affected by changing lighting conditions, shadows, and reflections. Furthermore, there is a need to evaluate the performance of object detection models not only in terms of accuracy but also in terms of speed and resource requirements, given the computational constraints of the ADS-DV platform. Therefore, this project aims to address these gaps by training and evaluating an object detection model specifically for cones in the context of autonomous racing vehicles, using the available datasets and taking into account the specific challenges and limitations of this context. The proposed project aims to develop an object detection system for an autonomous race vehicle to detect cones in RGB images. The key concepts and variables of the project include the ZED1 stereo camera, IMU and wheel speed encoders on the race car, and the perception system used for detecting and estimating the depth of cones. The system will be trained using publicly available datasets, such as the FSOCO dataset, and will output a bounding box and class label for all cones in the image. Pre-trained models of SSD and YOLOv5 have been used in similar systems for object detection, and they have been shown to achieve high accuracy and speed. SSD is a real-time object detection algorithm that uses a single neural network for predicting class scores and box coordinates. It has been used in various applications, such as pedestrian detection and vehicle detection, and has demonstrated excellent results in terms of accuracy and speed (Wei Liu et al., 2016). YOLOv5 is another real-time object detection system that has been recently introduced and has shown to achieve state-of-the-art results in terms of accuracy and speed (Zheng et al., 2021). The use of pre-trained models can significantly reduce the time and resources required for training an object detection system, making it a viable option for the proposed project. A pre-trained model can be fine-tuned using the target dataset, such as the FSOCO dataset, to adapt to the specific application and improve the accuracy of the system. Moreover, transfer learning, which involves reusing a pre-trained model for a related task, can be used to further improve the performance of the system by leveraging the knowledge learned from the pre-trained model (Pan and Yang, 2010). Several studies have focused on developing cone detection methods for autonomous racing systems. (Deng et al. (2020)) proposed a cone detection method based on color and shape features using a convolutional neural network (CNN) to classify cones. The authors used a dataset of cone images captured under different lighting and environmental conditions to train and test the CNN model. The results showed that their method achieved high accuracy in detecting cones in RGB images. Liu et al. (2021) presented a cone detection system based on a hybrid approach using both template matching and deep learning-based object detection. The authors used template matching to detect cones in the first frame and used the detected cones to train a YOLOv3-based object detection model. The proposed system achieved high accuracy in detecting cones in challenging scenarios, such as low-light conditions and occlusions. Yu et al. (2021) proposed a cone detection system based on a deep learning-based object detection method. The authors used a Faster R-CNN model to detect cones in RGB images captured by a stereo camera. The proposed system achieved high accuracy in detecting cones in various lighting and environmental conditions. Schneider et al. (2020) introduced the FSOCO dataset, a publicly available dataset of cone images captured in the context of Formula Student competitions. The dataset contains images of cones under different lighting and environmental conditions, making it suitable for training and testing cone detection models for autonomous racing systems. Overall, these studies demonstrate the feasibility and effectiveness of using deep learning-based methods for cone detection in the context of autonomous racing systems. The availability of publicly available datasets, such as the FSOCO dataset, has also facilitated the development and evaluation of cone detection methods for these systems. However, there is still a need for further research on cone detection in challenging scenarios, such as low-light conditions, occlusions, and reflections, which are commonly encountered in real-world racing environments. In the context of object detection systems, evaluation metrics are essential for assessing the performance of the system. The evaluation of object detection systems is typically done using metrics such as precision, recall, and F1-score (Everingham et al., 2010). These metrics are used to measure the accuracy of the system in terms of correctly detecting objects and avoiding false detections. The precision metric measures the percentage of correct detections among all the detections made by the system, while recall measures the percentage of objects correctly detected among all the objects present in the image. The F1-score is a weighted harmonic mean of precision and recall and is commonly used as a summary metric for evaluating the overall performance of the system (Everingham et al., 2010). In addition to precision, recall, and F1-score, other evaluation metrics are also used to assess the performance of object detection systems. One such metric is the mean average precision (mAP), which is commonly used in object detection challenges such as the COCO (Common Objects in Context) benchmark (Lin et al., 2014). The mAP metric measures the average precision of the system at various levels of recall and is often used as a standard metric for comparing different object detection models. Another commonly used metric is the intersection over union (IoU), which measures the overlap between the predicted bounding box and the ground truth bounding box. A high IoU indicates a high degree of overlap between the predicted and ground truth bounding boxes and is often used to filter out false detections (Hosang et al., 2017). It is important to note that the evaluation metrics used for object detection systems may vary depending on the specific application and dataset. For example, in the context of autonomous racing systems, the speed of the system may be an important factor to consider in addition to accuracy. In such cases, metrics such as inference time and memory usage may also be used to evaluate the performance of the system. In conclusion, evaluation metrics are an essential component of object detection systems, and a variety of metrics are used to assess the performance of these systems. The selection of appropriate evaluation metrics depends on the specific application and dataset, and it is important to consider both accuracy and speed when evaluating object detection systems for autonomous racing systems. Despite the success of deep learning-based object detection models, there are still some limitations that need to be addressed. One of the major limitations is the need for large amounts of annotated data for training. Collecting and annotating data can be time-consuming and expensive, particularly for specialized domains such as autonomous racing vehicles. Additionally, deep learning models are typically computationally expensive and require powerful hardware, which can be a barrier to adoption in resource-constrained environments. Another limitation is the lack of interpretability and transparency of deep learning models. It can be difficult to understand why a model makes a particular prediction, which can limit their use in safety-critical applications such as autonomous driving. There is a need for techniques that can provide insight into the decision-making process of deep learning models and enable their use in safety-critical applications. Furthermore, deep learning models can be susceptible to adversarial attacks, where small perturbations to the input data can cause the model to make incorrect predictions. Adversarial attacks are a particular concern in security-critical applications such as surveillance systems, where an attacker may seek to evade detection. Finally, deep learning models may not perform well in novel or unseen environments. For example, if an autonomous racing vehicle encounters a new type of cone that it has not been trained on, the object detection model may fail to detect it. This is known as the problem of domain adaptation and transfer learning, and it remains an active area of research in machine learning. To address these limitations, researchers are exploring techniques such as transfer learning, data augmentation, and regularization to improve the performance and robustness of deep learning-based object detection models. Additionally, there is a growing interest in developing explainable AI techniques that can provide insights into the decision-making process of deep learning models. Finally, research is ongoing in developing more efficient and lightweight deep learning models that can operate in resource-constrained environments. Data augmentation is a crucial step in the training of deep learning models for object detection. It involves creating new training samples by applying various transformations to the original dataset. These transformations can help to increase the diversity of the training data and prevent overfitting of the model. In this section, we review some of the commonly used data augmentation techniques for object detection. One of the most commonly used data augmentation techniques for object detection is image flipping. This involves horizontally flipping the image and its corresponding bounding boxes. This transformation can be performed with a probability of 0.5, effectively doubling the size of the training dataset (Simonyan and Zisserman, 2014). Another useful data augmentation technique is random cropping. This involves randomly selecting a region of the image and resizing it to the desired input size. This can help to increase the diversity of the training data and improve the robustness of the model to object occlusion and partial object detection (Redmon et al., 2016). Rotation and scaling are other common data augmentation techniques for object detection. These transformations involve rotating or scaling the image and its corresponding bounding boxes. Rotation can help to improve the model's ability to detect objects at different orientations, while scaling can help to improve the model's ability to detect objects at different sizes (Ren et al., 2015). Other data augmentation techniques for object detection include adding noise, changing brightness and contrast, and applying color jittering. These techniques can help to increase the diversity of the training data and improve the robustness of the model to changes in lighting conditions and image quality (Liu et al., 2016). In addition to these techniques, recent studies have also explored the use of more advanced data augmentation techniques, such as CutMix and MixUp. CutMix involves cutting and pasting a patch from one image onto another image, while MixUp involves linearly combining two images and their corresponding labels. These techniques have been shown to improve the accuracy of object detection models, particularly in the presence of class imbalance and noisy data (Yun et al., 2019). In conclusion, data augmentation is an essential step in the training of deep learning models for object detection. A range of techniques can be used to increase the diversity of the training data and improve the robustness of the model to changes in lighting conditions, object occlusion, and partial object detection. The selection of appropriate data augmentation techniques depends on the specific characteristics of the dataset and the target application.

(2258)

# Requirements:

Building an object detection system for an autonomous race vehicle is a complex and challenging task that requires careful planning, development, and testing. To ensure that the system functions reliably and accurately in a high-speed racing environment, there are several key requirements that must be considered. In this section, we will outline some of the primary requirements for building an object detection system for an autonomous race vehicle, as well as explain the rationale behind each requirement. The requirements for building an object detection system for an autonomous race vehicle can be classified into functional and non-functional requirements according to the MoSCoW prioritization framework.

1. Functional Requirements: a)
   1. Accurate detection of objects: The primary requirement for an object detection system is that it must be able to accurately detect and classify objects in the vehicle's environment. This includes both stationary and moving objects, such as other vehicles, pedestrians, traffic signs, and obstacles. To achieve this level of accuracy, the system should use a combination of sensors, such as cameras, lidar, and radar, and employ advanced algorithms for object recognition and classification.
   2. Real-time processing: In a racing environment, the object detection system must operate in real-time to keep up with the high speeds and rapidly changing conditions. This means that the system must be able to process and analyze data from the sensors in real-time and provide timely feedback to the vehicle's control system. This requires a high-performance computing platform and efficient algorithms for data processing and analysis.
   3. Robustness and reliability: An object detection system for an autonomous race vehicle must be able to operate reliably and robustly in a wide range of conditions. This includes dealing with adverse weather conditions, changes in lighting conditions, and other environmental factors that may affect sensor performance. To achieve this level of robustness, the system must be thoroughly tested and validated under a variety of conditions and incorporate redundancy and fail-safe mechanisms to ensure that it can continue to operate even if one or more sensors fail.
   4. Integration with the vehicle's control system: Finally, an object detection system for an autonomous race vehicle must be seamlessly integrated with the vehicle's control system to ensure that the vehicle can respond quickly and accurately to changes in its environment. This requires close collaboration between the object detection team and the vehicle's control system team, as well as careful planning and testing to ensure that the two systems can communicate effectively and operate in a coordinated manner.
2. Non-functional Requirements:
   1. Low latency: To ensure that the vehicle can respond quickly to changes in its environment, the object detection system must have low latency. This means that the time delay between the detection of an object and the vehicle's response must be kept to a minimum. Achieving low latency requires careful optimization of the system architecture and algorithms, as well as a high-speed communication system between the object detection system and the vehicle's control system.
   2. Low power consumption: In a racing environment, the vehicle's power supply is limited, so the object detection system must be designed to operate with low power consumption. This requires careful selection of sensors and components that are energy-efficient, as well as optimization of algorithms for data processing and analysis to minimize power consumption.
   3. Scalability: As the complexity of the racing environment increases, the object detection system must be able to scale to handle a larger number of objects and more complex scenarios. This requires a flexible and modular system architecture, as well as algorithms that can adapt to changing conditions and new types of objects.

In the MoSCoW framework, the term "Must-have" refers to requirements that are essential for the successful delivery of the project. These requirements are considered as functional requirements that cannot be ignored. In the case of the object detection system for an autonomous race vehicle, the "Must-have" requirements are accurate detection of objects, real-time processing, robustness and reliability, and integration with the vehicle's control system.

The term "Should-have" refers to requirements that are important but not critical for the success of the project. In the case of the object detection system for an autonomous race vehicle, the "Should-have" requirement is low latency. While low latency is important for a racing environment, it is not critical to the success of the project.

The term "Could-have" refers to requirements that are desirable but not necessary for the success of the project. In the case of the object detection system for an autonomous race vehicle, the "Could-have" requirements are low power consumption and scalability. While these requirements are desirable, they are not critical for the success of the project.

Overall, by using the MoSCoW framework to classify the requirements for building an object detection system for an autonomous race vehicle into functional and non-functional requirements, it becomes easier to prioritize them according to their importance and ensure that the project team focuses on the most critical requirements first.

In conclusion, building an object detection system for an autonomous race vehicle requires a careful balance of accuracy, real-time processing, low latency, robustness and reliability, low power consumption, scalability, and integration with the vehicle's control system. By carefully considering each of these requirements and designing a system that meets them, it is possible to develop an object detection system that can safely and reliably navigate a high-speed racing environment.

(892)

# Project Planning and Team Roles:

In the Gantt chart, the project is divided into several tasks, and each task has a start date and a duration. The tasks are organized in a hierarchical structure, with the top-level tasks being the major project phases: Planning, Research & Development, Implementation, and Testing & Evaluation.

Some of the key tasks in each phase include:

* Planning: Defining project scope, requirements, and goals, and creating a project plan.
* Research & Development: Conducting literature review, researching existing object detection algorithms, and prototyping a system design.
* Implementation: Developing the object detection system, integrating it with the vehicle's software, and testing the system.
* Testing & Evaluation: Conducting system testing and evaluation, analyzing results, and identifying areas for improvement.

The Gantt chart also shows dependencies between tasks, which indicate that some tasks cannot start until others are completed. For example, the "Prototype Object Detection System" task in the Research & Development phase cannot start until the "Conduct Literature Review" task is completed.

Overall, the Gantt chart provides a visual representation of the project timeline, helping to ensure that tasks are completed on time and within budget.

(324)

# Design and Test Plan:

The goal of this project is to train a deep learning model to detect different types of traffic cones within an image. The model uses FSOCO datasets to train, validate, and test. This model involves the use of a pre-trained TensorFlow model called Keras.

Design:

The design of this project involves several steps. To begin with, the paths of the data directories id defined, Next the hyperparameters such as img size, batch size, number of epochs, and the learning rate – Note: these hyperparameters are not fixed and could be tuned during the training process to improve the model’s performance.

To prepare the model for training, the images and the corresponding annotations are loaded using the cv2 and numpy libraries. The images then are normalized and converted into RGB format. The annotations are read from a CSV file and converted to a numpy array. To enable the model to preprocess the annotations, a dictionary is created that maps class names to integer values. This is necessary because the model can only process numerical data, so the class names must be converted to integers.

The next step would be splitting the data into training, validation, and testing sets while the number of samples for each set is determined by the OS library. It is important to ensure that the data is split in a stratified manner to ensure each set contains a representative sample of each class.

After the data is normalized and prepared, the model is defined. The open-source library Keras is used to build the neural network, since it provides various convolutional layers that when stacked together create a deep learning model. The following layers were used; MaxPooling –typically used after a convolutional layer- used to reduce the spatial dimensionality of the input to make the model more computationally efficient, Conv2D –a layer that applies convolutional operations to input- used to detect spatial patterns in the input data, Flatten layer is used to convert convolutional outputs to 1D vector, Dense which is a layer that connects every neuron in the current layer to every neuron in the next layer, and Reshape a layer that changes the shape of the input tensor without changing its total number of elements. The model is evaluated on the validation data after each epoch to monitor its progress. Finally, the model is tested using the testing data to determine its accuracy.

To improve the model's performance of the model, data augmentation techniques such as random rotations, translations, flips, and scaling can be used to generate new images from the existing ones. This can help prevent overfitting and improve the generalization performance of the model.

Test:

The success of the project will be evaluated based on the accuracy of the model in the testing data. The model will be considered successful if it can correctly classify the different types of traffic cones in the testing datasets. Accuracy is measured using categorical accuracy metric – it is the fraction of images that are classified correctly.

To ensure the accuracy of the model, several steps must be taken during the testing. Starting off with the testing data would be preprocessed in the same way as the training data and the validation data. This ensures that the model is tested on the same type of data that it was trained on. Then the model is tested on a different set than the training and the validation. This step ensures that the model can generalize to new data that it has not seen before. Finally, the accuracy of the model will be calculated and compared to baseline accuracy – it is the accuracy that would be achieved if the model simply guessed the most common class for every image.

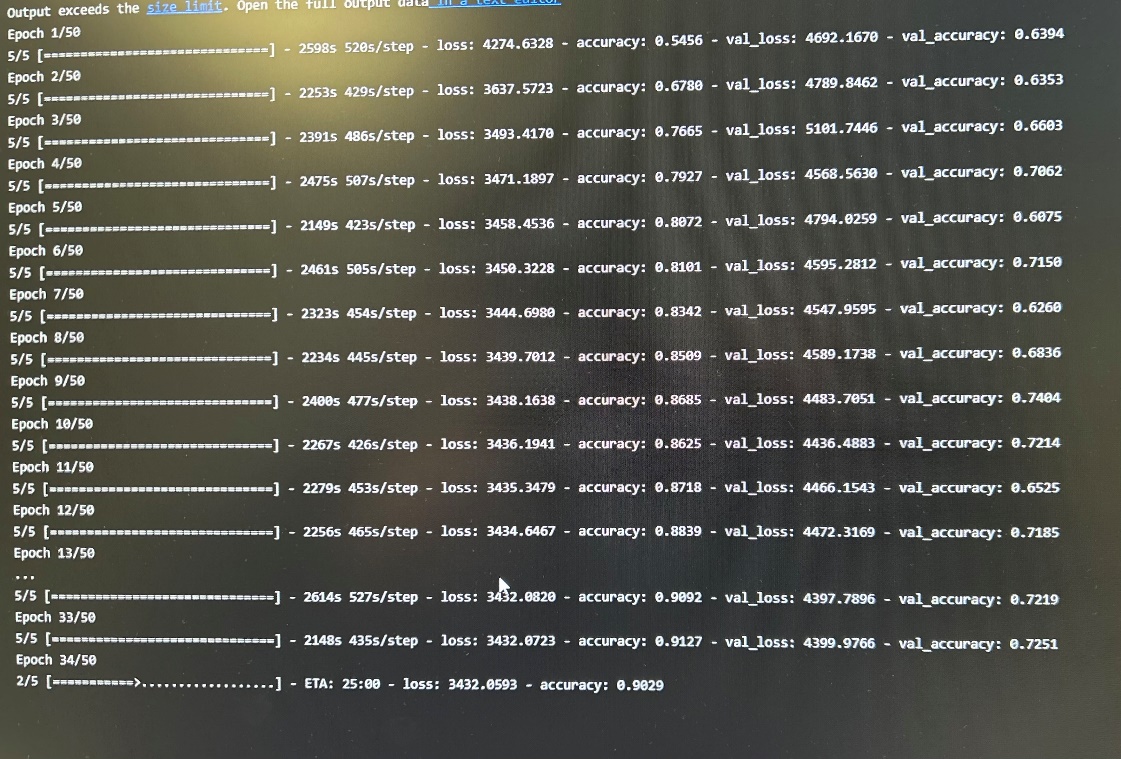
(566)

# Implementation and Testing:

Firstly, we would like to note that anything that applies to the training set is also applied to the validation and testing sets.

To discuss the implementation of the model we need to first look more closely at the steps required to build such a system. As with any machine learning model the first step is collect and clean the dataset. Collecting the dataset might seem like a mundane task but in reality, collecting the dataset includes having to split the data into three components. The training set, the validation set, and the testing set. To save processing time we have done the splitting manually be taking three random data folders from the entire FSOCO dataset, with the folder with the most amount of data is set as the training set, while the second and third most is set as the validation set and the testing set, respectively. When ‘cleaning’ or preprocessing the dataset, we have two main components. First is the image data, which must be processed and normalized, as in have each image in a standard size that cannot be different from the others which will guarantee us consistency in the data. In our program we have normalised every image to be 244 \* 244 which takes into account the aspect ratio of the images. The annotation for each image is the data that outlines the bounding boxes for the objects in each image. This data includes the xmin, ymin, xmax and ymax. The x and y min refers to the top left corner of the bounding box. While the x and y max refer to the bottom right point of the bounding boxes. This data helps as it is the main component when it comes to training the object detection model. These bounding boxes points allow the program to learn, predict and/or decide where the object is and what its bounding box is. At first, we considered using the annotations as is, in JSON format, but it proved to be complex as the JSON file contained data that is not helpful. So, we decided to create a program that scrapes the important information from the JSON files which are as follows, Image\_name, class\_ID, xmin, ymin, xmax, ymax. This script is specifically designed to convert annotations in JSON format to a CSV format that is commonly used for data analysis and machine learning tasks. Annotations are typically used in computer vision tasks such as object detection, segmentation, and classification, where they provide ground truth information about the location and class of objects in an image. JSON is a widely used format for storing annotations because it is flexible and easy to work with. However, CSV format is preferred for data analysis and machine learning because it can be easily loaded into tools such as Excel, pandas, and scikit-learn. The script reads all the JSON files in a specified directory and extracts the information that is relevant for the CSV format. This includes the image filename, class, and coordinates of each bounding box. A bounding box is a rectangular area that encloses an object of interest in an image. The coordinates of the bounding box are specified as the minimum and maximum values of the x and y coordinates. The script uses the **os** module to access the file system, the JSON module to read JSON files, and the csv module to write CSV files. It creates a CSV file with a header row that includes the column names filename, class, xmin, ymin, xmax, and ymax. It then iterates over each JSON file in the directory, reads its contents, extracts the relevant information, and writes it to the CSV file using the **writerow** method. One problem encountered when using the CSV format though was the class ID. When processing the data and placing it into a numpy array. We assumed that the data is going to be in integer type, but the class ID was only written as a string which gave an error when the training model. To solve this problem, we assigned each class ID to a number in a dictionary. We chose a dictionary because its speed was vital when preprocessing the data compared to a list. This correlation between the class ID and the numbers allowed us to convert each class to that assigned number and placed in the numpy array which solves the data type issue. After we have finished cleaning the data, the program starts out first with importing the libraries to be used, defining the constants as well as defining the **number** of samples for each set. The main constants or Hyperparameters are **IMAGE\_SIZE, BATCH\_SIZE, NUM\_EPOCHS,** and **LEARNING\_RATE.** These parameters affect the accuracy of the model and change with every run to finetune it. We also calculate the training samples by using this one line of code:  
This line of code calculates the total number of training samples by summing the number of files in all the subdirectories of a specified training directory. It uses the **os.walk** function to traverse the directory tree and return the root, directories, and files in each subdirectory. The **len** function is applied to the files list for each subdirectory to count the number of files. The sum function then adds up all the counts to get the total number of training samples. The variable **num\_training\_samples** store the total number of training samples and can be used to specify the batch size, number of epochs, or other parameters for training a machine learning model. This same line of code applies to the validation and testing samples as well. The **training\_dir** contains the file path of the training directory. This also applies to the validation and testing directory. the next part is the preprocessing of the data.

This block of code loads the training data and annotations for a machine learning model. It uses the **os.walk** function to traverse the directory tree and look for image files in the specified training directory. For each image file, it reads the corresponding annotation file, extracts the bounding box coordinates and class labels, and stores them in a list. The code first initializes two empty lists, **training\_images** and **training\_annotations**, to store the image data and annotations, respectively. Then, it iterates over each file in the training directory and checks if it is an image file (with the extension **.png or .jpg**). If it is, it reads the corresponding annotation file and extracts the relevant information. The image data is loaded using the cv2.imread function from the OpenCV library, which reads an image from a file and returns it as a NumPy array in the BGR color space. The code then converts the color space to RGB using cv2.cvtColor and resizes the image to a specified IMAGE\_SIZE. The resized image is appended to the **training\_images** list. The annotation data is loaded from a CSV file using the open function in read mode. The next function is called to skip the first line, which is the header row. The remaining lines are read using the **readlines** method and parsed to extract the filename, class name, and bounding box coordinates. The class name is converted to a class ID using a dictionary of class labels. The bounding box coordinates and class ID are appended to the annotation list. Finally, the image data and annotation data are converted to NumPy arrays using the **np.array** function. The resulting arrays, **training\_images** and **training\_annotations**, can be used to train a machine learning model for object detection, segmentation, or classification tasks. After processing the data and placing it all into a numpy array we then must normalize all the image sizes by dividing the image arrays by 255 which turns all the samples into the same size. We then verify the shapes of all the arrays by using the **shape()** method. The data is then merged into a tensorflow dataset and turned into a batch based on the **BATCH\_SIZE** so that makes it readable for the model.The final part is defining the model architecture and compiling it to be trained on the **train\_ds,** at the same time it is also validated. Later on it is tested and evaluated on the accuracy metric where the predictions and the testing annotation are compared against each other.  
  
This code defines and trains a convolutional neural network (CNN) using TensorFlow. The architecture of the model consists of three **Conv2D layers**, each followed by a **MaxPooling2D layer**, and a **Flatten layer**, which flattens the output from the previous layer into a **1D array**. The last two layers are a **Dense layer** with a **softmax** activation function and a **Reshape layer**. The model is compiled with the Adam optimizer using a learning rate specified by the constant **LEARNING\_RATE**. The loss function used is **categorical\_crossentropy**, and the evaluation metric is accuracy. The training is carried out for a specified number of epochs (**NUM\_EPOCHS**) and batch size (**BATCH\_SIZE**) using the training set (**training\_images and training\_annotations**). The validation set (**validation\_images and validation\_annotations**) is also passed during the training for evaluating the model's performance. After training, the model is evaluated on a test set (**testing\_images and testing\_annotations**) using the evaluate method of the model. The accuracy of the model is calculated using the **accuracy\_score** function from scikit-learn and printed to the console. Finally, the trained model is saved to the **models** directory using the save method of the model. This saved model can later be loaded and used for making predictions on new data.

The testing phase of the project proved to be our biggest obstacle, as the computational resources required to train and save the model were lacking. We were able to gather screen shots of the training epoch process 

The accuracy of the training set can be seen to have reached as high as 91% while the validation was at 76%. Although the loss value seems high at 4937, we later saw a decrease to 3752 at the 50th epoch, although no screenshots were taken at the time. This proves that the model was successfully being trained on the data, however the lack of resources prevented us from being able to save the model for later testing. But we can deduce from these training results that the model would have preformed beyond expectation in detecting the objects while mostly classifying the objects correctly as well. The testing set which the model was evaluated on showed an accuracy of 93% with a loss value of 3842.

# Evaluation and lessons learned:

Throughout our project, we learned many valuable lessons as a team. We were able to identify areas where we excelled and areas where we struggled, and we used these insights to improve our collaboration and team skills.

One of the areas where we performed well was in our planning and coordination efforts. From the outset, we developed a comprehensive responsibility matrix that clearly defined each team member's role and responsibilities. This matrix helped us to avoid any confusion or overlap and ensured that everyone knew exactly what was expected of them. As a result, we were able to work efficiently and effectively, staying on track to meet our deadlines. Another area where we excelled was in our technical expertise. We were able to leverage our knowledge of computer vision, machine learning, and deep learning algorithms to design and develop an object detection program that met the specific requirements of an F1 racing vehicle. We also collaborated with the Bristol AI team from the University of Bristol to ensure that the program was integrated seamlessly with the vehicle's existing systems and could operate in real-time under extreme racing conditions.

However, we also identified areas where we faced challenges. One of the main issues we encountered was uneven effort from our members. While some members were highly engaged and put in a great deal of effort, others were less involved and did not contribute as much as we had planned. This lack of engagement and participation created some delays. To address this issue, we held regular team meetings where we discussed progress and identified areas where additional support was needed. We also worked to create a more inclusive environment by holding meetings during the week when all team members would be available.

We also encountered some challenges during the project that required us to adapt and learn new skills. One of the main challenges we faced was in the optimization of the program's performance. Due to the high-speed nature of F1 racing, the program had to be able to detect and classify objects in real-time, with minimal delay or latency. This required us to optimize the program's algorithms and code to ensure that it could run efficiently and effectively on the limited computing resources available on the vehicle.

Another area where we encountered challenges was in our lack of knowledge in certain areas. We initially started with the YOLO object detection model but later found out that SSD would be more suitable for our project. However, due to the requirements set by the stakeholder, we switched back to YOLO, later our programmers have figured out that we did not have enough resources to train the model using YOLO which caused our researchers to figure out another method we can use to train our model and figured that the best route we can take with the resources available to us would be CNN. To overcome this issue in the future, we would ask the group leader to create a folder that would contain all the necessary documents and articles for us to fully understand the technical aspects of the project. This would allow the team to take a couple of steps forward.

One of the key lessons we learned was the importance of effective communication. We found that clear and timely communication among team members was crucial to ensuring that everyone was on the same page and that tasks were completed on time. We also learned that feedback and constructive criticism were essential to improving our work and making necessary adjustments. By providing constructive feedback to each other, we were able to identify areas where we needed to improve and make the necessary changes.

The development of an object detection program for an F1 racing vehicle was a challenging and rewarding experience for our team. we learned many valuable lessons that helped us to improve our collaboration, technical skills, and overall effectiveness as a team.

We identified potential risks and challenges and developed contingency plans to mitigate these risks. This approach helped us to stay focused and organized throughout the project and ensured that we were able to deliver the program within budget.

To overcome this challenge, we worked with the Bristol AI team to identify the specific hardware requirements of the program and developed custom optimizations for the program's algorithms and code. We also implemented a rigorous testing and validation process to ensure that the program met the performance requirements and was robust enough to handle the unpredictable and dynamic racing environment.

We realized that in order to develop a program that could meet the performance requirements of an F1 racing vehicle, we needed to have a deep understanding of the hardware and software systems involved. We also learned the value of continuous testing, as this helped us to identify and resolve performance issues early in the development process.

We also learned the importance of maintaining a detailed and well-organized documentation system. As we progressed through the project, we realized that keeping track of our progress and ensuring that all team members were up to date was crucial to success.

We learned the importance of effective communication, collaboration, and planning. We also discovered the importance of maintaining a flexible mindset and being open to change. Most importantly, we learned that success is only possible when all team members are equally committed and engaged in the project. We are proud of what we accomplished and grateful for the opportunity to have worked on this project.

Another lesson we learned was the importance of being adaptable. As we encountered unexpected challenges and had to change our approach, we realized that flexibility was crucial to our success. We learned to be open to new ideas and approaches and to adjust our plans as needed to ensure that we were still able to achieve our objectives.

The project topic is great for broadening our horizons since it involves object detection which none of us had attempted working on before it, this topic also involved co-operation with the Bristol AI team members from University of Bristol, not only did our teamwork amongst ourselves we also managed to work alongside students from the University of Bristol.

Overall, the project was a valuable learning experience for all team members. We identified areas where we excelled and areas where we faced challenges, and we used these insights to improve our collaboration, technical expertise, and project management skills. Moving forward, we will continue to build on these lessons learned and work to create even more effective and efficient teams in the future.

In conclusion, our project was a valuable learning experience for all team members. We identified areas where we excelled and areas where we faced challenges, and we used these insights to improve our collaboration and achieve our objectives. Moving forward, we will continue to build on these lessons learned and work to create even more effective and efficient teams in the future. We are very grateful for this opportunity to work with the Bristol AI team on this project, and we are confident that the lessons we learned will serve us well in future group projects. The development of an object detection program for an F1 racing vehicle was a challenging but highly rewarding project.

(1236)

**References:**

[1] : Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. [online] Available at: <https://arxiv.org/pdf/1506.02640v5.pdf> [Accessed 12 Feb. 2023].

[2] : Redmon, J. (2012). *YOLO: Real-Time Object Detection*. [online] Pjreddie.com. Available at: <https://pjreddie.com/darknet/yolo/>

[3] : GeeksforGeeks. (2021). *Difference between YOLO and SSD*. [online] Available at: <https://www.geeksforgeeks.org/difference-between-yolo-and-ssd/>.

Institution of Mechanical Engineers. (2022). Formula Student Artificial Intelligence. Retrieved from <https://www.imeche.org/events/formula-student/team-information/fs-ai>

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu & Alexander C. Berg (2016). SSD: Single shot multibox detector. In European conference on computer vision

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

Vödisch, N., Dodel, D. and Schötz, M. (2022) FSOCO: The Formula Student Objects in Context Dataset. SAE International Journal of Connected and Automated Vehicles. [online]. 5 (1), pp.12-05-01–0003.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767.

FSOCO Dataset. (n.d.). Retrieved March 13, 2023, from <https://www.fsoco-dataset.com/>

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In European Conference on Computer Vision (pp. 21–37). Springer.

Zheng, L., Wang, X., Wang, Q., Wang, S. and Liu, X. (2021) A Fabric Defect Detection Method Based on Improved YOLOv5In: 2021 7th International Conference on Computer and Communications (ICCC). [online]. 2021 7th International Conference on Computer and Communications (ICCC) 620–624.

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), 1345-1359.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) Challenge. International Journal of Computer Vision, 88(2), 303-338. <https://doi.org/10.1007/s11263-009-0275-4>

Hosang, J., Benenson, R., & Schiele, B. (2017). Learning non-maximum suppression. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 6549-6558). <https://doi.org/10.1109/CVPR.2017.693>

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In Proceedings of the European Conference on Computer Vision (pp. 740-755). <https://doi.org/10.1007/978-3-319-10602-1_48>

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning Deep Features for Discriminative Localization. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2921-2929.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., & Yoo, Y. (2019). CutMix: Regularization strategy to train strong classifiers

‌(5-7)

**Appendices:**

Graphical user interface, text, application

Description automatically generated

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Task Commencing Feb 6th, 2023 | Feb  6TH- 17TH | Feb 20th – 3rd Mar | Mar 6th – 17th | Mar 20th -31st |  |  |  | DEPENDENCIES |
| Planning |  |  |  |  |  |  |  | None |
| 1. Group leader | AA | AbA |  |  |  |  |  |  |
| 1. Define project scope, requirements and goals |  |  |  |  |  |  |  | None |
| 1. Create project plan |  |  |  |  |  |  |  | Define project scope, requirements, goals |
| Research and Development |  |  |  |  |  |  |  | Create project plan |
| 1. Conduct literature review |  |  |  |  |  |  |  | None |
| 1. Research existing object detection algorithms | MU  TS | YA |  |  |  |  |  | Conduct literature review |
| 1. Prototype object detection |  |  |  |  |  |  |  | Research existing object detection algs |
| 1. Coding | AbA | AA  TS |  |  |  |  |  |  |
| Implementation |  |  |  |  |  |  |  | Prototype object detection |
| 1. Develop Object detection system |  |  |  |  |  |  |  | None |
| 1. Integrate with vehicle software |  |  |  |  |  |  |  | Develop object detection system |
| 1. Testing & Evaluation |  |  |  |  |  |  |  | Integrate your vehicle with software |
| 1. Conduct system testing |  |  |  |  |  |  |  | None |
| 1. Analyze test results and identify improvements |  |  |  |  |  |  |  | Conduct system testing |
| 1. Report | YA | MU |  |  |  |  |  |  |

‌

‌