
MCAST

# Real-Time Object Detection and Classification for Low Vision Accessibility in Video Games

*Tyrone Camilleri*

*Supervisor: Ivan Briffa*

**June - 2024**

**A dissertation submitted to the Institute of Information and Communication Technology in partial fulfilment of the requirements for the degree of BSc (Hons) in Software Development**

# Authorship Statement

This dissertation is based on the results of research carried out by myself, is my own composition, and has not been previously presented for any other certified or uncertified qualification.

The research was carried out under the supervision of Ivan Briffa Real-Time Object Detection and Classification for Low Vision Accessibility in Video Games, Tyrone Camilleri

..... .....

Date Signature

# Copyright Statement

In submitting this dissertation to the MCAST Institute of Information and Communication Technology, I understand that I am giving permission for it to be made available for use in accordance with the regulations of MCAST and the Library and Learning Resource Centre. I accept that my dissertation may be made publicly available at MCAST's discretion.

..... .....

Date Signature

# Acknowledgements

I would like to express my gratitude to Ivan Briffa, my mentor, for providing guidance and support throughout the research process.

I want to thank my friends and family for helping me along the way.

Furthermore, I am grateful to APS Bank and my colleagues, especially for allowing me the flexibility needed to complete this research. Their encouragement and support were invaluable.

# Abstract

This section should clearly state what the study is about, summarizing how it was carried out and what the results were. References are not to be included in the abstract. It should present only the essentials of the work in general.

TODO Add this section

**Keywords:** Dissertation, keywords.

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **NN** | Neural Network |
| **ML** | Machine Learning |
| **DL** | Deep Learning |
| **FCN** | Fully Convolutional Network |
| **CNN** | Convolutional Neural Network |

FIXME Add this section

# Chapter 1: Introduction

## 1.1 Introduction

According to a report from the World Health Organization [1], at least 2.2 billion people worldwide have a form of vision impairment, which underscores the need for accessible gaming solutions for the visually impaired. For individuals who are completely blind, engaging in games that rely on visual cues is difficult, Instead they have to depend on auditory cues, speech, or specialized equipment. On the other hand, those with low vision can detect light and possibly motion but are limited in what they can, distinguish [2], making traditional gaming experiences challenging.

The gaming industry, is a rapidly growing market valued at over $249 billion in 2022, has begun to recognise the importance of accessibility features for players with various disabilities [**?**]. Despite this progress, not all companies or developers have adhered to these practices. This is often due to the lack of knowledge on how to develop accessible features, insufficient resources, limited time to prioritize accessibility, and a general lack of awareness.

The aim of this study is to try to improve the accessibility of video games for individuals with low vision by utilising computer vision (CV). The research focuses on developing and evaluating a model capable of detecting and classifying objects within video game mini-maps to enhance spatial awareness.

## 1.2 Purpose Statement

This study included several variables, which are divided into three groups: dependent, independent, and control. Independent variables included the background of the mini-map, the random position of the objects, and the number of mini-map icons. Dependent variables will be the measured outcomes, such as the accuracy of the object detection model and the speed and processing time of the model. Control variables included, such as the style of the icons, the resolution of the mini-map images, the programming environment, and the hardware of the machine that is running the model. By carefully controlling these variables, the study aims to ensure that the results are reliable and that the effects of the variables on the dependent variables are accurately measured.

## 1.3 Hypothesis and Research Questions

The hypothesis for this research, is that using computer vision, it is possible to extract information from video games to address challenges experienced by individuals with low vision while playing video games. To address this hypothesis, the following research questions were asked:

- How does the model adapt to static objects, versus moving objects?
- Can the model be adapted for various game styles while still being effective?
- How accurate is the model in detecting objects in real-time game scenarios?

# Chapter 2: Literature Review

In this chapter, we aim to review key topics and relevant work, The Literature review is split into 3 sections. The first part is focuses on important work related to computer vision & artificial intelligence. The second part aims to investigate and discuss the visual impairment spectrum, and finally the third part reviews existing technologies and spatial awareness tools.

## 2.1 Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that focuses on creating a system that learns autonomously without being specifically programmed; instead, it learns from input to imitate the way humans learn. These inputs are normally in the form of datasets that are comprised of relevant information that is the baseline to make the system accurately train and make predictions based on what it has learned. Machine learning systems are commonly trained using three different methods: supervised learning, semi-supervised learning, and unsupervised learning [3, 4].

### *2.1.1 Machine Learning Methods*

**Supervised Learning**

Supervised learning relies on labelled datasets to teach models how to classify or predict outcomes [5]. A labelled dataset is when the target outcome or prediction

is included in the dataset alongside other with other variables [6]. Supervised learning aims to train models to classify unseen or new data and make predictions from an already seen training dataset [7].

**Unsupervised**

Unsupervised learning is the opposite, where it uses an unlabelled dataset to analyse and cluster data [5]. Unlabelled datasets do not include classification labels; instead, it is the algorithm's purpose to automatically develop classification labels [8].

**Semi-Supervised**

Semi-supervised learning is the combination of unsupervised or supervised learning using both labelled and unlabelled data (partially labelled data). [9] This approach is useful when unlabelled data is present, but labelling it requires significant effort. [3]

## 2.2 Computer Vision

CV is a field of AI that enables machines to be able to see and extract data, simulating human vision. [10]. As shown in the below figure 2.1, Pandey [11] in a 2023 paper described the CV workflow as the following CV works by using an algorithm to process data from visual data, such as images or video frames. After the preprocessing stage is complete, relevant features are extracted from the prepossessed data; these features include aspects such as colour, texture, shape, and edges. The computer vision model utilises algorithms, models, and other

techniques for tasks like image classification, object detection, and segmentation, leveraging the extracted features for analysis. The post-processing stage further refines these results, potentially involving improving the accuracy of the model. The final output can include classifications, bounding boxes, and pixel labels, depending on the desired outcome.

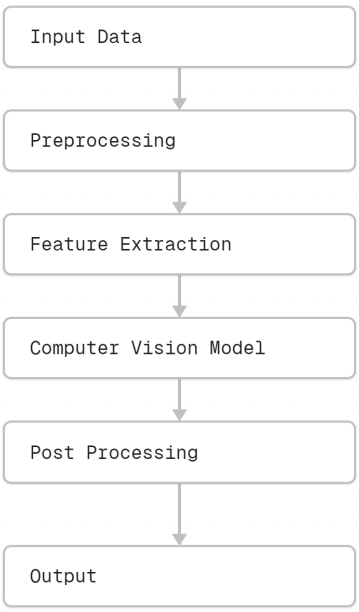


***Figure 2.1:*** *Computer vision flow (Pandey, 2023)*

### *2.2.1 Classification Techniques*

#### Object Detection

The purpose of an object detection model is to locate and classify an object presented in the image. Generally, the objects are labelled with the predicted class and confidence level, and the object is usually surrounded with a bounding

box. [12] Object detection methods generally fall under two types.

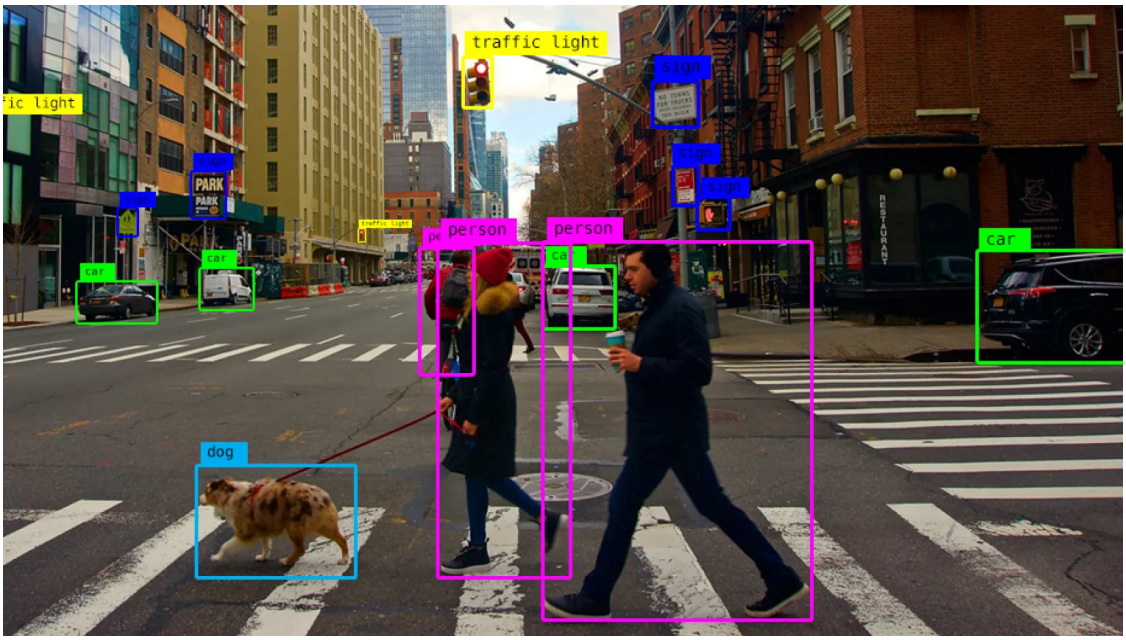


***Figure 2.2:*** *Object Detection (How to implement object detection using Deep Learning, 2023)*

### Region Proposal Network

Region Proposal Network (RPN) takes an image as input, generates region proposals, and uses a fully convolutional network to process these proposals. It slides a small network over the convolutional feature map and uses box-regression and box-classification layers. RPN generally outputs rectangular bounding boxes, each with a confidence level score [12, 13].

### Regression/Classification based

A regression-based/classification-based object detection algorithm takes an image as input, processes it through a unified network, and directly outputs detection results. It divides the input image into grids, confirms point positions and confidence levels, and assigns category scores to each grid, finally outputting bounding

boxes with associated confidence scores and classes [12, 14].

### Image Segmentation

Image segmentation involves detecting, recognising, and segmenting objects within an environment [15, 16]. Segmentation can be implemented using different types of techniques, including thresholding, region growing, edge detection, and active contours. This is used in medical image analysis and other various applications [16–18].

***Figure 2.3:*** *Image Segmentation (Potter, Image segmentation: The Deep Learning Approach 2022)*

### Image Classification

The image classification involves various algorithms and techniques to identify and categorise image features. The process of image classification involves several steps. First, relevant image datasets are collected and preprocessed through

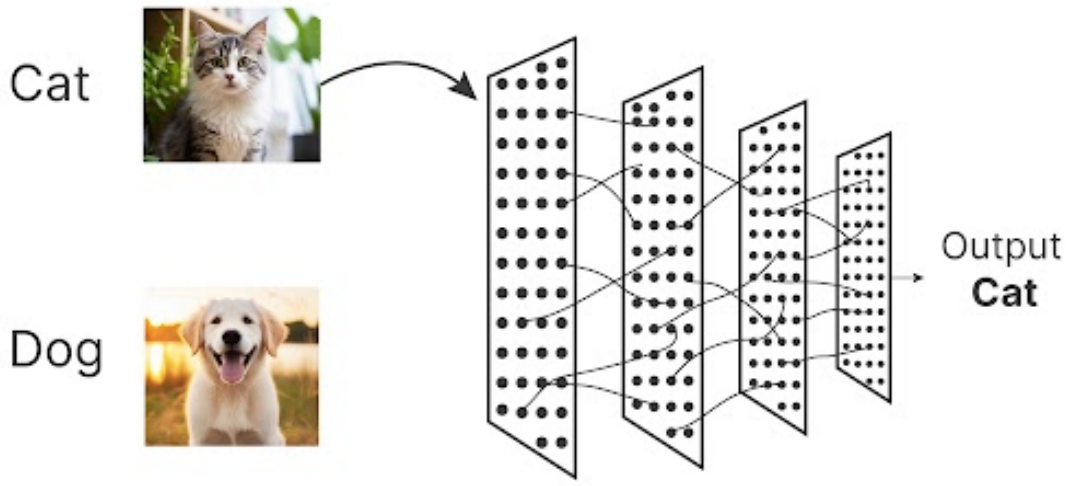


***Figure 2.4:*** *(Build your first image classification model in just 10 minutes!, 2024)*

resizing, cropping, greyscale conversion, and normalisation. Critical features like edges and textures are extracted and selected to reduce complexity and improve accuracy. An appropriate model, such as support vector machine (SVM) or neural networks, is chosen and trained using annotated datasets. Once deployed, the model classifies new images for practical applications [19].

### *2.2.2 Object Detection Algorithms*

In a 2023 study, Setia et al. [20] examined a broad range of object detection algorithms, and they reviewed and discussed the applicability, performance, and efficacy of these methods of detection. The review thoroughly assesses various algorithms, such as Convolutional Neural Networks (CNNs), R-CNN, Fast R-CNN, Faster R-CNN, Single Shot Detector (SSD), and You Only Look Once (YOLO).

### Single Shot Detector

The SSD model was developed in 2016 by Liu et al. [21]. The SSD model makes use of a CNN to generate bounding boxes and scores for object classes. The model applies multiscale feature maps, enabling the prediction of objects of various sizes and aspect ratios across different resolutions. Small convolutional filters are used in this method to guess the categories and bounding box offsets for each feature map location. This combines detections from layers with different scales. This design achieves a detection speed of 59 frames per second (FPS) with a 300×300 input size. The SSD model did better in the VOC2007 test set, with a mean average precision (mAP) of 74.3%. It showed better scores than the Faster R-CNN model, which got 73.2% mAP at just 7 frames per second.

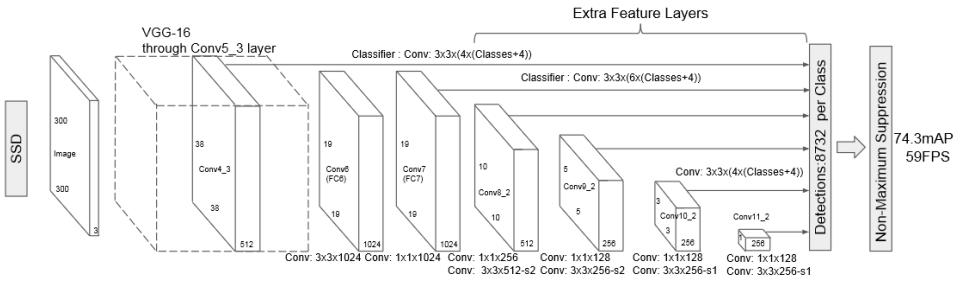


***Figure 2.5:*** *Overview of SSD Model (Liu et al. 2016)*

### Faster Region Convolutional Neural Networks

In their 2015 paper, Ren et al. [13] developed Faster R-CNN, a network for object detection that works in two main steps. First, a CNN processes an input image to create feature maps. Then, an RPN scans these feature maps to suggest possible regions where objects might be. The suggested regions are changed to remove overlaps. Next, these regions are resized to a fixed size using Region

of Interest (ROI) pooling. The Faster R-CNN detector then classifies the resized regions and refines their boundaries. The CNN, RPN, and Faster R-CNN share the same layers, allowing the whole system to be trained together, making it both accurate and efficient.

**You Only Look Once**

The YOLO model introduced by Redmon et al. [22] streamlines object detection by framing it as a single regression problem, simplifying the pipeline, and improving real-time performance. The YOLO model uses a single CNN to predict bounding boxes and class probabilities directly from the full image in one evaluation. The network has 24 convolutional layers, followed by 2 fully connected layers, and is based on the GoogleLeNet model. This design allows YOLO to process images at 45 fps, and a faster variant, Fast YOLO, reaches up to 155 fps. The input image is divided into a grid, and each grid cell predicts multiple bounding boxes, their confidence scores, and the class probabilities of the objects within them. On the PASCAL VOC dataset, YOLO achieved a mAP of 63.4%, while Fast YOLO reached an mAP of 52.7%. Despite making more localization errors compared to methods like R-CNN, YOLO reduces false positives and background errors, leading to more accurate detections overall. Because of its speed and precision, the model can be used in real-time scenarios where accurate and timely object recognition is essential, such as autonomous systems and video processing. High-speed processing and competitive accuracy are combined in YOLO's unified and efficient methodology.

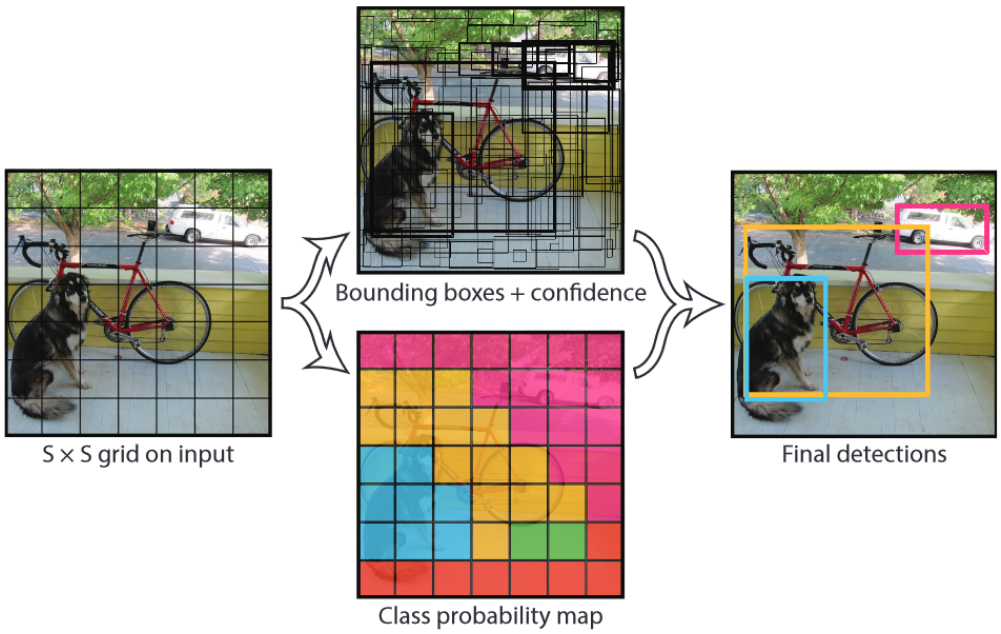


***Figure 2.6:*** *YOLO, (Redmon et al. 2016)*

**Analysis of Different Object Detection Techniques**

A compression of the previously reviewed algorithms (Faster-RCNN, YOLO, SSD) took place to discover which one is the most suitable for our needs.

In their 2021 study, Tan et al. [23] compared three object detection models, YOLO v3, Faster R-CNN, and SSD for real-time pill identification. The study emphasizes the importance of accurate and efficient pill identification in ensuring patient safety and resource management in hospital pharmacies. The authors trained each model on a dataset comprising over 5000 pill images and evaluated their performance using metrics such as mAP and FPS. Their findings indicate that while Faster R-CNN achieves the highest mAP (87.69%), YOLO v3 offers superior detection speed, processing 51 images per second with a MAP of 80.17%. On the other hand, SSD, although faster than Faster R-CNN, does not

outperform YOLO v3 in either mAP or FPS. The study concludes that YOLO v3 is the most suitable model for real-time pill identification applications in hospital settings, where speed and accuracy are crucial.

| **Algorithm** | **Precision (%)** | **Recall (%)** | **F1 (%)** | **mAP (%)** | **FPS** |
|---|---|---|---|---|---|
| YOLO v3 | 69.13 | 80.19 | 70.14 | 80.17 | 51 |
| Faster R-CNN | 62.19 | 94.24 | 78.23 | 87.69 | 7 |
| SSD | 63.17 | 88.69 | 72.13 | 82.41 | 32 |

***Table 2.1:*** *Performance Comparison of Object Detection Models (Tan et al. 2021)*

Li et al. [24] conducted a comparative evaluation of three CNN-based models Faster R-CNN, YOLO v3, and SSD for detecting agricultural greenhouses from high-resolution satellite images. The study employed transfer learning and fine-tuning to train the models. Their evaluation results indicated that YOLO v3 achieved the best performance in terms of mAP and FPS. Specifically, YOLO v3 obtained an mAP of 90.4% and an FPS of 73, outperforming Faster R-CNN and SSD, which had mAPs of 86.0% and 84.9%, and FPS values of 12 and 35, respectively.

| **Algorithm** | **Precision (%)** | **Recall (%)** | **F1 (%)** | **mAP (%)** | **FPS** |
|---|---|---|---|---|---|
| Faster R-CNN | 85.2 | 86.8 | 86.0 | 86.0 | 12 |
| YOLO v3 | 91.0 | 89.8 | 90.4 | 90.4 | 73 |
| SSD | 83.0 | 85.0 | 84.0 | 84.9 | 35 |

***Table 2.2:*** *Comparison of Faster R-CNN, YOLO v3, and SSD (Li et al. 2020)*

As noted by Setia et al. [20], while each algorithm has its strengths and weaknesses, YOLO is particularly noted for its speed, making it ideal for real-time detection tasks, whereas Faster R-CNN is preferred for applications requiring high accuracy; while SSD offers a balanced compromise between speed and ac-

curacy.

## 2.3 Spatial Awareness Tools & Other Existing Technologies

The study in 2019 by Andrade et al. [25] identifies three primary methods through which games convey information to visually impaired (VI) players: text, sounds, and radar.

### *2.3.1 Text*

Andrade et al. [25] describe text as how information is presented in a way that works for screen readers. Screen readers are an assistive technology that interprets written information into either spoken words or braille, providing VI players with access to written information. This level of accessibility is important to allow players to understand narratives, instructions, and other written elements within the game.

### *2.3.2 Audio*

Andradel et al. & Nees and Lebman [25,26] both describe auditory icons, earcons, and other auditory cues to convey information. Auditory icons are representational sounds that mimic real-world noises such as a car crash, while earcons are artificial sounds like beeps or musical tones used to signify specific events or actions within the game. These auditory cues help VI players understand and navigate the game environment by providing essential information through sound.

### *2.3.3 Radar*

According to Andradel et al. [25], radar in video games works similarly to a cane that people with VI typically use in real life. It plays different sounds when detecting open spaces, walls, or other objects, thereby providing spatial awareness and aiding navigation within the game. This method allows VI players to form a mental map of their surroundings and interact more effectively with the game world.

### *2.3.4 Spatial Awareness Tools*

**NavStick**

Nair et al. [27] developed a self-directed audio tool called NavStick, aimed at enhancing the accessibility of 3D video games for visually impaired players (VIPs). The NavStick system repurposed a game controller's thumb stick to enable VIPs to look around within virtual environments by scrubbing the thumb stick to hear audio descriptions of objects in various directions. This tool addresses the limitations of current blind-accessible games, which often confine players to simplified 2D grids or rely on continuous and sometimes distracting audio cues. NavStick was found to help VIPs form more accurate mental maps of their surroundings compared to traditional menu-based surveying methods, offering them more freedom and agency within game worlds.

**Effectiveness and Limitations of Current Tools**

Nair et al. [28] implemented four spatial awareness tools (SATs) to enhance gameplay for visually impaired players (VIPs) within 3D video game environments: smartphone maps, whole-room shockwaves, directional scanners, and simple audio menus. The smartphone map lets players touch the map, the whole-room shockwave uses an audible pulse to tell players what's around them; the directional scanner lets players use a joystick to scan their environment.

Each tool mentioned has its limitation. The smartphone map can be difficult to navigate when constantly switching between the game and the smartphone. The whole-room shockwave can be overwhelming when by providing too much information at once. the directional scanner does not provide comprehensive spatial details. The simple audio menu can reduce the sense of exploration by revealing too much information at once. The simple audio menu is easy to understand, but it can make it difficult to explore if it gives too much information at once.

### *2.3.5 How Researchers and Developers are Making Video Games Accessible*

Researchers and developers are increasingly focusing on making video games accessible to players with disabilities. This effort aims to promote inclusively and ensure that all individuals, regardless of their perceptual, physical, or cognitive abilities, can enjoy and benefit from video games. Various accessibility features are being integrated into games to accommodate players with different needs, demonstrating a growing commitment within the gaming industry to address these

issues.

### PowerUp

PowerUp is a multiplayer, educational virtual world game designed with a range of accessibility features to ensure it is usable and enjoyable for individuals with various disabilities. Notable accessibility features of PowerUp include self-voicing text-to-speech output, which allows all textual information within the game to be read aloud, making the game accessible to players with visual impairments. Additionally, sound-based representations in the form of audio cues and sound-based representations of in-world objects and environments help players orient themselves in the virtual space, distinguish different locations, and notice changes in the game's state. These features make PowerUp a notable example of an accessible educational game, ensuring that it caters to individuals with perceptual, physical, and cognitive disabilities [29].

### Terraformers

Terraformers is a pioneering real-time 3D graphic game designed to be accessible to both Blind or Visually Impaired Persons (BVIPs) and fully sighted gamers. The game integrates a sound interface with 3D graphics to create an inclusive gameplay experience. The sound interface enhances the gameplay experience by providing audio feedback that complements the 3D graphics, making the game accessible to players with various vision levels. Terraformers' innovative approach to accessibility earned it the "Innovation in Audio Award" at the 2003 Independent Games Festival (IGF), highlighting its success in creating an inclusive

gaming experience [30].

### The Last Of Us Part II

The Last of Us Part II (TLOU2) is an action-adventure game developed by Naughty Dog and released in June 2020 for the PlayStation 4. It is widely regarded as the most accessible mainstream video game, featuring over 60 accessibility options. These features address various disabilities, including visual, auditory, and motor impairments. Notable visual accessibility features include high-contrast modes and text resizing, which make the game accessible to players with visual impairments. Auditory accessibility features, such as subtitles and visual cues for sound effects, assist players with hearing impairments. Additionally, customisable controls and input options cater to players with motor disabilities. Due to these comprehensive accessibility features, TLOU2 was recognized by the game awards and won an award in the "Innovation in Accessibility" category [31, 32].

## 2.4 Visual Impairment Spectrum

Schiefer et al. [33] highlighted that refractive errors are very common worldwide and significantly impact daily life. Common visual errors include myopia, hyperopia, astigmatism, presbyopia, and anisometropia. Myopia, or near-sightedness, causes distant objects to appear blurry. Hyperopia, or far-sightedness, makes it difficult to see close objects clearly. Astigmatism leads to distorted vision due to the uneven curvature of the eye. Presbyopia, which comes with ageing, results challenged focusing on close objects. Anisometropia is when each eye has

a different refractive power, which can lead to issues if not corrected. These errors are often diagnosed with simple tests like using a pinhole aperture, which helps identify whether the problem is a minor refractive issue or something more serious.

### *2.4.1 Challenges faced in Video games*

Blind and visually impaired players (BVIPs) face significant challenges in video gaming due to limited access to visual information, which can negatively impact their confidence and independence [34]. These players often encounter substantial obstacles as essential visual cues are typically not accessible to them, severely restricting their ability to fully engage with and experience games [35]. This issue primarily arises from the neglect of crucial accessibility features, such as audio cues and user-friendly interfaces [34].

Due to these limitations, BVIPs often gravitate towards specific game genres, including audio games, text-based games, choice-driven games, management games, or turn-based games. Games that involve 2D or 3D maps, require rapid and precise inputs, or feature excessive on-screen activity tend to make BVIPs feel disoriented or find the gameplay impractical [36]. Research highlights a significant scarcity of both single-player and multiplayer games designed with BVIPs in mind, resulting in blind players frequently relying on external support to navigate games intended for sighted players [37, 38]. This underscores a broader lack of urgency within the game development industry to create games that accommodate BVIPs.

Despite numerous advancements in video game accessibility, BVIPs continue

to struggle to find commercially available games that meet their needs [27].

### *2.4.2 Overcoming challenges*

Research suggests that one of the primary challenges BVIPs face in gaming is navigation. Incorporating audio cues leads to more accessible games and easier navigation; however, even audio feedback also has is limitations, such as the ability to pinpoint an objective or enemies in a scenario with overwhelming different audio feedback [38]. The task of navigation becomes easier when players have SAT (Spatial awareness tools) sighted players utilized a combination of vision and SAT such as mini-maps to navigate. BVIPs will often rely on the assistance of SAT if available when faced with a complex scenario and are unable to rely on audio cues alone. While everyone has different needs, the utilising of SAT allows information to be filtered, which allows BVIPs to navigate through environments than faster normally would have.

# Chapter 3: Research Methodology

## 3.1 Research Methods

This research follows a postpositivist world-view, utilising a monomethod, experiment-based approach to gather and statistically analyse data, which is inherently a quantitative approach. This strategy was favoured because it centres on determining the variables that influence the dependent variables. Creswell [39] favoured this strategy.

### *3.1.1 Hardware & Software Setup*

**Hardware Setup**

For this study, a Windows 11 (Home) machine with an Nvidia 3070 Graphics Processing Unit (GPU), an Intel i5-10400F as the Central Processing Unit (CPU) and 16 gigabytes of Random Access Memory (RAM) running on an NVMe Solid-State Drive (SSD). Including these hardware specifications supports reproducibility and provides a benchmark for assessing the CV model's efficiency and performance.

**Software Setup**

**Programming Language and Environment** We selected Python as our primary programming language due to its extensibility. The development environment is configured in an Anaconda distribution. Two Anaconda environments were created

with Python versions 3.9 and 3.11, respectively, chosen for their compatibility with common libraries.

**Libraries Used**

- **PyGame** [40]: A library that helps with the creation of video games. PyGame offers real-time graphics and event handling capabilities, allowing developers to manage game windows, handle user inputs, and render graphics. It simplifies tasks like loading and resizing images, handling game assets, and includes collision detection mechanisms and frame rate control. In this project.

- **Random Library**: This library offers functions for generating random numbers. It allows for the random generation of integers, floating-point numbers, and choosing random elements from a list. It is commonly used to add unpredictability to an application. Random Number Generation (RNG) allows us to construct dynamic situations that mimic uncertain real-world environments.

- **Ultralytics YOLO**: For object detection tasks, the YOLOv8 model from Ultralytics was used [41]. This model is known for its real-time analysis capabilities and adaptability across various computational environments. The Python library provided by Ultralytics was installed and utilized within the project for training and evaluating the object detection model.

- **pyttsx3:** This library developed by Bhat [42] is a Text-to-Speech (TTS) library which has the capacity to work offline without an internet connection.

## 3.2 Data Acquisition

### *3.2.1 Data Gathering*

**Manual Data Gathering** During our research, we engaged in manual data gathering, which involved collecting data through our efforts and interactions. This method was carried out at first, but then it seemed to be time-consuming and inefficient, making it impractical for large-scale implementation.

**Selecting a Game** After thorough consideration, "Call of Duty: Modern Warfare" was selected for the data gathering process. This game was chosen due to its simple and consistent mini-map, which is essential for the dataset.

**Recording Gameplay** Once we selected our game, the next step was to record gameplay. This was done to capture the mini-map in various scenarios and conditions, providing a diverse set of images to produce our dataset.

**Cropping Images** After we created the gameplay, the images were cropped to only showcase the mini-map. This step was performed so that the focus was solely on the mini-map, removing any unnecessary elements of the gameplay footage.

**Extracting Frames** Finally, the video footage was processed frame by frame to extract individual images of the mini-map. During this process, we also took an additional step to remove frames where the mini-map was not included. These frames form the basis of the dataset, capturing the mini-map in different states and conditions.

***Figure 3.1:*** *Mini-map from "Call Of Duty: Modern Warfare"*

**Automated Data Gathering Overview** Automated data gathering involved using scripts to collect, create, and annotate images with minimal human interaction. This method was effective for collecting large volumes of data quickly.

**Steps Involved** After analysing the mini-map from the manually gathered data and breaking down its features (which will be explained in the Data Exploration section), we decided to create a tool that would quickly generate a dataset using similar features to the manually collected mini-map data. This tool generates 2D sprites in random locations with random rotation and random opacity, with a random background from a folder or a simple colour.

**Tool Development** The program was created using Pygame and the Random library. It generates sprites for enemies and allies with random properties and saves the generated images and their corresponding labels.

**Synthetic Dataset** With our automated data gathering, we are essentially creating synthetic data. Nikolenko [43], described synthetic data as a valuable tool for training deep learning models in computer vision, robotics, and other fields,

with potential applications in various fields. Synthetic data are artificially generated data that closely resemble the patterns and characteristics of real-world data. Computer graphics, algorithms, and generative AI are techniques used to create diverse and representative synthetic datasets that supplement or replace traditional training data.

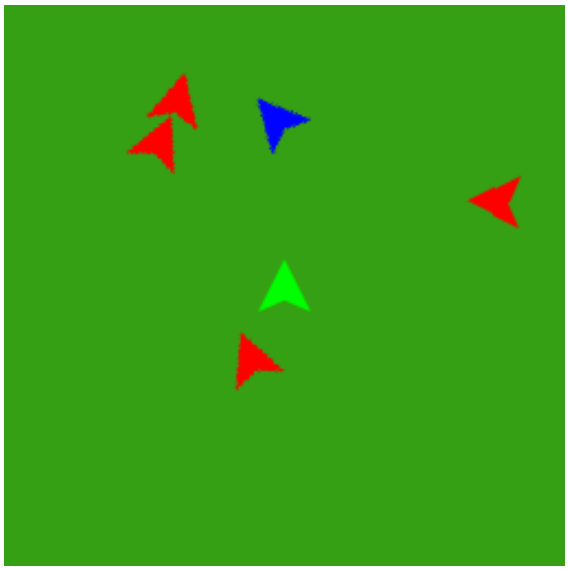

***Figure 3.2:*** *Synthetic image from custom tool*

### *3.2.2 Data Exploration*

Mini-maps provide a condensed view of larger environments to aid users in navigation and spatial awareness. Key features of a typical mini-map include icons or markers representing entities such as allies and enemies, objectives, or points of interest, often highlighted with distinct shapes or colours for quick identification. Additionally, mini-maps may feature dynamic elements, such as moving markers, to indicate the real-time positions of entities within the environment.

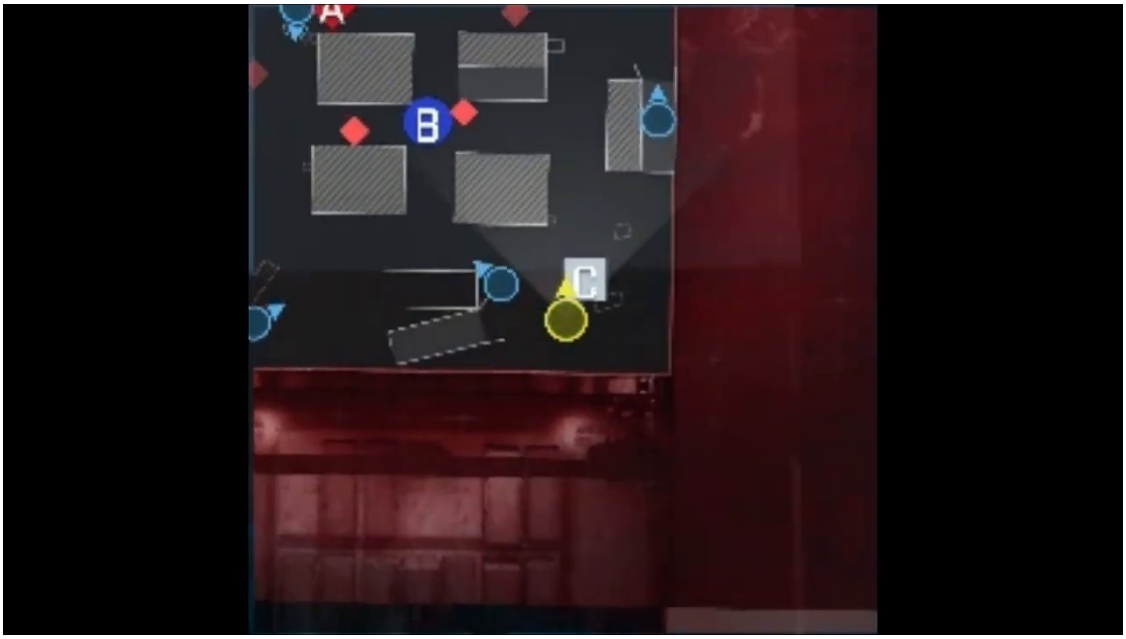


***Figure 3.3:*** *Example of Mini-map from "Call of Duty: Modern Warfare 2"*

### *3.2.3 Data Transformation*

**Manual Annotation**

**Overview** Following the generation of the dataset, these images undergo an annotation process using CVAT. This tool plays a role in providing a structured framework where objects within each mini-map are accurately located and properly categorized. The utilisation of CVAT ensures that the datasets are prepared with precise and reliable annotations, which are fundamental for the training and evaluation of the CV model later in the project.

**Setting Up CVAT** To construct a dataset using CVAT, a new project titled "Minimap_SAT" was created. The project also requires the definition of labels to categorize various icons found in the images. For this project, the labels "Enemy", "Ally", "Player" were created to identify all objects within the mini-map

images. The next step involves creating a new task within the project. This task also requires uploading the randomly generated images sourced from a mini-map generator.

**Annotation Interface** Upon accessing the specified task, navigation leads to a job interface. When the job page is launched, it shows the annotation interface, as shown in the below figure. This interface hosts various annotation tools which are positioned on the left sidebar. These tools are utilized to draw shapes around the icons to create bounding boxes. These bounding boxes are crucial as they help in accurately identifying and localizing each icon within the images. This precise localization is essential for training machine learning models to recognize similar icons automatically in new, unseen images. The accurate annotation of these icons with bounding boxes directly influences the effectiveness and accuracy of the model's performance in real-world applications.

FIXME Shorten the above

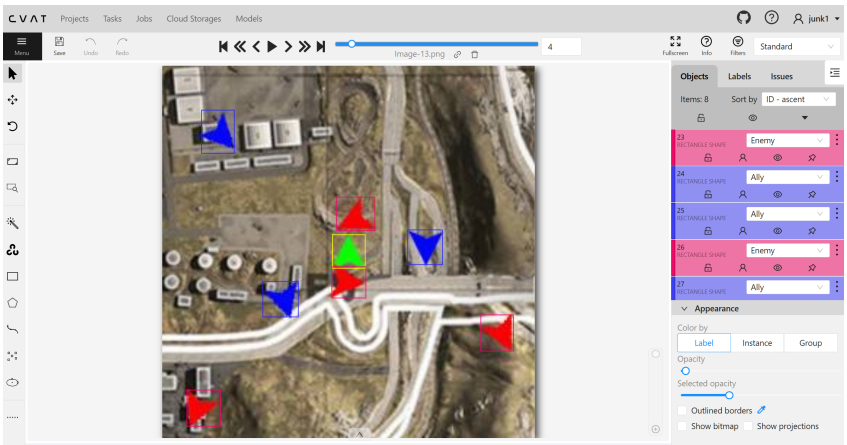


***Figure 3.4:*** *Annotation Interface*

Once the annotation is completed, the labels are exported and stored in YOLO format and are stored in an annotations' folder, which is located alongside where

images are stored.

**Automated Annotation — Object Detection**

**Overview** Given that we are creating the images and know the exact location and class of each object, we can automatically annotate the dataset as it is generated.

**Process** The process starts by generating images which are stored in an images' folder, alongside the annotation for each image which is stored in a label folder. The annotations are stored in the YOLO format.

**YOLO format** The YOLO format specifies the class and bounding box information for each object in the image. An annotation in the YOLO format consists of a single line per object, with the following values:

⟨object-class⟩ ⟨x-center⟩ ⟨y-center⟩ ⟨width⟩ ⟨height⟩

- **object-class**: The class of the object, represented by an integer.
- **x-center**: The x-coordinate of the centre of the bounding box, relative to the width of the image (a value between 0 and 1).
- **y-center**: The y-coordinate of the centre of the bounding box, relative to the height of the image (a value between 0 and 1).
- **width**: The width of the bounding box, relative to the width of the image (a value between 0 and 1).

- **height**: The height of the bounding box, relative to the height of the image (a value between 0 and 1).

An example annotation in the YOLO format:

1 0.722656 0.921875 0.101562 0.101562

In this example:

- The object class is 1.
- The x-coordinate of the centre of the bounding box is 0.722656.
- The y-coordinate of the centre of the bounding box is 0.921875.
- The width of the bounding box is 0.101562.
- The height of the bounding box is 0.101562.

**Automated Annotation — Image Classification**

**Process** The process starts by generating images which are stored in a folder called 'train', the images are stored in sub folders depending on the orientation of the player, such as "top", "top_left", "top_right", "left", "right", "bottom", "bottom_right", "bottom_left".

**Benefits** Automated annotation can significantly reduce the time and effort required to annotate large datasets. It also helps in maintaining consistency and accuracy across the annotations. Since we generate the images and know the

exact properties of each object, we can ensure that the annotations are both accurate and consistent.

## 3.3 Experimentation

### *3.3.1 Model Architecture*

For this research, we implemented two YOLO v8 models from Ultralytics [41]. We aimed to explore the performance and efficiency of these models in various conditions to better understand their practical applications. Ultralytics has various architectures as shown in the below figures 3.1 3.2; for our project, we chose 'YOLOv8n' and 'YOLOv8n-cls' because they are the smallest architecture and offer the fastest speed.

| Model | Size | mAP | CPU (ms) | A100 (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 |

***Table 3.1:*** *YOLOv8 Object Detection Model Performance*

| Model | Size | Top1 | Top5 | CPU (ms) | A100 (ms) | Params (M) | FLOPs (B) |
|---|---|---|---|---|---|---|---|
| YOLOv8n-cls | 224 | 69.0 | 88.3 | 12.9 | 0.31 | 2.7 | 4.3 |
| YOLOv8s-cls | 224 | 73.8 | 91.7 | 23.4 | 0.35 | 6.4 | 13.5 |
| YOLOv8m-cls | 224 | 76.8 | 93.5 | 85.4 | 0.62 | 17.0 | 42.7 |
| YOLOv8l-cls | 224 | 76.8 | 93.5 | 163.0 | 0.87 | 37.5 | 99.7 |
| YOLOv8x-cls | 224 | 79.0 | 94.6 | 232.0 | 1.01 | 57.4 | 154.8 |

***Table 3.2:*** *YOLOv8 Classification Model Performance*

### *3.3.2 Models Configuration*

#### Objection Detection Model Configuration

We created the following folder structure: a main folder named images with two subfolders, train for training images and val for validation images, to hold the images used to train the model and test its accuracy, respectively. In the same main directory where the images' folder resides, we set up another folder called labels that mirrored the structure of the images folder with train and val subfolders, but these contained text files with annotation data. A configuration file named config-ObjectDection.yaml was created to set up four important settings for the project: path for the exact location of the project's main directory, train and val for the locations of the training and validation data, respectively, and a specification that we had three classes: Allies, Enemy, and Player. An Example configuration file is shown below.

```
path: E:\Mini-mapObjectDetection\screenshots\20240517_150945
train: Images/train
val: Images/val

names:
0: enemy
1: ally
2: player
```

#### Classification Model Configuration

We needed to create the following folder structure: a main folder named images with eight subfolders, top, right, bottom, left, top_right, top_left, bottom_right, and

bottom_left, to hold the images used for the project. A configuration file named config-Classification.yaml was created to set up four important settings for the project: path for the exact location of the project's main directory, train and val for the locations of the training and validation data, respectively, and a specification that we had three classes: Allies, Enemy, and Player. An example configuration file is shown below.

```
path: C:\Mini-mapObjectDetection\screenshots\20240530_191933
train: Images/train
val: Images/val

nc: 8

names: ['top', 'top_left', 'top_right', 'left', 'right', 'bottom',
'bottom_left' 'bottom_right']
```

### Object Detection Model Training

We trained our YOLO model using the Ultralytics framework with a training script implemented in Python. The script initialises the YOLO model with the configuration file `yolov8n.yaml` and trains it using the dataset specified in the configuration file. The training process is configured to run for 50 epochs, with early stopping after 10 epochs of no improvement in validation metrics. The training utilises GPU device 0 for acceleration. Below is the script we used to train the object detection model, we trained with the below parameters once with pre-trained set to true and once as pre-trained false.

```
from ultralytics import YOLO
```

```
def main():
     yoloModel = YOLO('yolov8n.yaml')



     results = yoloModel.train(
        data=r"C:\Users\Abzsorb\Desktop\Mini-mapObjectDetection\src\config-ObjectDetection.yaml",
        epochs=300,
        weight_decay=0.0005,
        dropout=0.5,
        project='./src/Models/',
        pretrained=False,
        name='LargeDataset-300E',
        device=0,
        plots=True
    )




if __name__ == '__main__':
   main()
```

### Image Classification Model Training

For the image classification model, we utilised the same approach as the object detection model. Below is the script we used to train the image classification model:

```
from ultralytics import YOLO


model = YOLO('yolov8n-cls.yaml')


model.train(
    data=r"C:\Mini-mapObjectDetection\dataset\classifications\20240530_195850",
    epochs=50,
    weight_decay=0.0005,
    dropout=0.5,
    pretrained=false,
    project='./src/CVModels/Ultralytics/Models/Classification',
    name='Icon-Rotation-V1',
    device=0
)
```

**Prototype** In our study, we developed a real-time object detection system using the YOLOv8 model from Ultralytics to identify and track various entities. For each frame, the YOLOv8 model detected objects, extracting bounding boxes and class labels.

To enhance accessibility, we integrated a text-to-speech (TTS) feature using the pyttsx3 library. This TTS functionality announces the closest detected entity's label and its relative position to the player in real-time. We calculated the relative position by determining the angle between the player's coordinates and the detected object's coordinates. Positions were categorized into eight directions:

right, top right, top left, left, bottom left, bottom, and bottom right.

The video feed was captured and processed frame-by-frame using OpenCV. Each frame was passed to the YOLOv8 model, which identified objects and extracted bounding boxes and class labels. Detected objects were categorized into player and non-player entities. The coordinates of the bounding boxes were used to determine the centre points of the detected objects.

If a player was detected, the corresponding frame region containing the player was cropped and passed to a separate YOLOv8 model trained for classification. This classification model, designed for real-time application, determined the player's orientation. The relative positions of non-player objects were calculated based on their coordinates and the player's orientation. A function compared the player's coordinates with those of the objects, adjusting the coordinates to match the player's perspective and determining the direction of each detected object.

The TTS functionality ran in a separate thread to ensure smooth performance, announcing the closest detected entity's label and its relative position in real-time.

Two separate display windows were used: one to show the annotated video frames with detected objects and another to present a textual list of detected entities along with their distances and relative positions. The annotated video frames displayed bounding boxes around detected objects, while the textual list provided a detailed overview of the entities detected in each frame.

### *3.3.3 Overfitting Mitigation Strategies*

**Mitigation Techniques**

We implemented several mitigation techniques during the training of our YOLO model. These techniques included regularisation methods to penalise large weights, early stopping to halt training when performance improvements plateaued, and dropout to randomly omit neurons during training. Together, these strategies help the model generalise better, maintain high accuracy, and perform consistently across various scenarios.

**Regularization** We implemented L2 regularisation by setting the weight decay parameter to 0.0005. This penalises large weights during training, helping to prevent overfitting. By applying L2 regularisation, we ensure the model remains robust and performs well across different datasets and scenarios.

**Dropout** We employed the drop-out technique during training. Dropout is a regularisation method that involves randomly omitting a fraction of neurons in the neural network layer during each training iteration. Specifically, we set the dropout rate to a value that ensures a balance between underfitting and overfitting. This technique helps the model learn more robust features by preventing it from relying too heavily on any one neuron, thereby promoting generalisation and improving performance on unseen data.

## 3.4 Research Focus

### *3.4.1 Test Cases*

This research will preform various test cases. These test cases are to help answer the research questions and to improve the validity of the research.

**Test Case 1** Evaluating the model's performance using manual annotation.

**Dataset** We created 50 images, and 10 of them are validation images, the rest are used for training.

**Purpose** The purpose of this experiment is to have baseline results to compare to the automatic labelling in the following test case.

**Methodology** The methodology for this test case was to create custom mini-maps via our custom mini-map tool, move the labels that are automatically generated into a separate folder, and place manually labelled annotations instead alongside the images. Then the model metrics were evaluated on an unseen dataset of 4393 images.

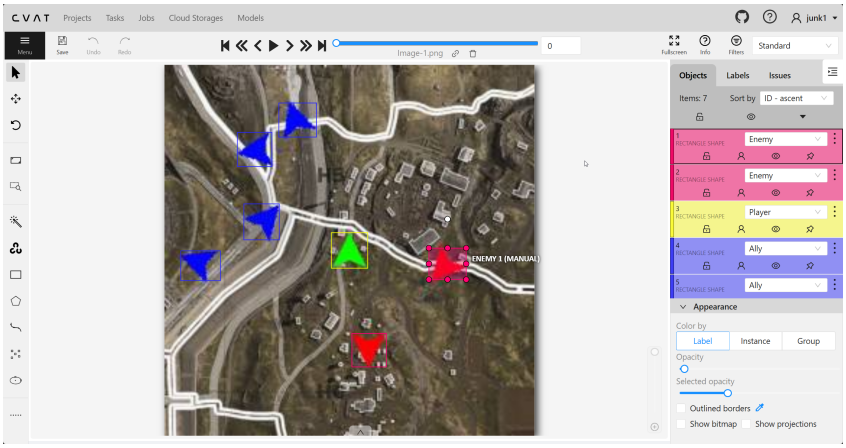


***Figure 3.5:*** *Test Case 1: Manual Annotation*

**Test Case 2** Evaluating the model's performance using automatic annotation.

**Dataset** We used the same dataset from the previous test case, but we replaced the annotation with automatic annotations generated by the custom mini-map generator.

**Purpose** The purpose of this experiment is to have baseline results to compare to the manual labelling in the prior test case.

**Test Case 3** Evaluating the model's performance on different mini-map different background.

**Dataset** We created another synthetic dataset with a complex background instead of a random colour background. This dataset featured the same mini-map icons, but only the background was different.

**Purpose** The purpose of this test case was to test the generalisability of the model to see if a model trained on a simple background can be used on a more

complex background.

**Test Case 4** Evaluating the model's performance on a smaller icon size to see if it is detectable.

**Dataset** We created another training dataset with smaller icons 12×12 pixels while originally 24×24. The mini-map icons will be on a complex background as well.

**Purpose** The purpose of this is to test if the size of the icons change, which sometimes can happen in real-world scenarios. Will the model still understand the objects, or does the model require instances of the occurs to be in the training dataset.

**Experiment 5** Evaluating the model's performance in a real-time scenario.

**Dataset** We recorded footage from "Call Of Duty: Modern Warfare" and then we extracted the frames that included the player and the allies so we can extract the icons and use them for our mini-map application.

**Purpose** The purpose of this test case is to verify if the application works in existing, and popular games.

TODO Update each methodology for each test case.

# Chapter 4: Analysis of Results and Discussion

## 4.1 Model Performance and Validation

### *4.1.1 Evaluation Metrics*

Several key metrics were used to evaluate the performance of our YOLO model, The metrics are Precision, Recall, and F1 Score. Precision shows how well the model avoids false positives. Recall indicates how well the model captures all relevant instances. The F1 Score provides a balanced evaluation by accounting for both false positives and false negatives.

**Precision**

To determine the precision, it requires dividing the number of TP by the sum of TP and false positives (FP), as given in the below formula. In our context, precision indicates how well the model can correctly detect objects on the mini-maps without mistakenly identifying irrelevant elements. High precision is generally important in applications where FP potentially might cause issues like mistakenly labelling items that are not related to what the model's aiming to identify.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

**Recall**

Recall measures the ability of a model to correctly identify all relevant objects within a dataset. It is calculated by dividing the number of TP by the sum of TP and FN, as shown in the below equation. In our context, recall evaluates how effectively the model can detect and classify every object in the mini-map. A high recall indicates that the model is capable of identifying most of the relevant objects, reducing the chance of missing detection. This is important in applications where overlooking a relevant object could have significant negative consequences, such as in our context, missing an important objective or overlooking an enemy on the mini-map.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

**F1 Score**

The F1 Score is a metric that combines both Precision and Recall to provide a balanced evaluation of a model's performance. It is calculated as the harmonic mean of Precision and Recall, which accounts for both false positives and false negatives, as shown in the following equation. In our context, the F1 Score evaluates how well the model performs overall in detecting and classifying objects on the mini-maps. A high F1 Score indicates that the model maintains a good balance between correctly identifying relevant objects and avoiding the misidentification of irrelevant elements. This balance is crucial in scenarios where both false positives and false negatives can lead to significant issues, ensuring that the

model performs reliably and consistently in various real-world conditions.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

**Mean Average Precision**

Mean Average Precision (mAP) is a metric that extends the concept of Average Precision (AP) by evaluating a model's performance across multiple object classes. It is calculated as the mean of the AP values for each class, providing a comprehensive assessment of the model's ability to detect and classify various objects. In our context, mAP evaluates how well the model performs overall in detecting and classifying different types of objects on the mini-maps. A high mAP score indicates that the model maintains a good balance between correctly identifying objects of different classes and avoiding the misidentification of irrelevant elements across all classes. There are two main types of mAP used in object detection:

mAP50: Mean average precision calculated at an intersection over union (IoU) threshold of 0.50. It's a measure of the model's accuracy considering only the "easy" detections, where the predicted bounding box and the ground truth bounding box overlap by at least 50%.

mAP50-95: The average of the mean average precision calculated at varying IoU thresholds, ranging from 0.50 to 0.95 with a step size of 0.05. It gives a comprehensive view of the model's performance across different levels of detection difficulty, including both "easy" and "hard" detections.

**Inference Speed**

Inference Speed is a metric that measures how quickly a model can process and analyse data to generate predictions. It is typically calculated as the time taken for various stages of the inference process, from pre-processing to post-processing. In our context, inference speed evaluates how quickly the model can detect and classify objects on the mini-maps, which is crucial for real-time applications. A high inference speed indicates that the model can process information rapidly, allowing for quick decision-making and responses in dynamic environments.

### *4.1.2 Object Detection Model Validation Performance*

**Metrics**

During the training of the model on a synthetic dataset comprising 6320 images, with 20% allocated as a validation set.

As shown in the figure below 4.1 The results from the training epochs show a clear trend of improvement over time. Initially, the model's errors were quite high, but by the fifth epoch, these errors significantly decreased. Precision and recall remained very high and stable, consistently above 98% after the first epoch. The mAP50 and mAP50-95 metrics, which measure the model's accuracy, also showed significant improvement. Validation losses for box, classification, and DFL also dropped, showing that the model was learning well and generalizing better.

The best performance was at epoch 269. At this point, the model achieved its highest accuracy, with precision and recall metrics nearly perfect at almost 100%.

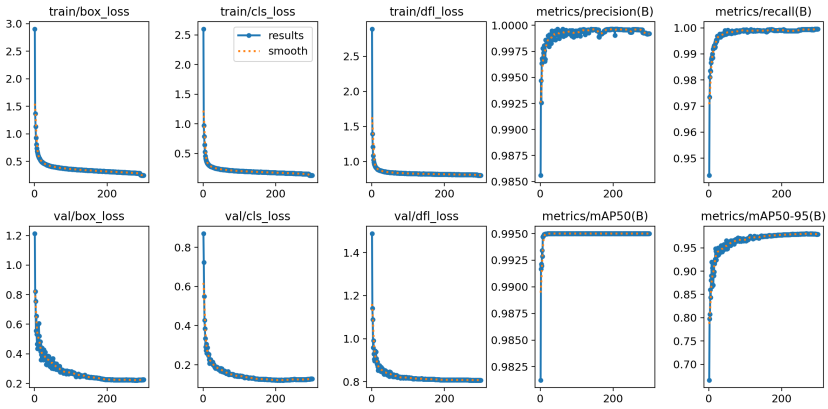


***Figure 4.1:*** *Model Results*

The validation losses were minimal, indicating the model's strong generalization capabilities and effective learning during training.

### *4.1.3 Object Detection — Training Dataset*

The evaluation of the model on the test dataset shows excellent performance across all categories. The model was tested with 3,234 images, resulting in the detection of 6,565 enemies, 6,534 allies, and 3,234 players. It accurately identifies enemies and allies 99.98% of the time and players 99.97% of the time. It correctly finds enemies 99.88% of the time, allies 99.89% of the time, and players 99.97% of the time. The combined measure of accuracy and completeness is also very high, close to 100% for all categories.

The model's overall accuracy, measured as the average precision at 50% overlap, is 99.5%. Additionally, its detailed accuracy across different levels of difficulty, measured as the average precision across 50% to 95% overlap, is 97.8%. The model works quickly, processing each image in just a few milliseconds. These results indicate that the model is good at detecting and classifying enemies, allies, and players quickly and accurately.

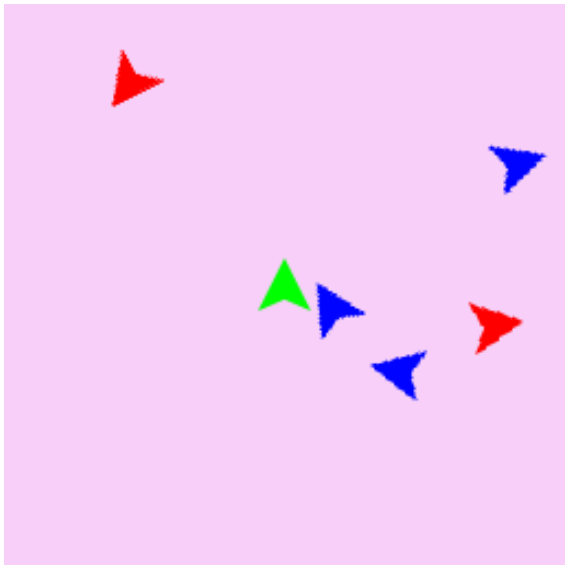

***Figure 4.2:*** *Testing Dataset Mini-map images*

TODO UPDATE TRAINING DATASET.

| Class | Images | Instances | Precision (%) | Recall (%) | mAP50 (%) | mAP50-95 (%) |
|---|---|---|---|---|---|---|
| Enemy | 3234 | 6565 | 100 | 99.9 | 99.5 | 97.2 |
| Ally | 3234 | 6534 | 100 | 99.9 | 99.5 | 96.7 |
| Player | 3234 | 3234 | 100 | 100 | 99.5 | 99.5 |

***Table 4.1:*** *Performance metrics for enemy, ally, and player detection.*

### *4.1.4 Classification Model — Metrics*

### *4.1.5 Testing Dataset*

### *4.1.6 Real-Time Prototype*

In the below figure 4.3 the prototype was able to detect all objects (enemies, allies, and players). The objects are being detected with high accuracy and speed approximately 45FPS per second. However, even though the model has high accuracy, there are some instances where the model fails to detect the objects as shown in the figure 4.4. The figure illustrates a hidden red icon (enemy) behind the blue icon (ally) right to the green icon (player).

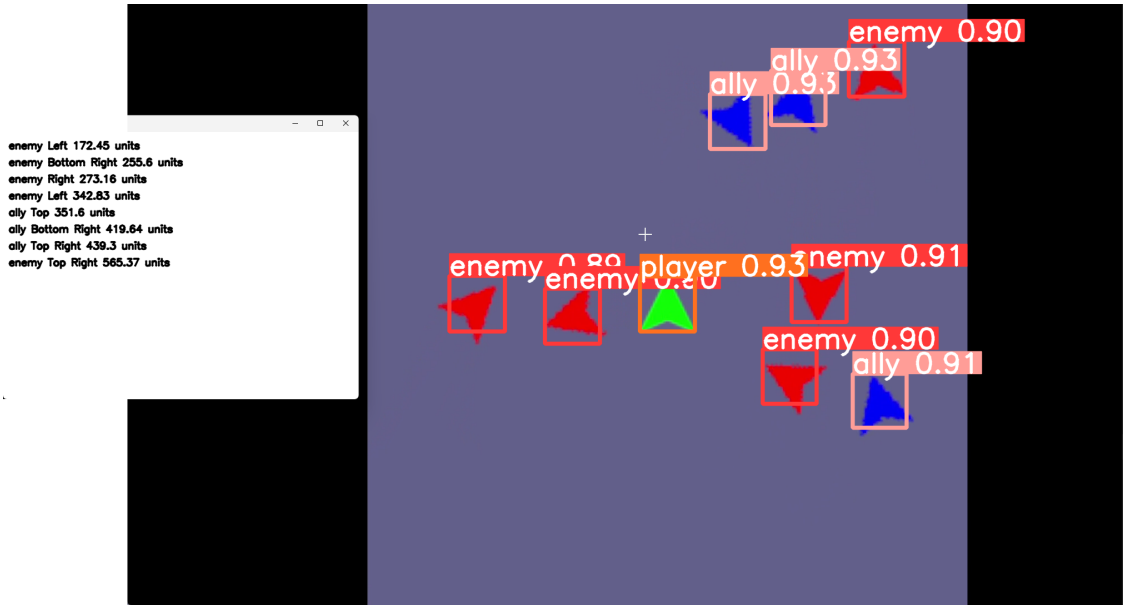


***Figure 4.3:*** *Real-time predictions*


***Figure 4.4:*** *Real-time prediction occlusion*

### *4.1.7 Test Cases*

**Test Case 1: Manual Annotation Test Case**

**Results**

In this the below Table 4.2, The model's showcases strong performance across all classes, indicated by its strong precision, recall, and F1 scores. These metrics suggest that the model can accurately identify objects of each class, with a low rate of false positives and false negatives. This is further reinforced by the below Figure **??** which illustrates all predictions preformed by the model compared to the ground truths which can be found in the below figure **??**.

| **Class** | **Precision (%)** | **Recall (%)** | **F1 Score (%)** |
|---|---|---|---|
| Enemy | 99.58 | 96.34 | 97.93 |
| Ally | 98.06 | 96.63 | 97.34 |
| Player | 95.95 | 92.58 | 94.23 |

***Table 4.2:*** *Performance metrics for each class in test case 1*


***Figure 4.5:*** *Test Case 1: Predictions*

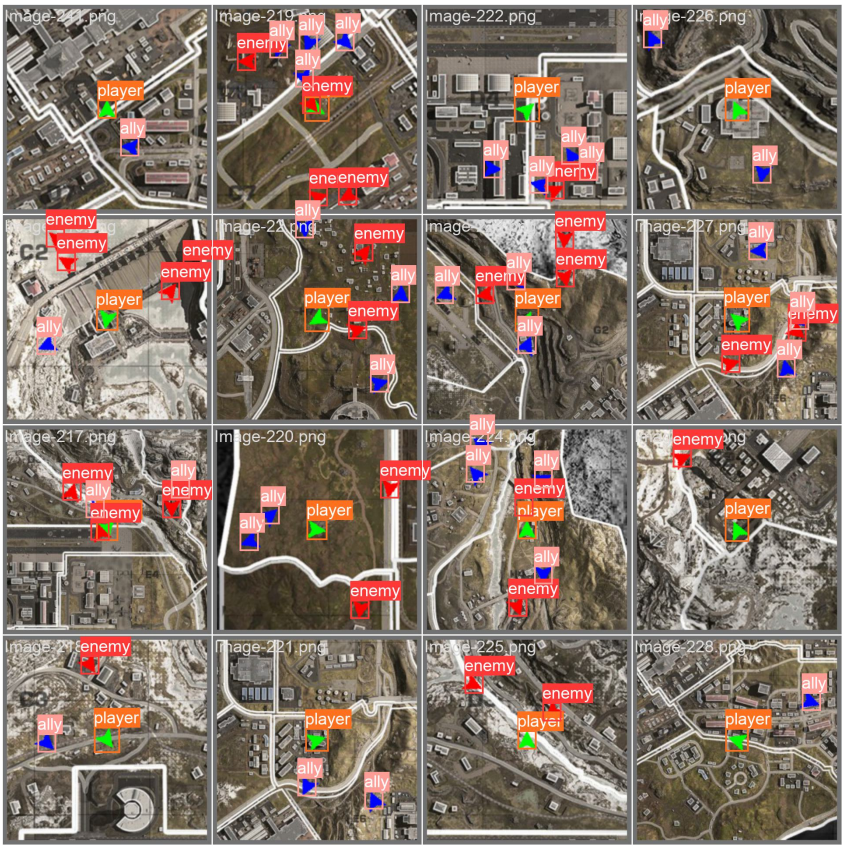

***Figure 4.6:*** *Test Case 1: Labels*

**Test Case 2: Automatic Annotation**

**Results**

The below table 4.3 presents the performance metrics for a model. The model demonstrates exceptional precision in identifying Ally instances, achieving a near-perfect value, and maintains high precision for Enemy and Player classes. In terms of recall, the model excels in detecting Enemy instances, with a high recall rate, and showcases a good ability to identify Ally and Player instances. The F1 scores, a balanced measure of precision and recall, further highlight the model's robust performance across all classes, with scores exceeding a high threshold for each class. However, when examining the predictions shown in the Figure4.7 we


***Figure 4.7:*** *Test Case 2: Predictions*

notice two things the overall confidence level is lower, and when the player is rotated to face a certain direction, the player is not being detected by the model.

| Class | Precision (%) | Recall (%) | F1 Score (%) |
|---|---|---|---|
| Enemy | 97.39 | 98.80 | 98.09 |
| Ally | 99.57 | 97.09 | 98.31 |
| Player | 96.66 | 90.30 | 93.37 |

***Table 4.3:*** *Test Case 2: Performance metrics for each class using automatic annotation.*

### Compression of Test Case 1 and Test Case 2

TODO Add this section

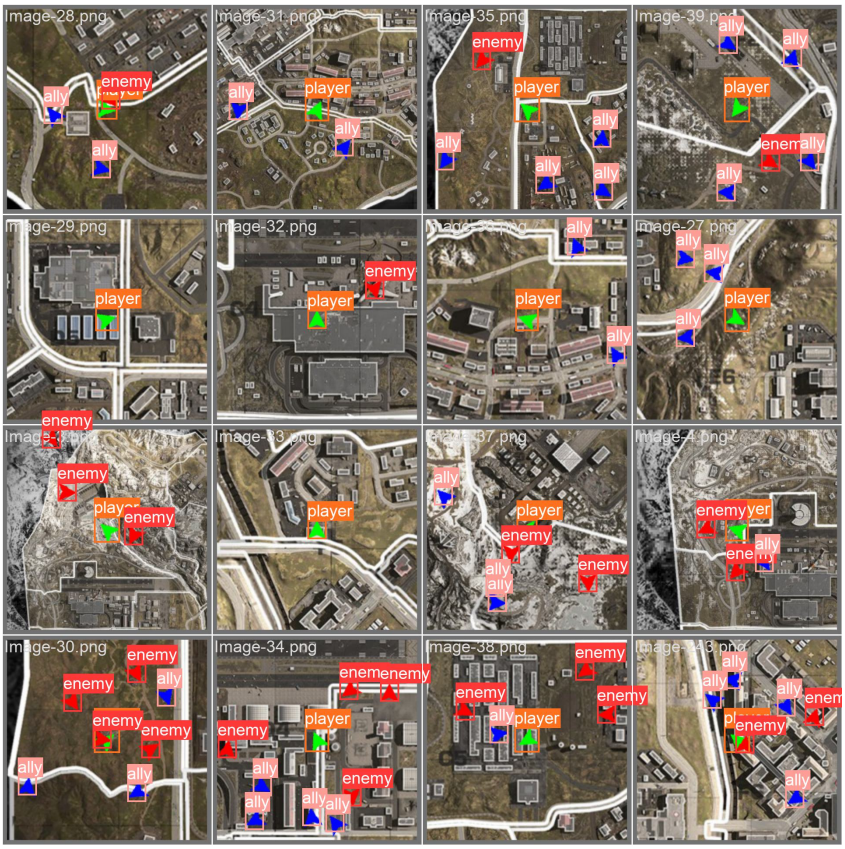


***Figure 4.8:*** *Test Case 2: Labels*

### Test Case 3: Different Mini-Map Backgrounds

### Results

In the below Table4.4, The model performed well for the enemy and ally classes, demonstrating its ability to correctly identify them. However, the performance for the player class suffered when evaluated on a more complex background than it was trained on. As seen in the below Figure 4.9 the model had a tendency to mistake the background as one of the classes compared to the ground truths shown in the below Figure 4.10

| Class | Precision (%) | Recall (%) | F1 Score (%) |
|---|---|---|---|
| Enemy | 99.27 | 99.59 | 99.43 |
| Ally | 98.83 | 99.79 | 99.31 |
| Player | 99.39 | 59.77 | 74.65 |

***Table 4.4:*** *Test Case 3: Performance metrics for each class in the background test case*

***Figure 4.9:*** *Test Case 3: Predictions*

***Figure 4.10:*** *Test Case 3: Validation Batch Labels*

### Test Case 4: Smaller Mini-Map Icon

### Results

The below table 4.5 shows the evaluated metrics for test case 4. Based on these metrics, the model exhibits strong performance is identifying allies demonstrated by high precision and F1, while it encountered challenges in identifying the player class with low recall and F1 scores. the model's performance in identifying enemy class is good. It achieved a good balance between precision and recall and F1 scores. However, as shown in Figures 4.11 and 4.12, the model misclassified the background as player, ally, or enemy classes, leading to false positives.

| Class | Precision (%) | Recall (%) | F1 Score (%) |
|---|---|---|---|
| Enemy | 69.73 | 65.28 | 67.43 |
| Ally | 91.39 | 65.91 | 76.59 |
| Player | 70.43 | 38.28 | 49.60 |

***Table 4.5:*** *Test Case 4: Performance metrics for each class with different icon sizes*


***Figure 4.11:*** *Test Case 4 : Validation Labels*


***Figure 4.12:*** *Test Case 4 : Predictions*

### Test Case 5: Real-World Scenario

TODO Add this section

**Results**


***Figure 4.13:*** *Test Case 5: Predictions*


***Figure 4.14:*** *Test Case 5: Double Detection*


***Figure 4.15:*** *Test Case 5: Edges*

### *4.1.8 Analysis of Results*

**Overall Performance Analysis**

Overall, the model's performance performed well, especially in the control environment, where it had high recall, precision, and F1 scores when we evaluated the model on a test dataset. Even when the model was trained on a simple background (random RGB colours) and then tested on both smaller icons (12×12) and complex backgrounds. The model was still able to detect the objects (enemies, allies, and players) and had high scores.

**Manual vs Automated Annotation** When we compared manual & automatic annotation, we noticed little to no difference when it came to the metrics.

**Weakness** When running the prototype as a real-time application, we noticed some issues. The main two issues were when the model recognised a single object as multiple, which has the potential to cause some confusion. Furthermore,

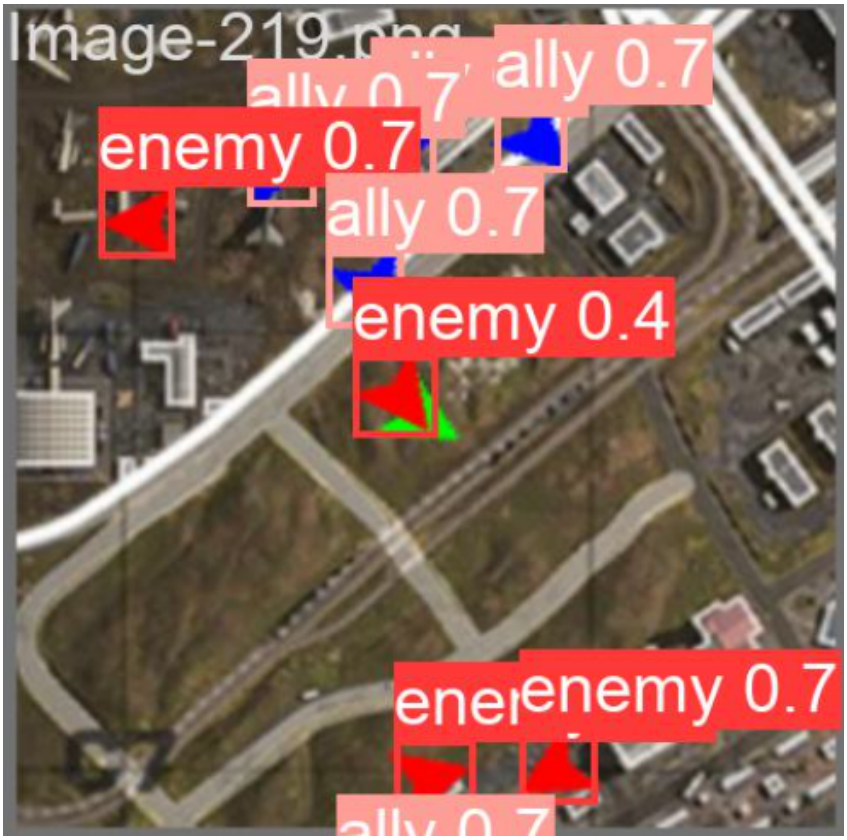


***Figure 4.16:*** *Enter Caption*

another common issue was that when the objects were clustered together, sometimes the object was occluded each other, and it would result in enemies not being picked up.

## 4.2 Comparison to other studies

TODO Add this section

## 4.3 Hypothesis & Research Questions

TODO Add this section

# Chapter 5: Conclusions and Recommendations

## 5.1 Limitations, Suggested Solutions and Enhancements

### *5.1.1 Limitations*

A limitation of the system as a whole was that the model required a lot of data for both classification & object detection models. We noticed this early on when we were manually gathering data. To streamline this, we implemented a synthetic dataset generation tool, and we achieved positive results with the tool; however, it is important to understand that this approach is dependent on the style of the game.

### *5.1.2 Suggested Solutions and Enhancements*

# List of References

[1] World Health Organization. *World Report on Vision*. World Health Organization, Geneva, 2019. Available under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 IGO licence (CC BY-NC-SA 3.0 IGO).

[2] Kevin Bierre, Jonathan Chetwynd, Barrie Ellis, D Michelle Hinn, Stephanie Ludi, and Thomas Westin. Game not over: Accessibility issues in video games. 2005.

[3] Batta Mahesh. Machine Learning Algorithms - A Review. 9(1).

[4] What Is Machine Learning (ML)? — IBM. = https://www.ibm.com/topics/machine-learning.

[5] What Is Unsupervised Learning? — IBM.

[6] Vratika Gupta, Vinay Kumar Mishra, Priyank Singhal, and Amit Kumar. An Overview of Supervised Machine Learning Algorithm. In *2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART)*, pages 87–92. IEEE, 2022.

[7] Prajakta Pawar, V Devendran, and Shivendra Singh. Deep learning based glance of real world scenes through decision tree. In *Proceedings of the Third International Conference on Advanced Informatics for Computing Research*, pages 1–9. ACM, 2019.

[8] Vladimir Nasteski. An overview of the supervised machine learning methods. 4:51–62.

[9] What Is Semi-Supervised Learning? — IBM. = https://www.ibm.com/topics/semi-supervised-learning.

[10] Victor Wiley and Thomas Lucas. Computer Vision and Image Processing: A Paper Review. *International Journal of Artificial Intelligence Research*, 2(1):22.

[11] Mrs. Arjoo Pandey. Computer Vision. 11(7):510–514.

[12] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object Detection with Deep Learning: A Review.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

[14] Meiyan Li and Joey S. Aviles. Research and Improvement of Object Detection Algorithm based on Regression Method. In *2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pages 198–201. IEEE.

[15] Swarnendu Ghosh, Nibaran Das, Ishita Das, and Ujjwal Maulik. Understanding Deep Learning Techniques for Image Segmentation.

[16] Shervin Minaee, Yuri Y. Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image Segmentation Using Deep Learning: A Survey. pages 1–1.

[17] Retracted: Deep Neural Networks for Medical Image Segmentation. 2023:1–1.

[18] Diya Chudasama, Tanvi Patel, Shubham Joshi, and Ghanshyam I. Prajapati. Image Segmentation using Morphological Operations. 117(18):16–19.

[19] Research on Computer Vision Image Classification. *Academic Journal of Computing & Information Science*, 6(6), 2023.

[20] Sonia Setia, Akshat Shukla, Amartya Raj, and Abhimanyu Rathore. A Detailed Review on Object Detection Algorithms. In *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 1521–1528. IEEE, 2023.

[21] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. 9905:21–37.

[22] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection.

[23] Lu Tan, Tianran Huangfu, Liyao Wu, and Wenying Chen. Comparison of YOLO v3, Faster R-CNN, and SSD for Real-Time Pill Identification.

[24] Min Li, Zhijie Zhang, Liping Lei, Xiaofan Wang, and Xudong Guo. Agricultural Greenhouses Detection in High-Resolution Satellite Images Based on Convolutional Neural Networks: Comparison of Faster R-CNN, YOLO v3 and SSD. *Sensors*, 20(17):4938, 2020.

[25] Ronny Andrade, Melissa J. Rogerson, Jenny Waycott, Steven Baker, and Frank Vetere. Playing Blind: Revealing the World of Gamers with Visual Impairment. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–14, Glasgow Scotland Uk, May 2019. ACM.

[26] Michael A. Nees and Eliana Liebman. Auditory Icons, Earcons, Spearcons, and Speech: A Systematic Review and Meta-Analysis of Brief Audio Alerts in Human-Machine Interfaces. *Auditory Perception & Cognition*, 6(3-4):300–329, October 2023.

[27] Vishnu Nair, Jay L. Karp, Samuel Silverman, Mohar Kalra, Hollis Lehv, Faizan Jamil, and Brian A. Smith. NavStick: Making Video Games Blind-Accessible via the Ability to Look Around. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 538–551, October 2021. arXiv:2109.01202 [cs].

[28] Vishnu Nair, Shao-en Ma, Ricardo E. Gonzalez Penuela, Yicheng He, Karen Lin, Mason Hayes, Hannah Huddleston, Matthew Donnelly, and Brian A. Smith. Uncovering Visually Impaired Gamers' Preferences for Spatial Awareness Tools Within Video Games. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 1–16, October 2022. arXiv:2208.14573 [cs].

[29] Shari Trewin, Vicki L. Hanson, Mark R. Laff, and Anna Cavender. PowerUp: an accessible virtual world. In *Proceedings of the 10th international*

*ACM SIGACCESS conference on Computers and accessibility*, pages 177–184, Halifax Nova Scotia Canada, October 2008. ACM.

[30] T Westin. Game accessibility case study: Terraformers – a real-time 3D graphic game. *Virtual Reality*, 2004.

[31] Margherita Antona and Constantine Stephanidis, editors. *Universal Access in Human-Computer Interaction. Design Methods and User Experience: 15th International Conference, UAHCI 2021, Held as Part of the 23rd HCI International Conference, HCII 2021, Virtual Event, July 24–29, 2021, Proceedings, Part I*, volume 12768 of *Lecture Notes in Computer Science*. Springer International Publishing, Cham, 2021.

[32] The last of us 2 - blind accessibility review, 2020. Accessed: 2024-05-13.

[33] Ulrich Schiefer, Christina Kraus, Peter Baumbach, Judith Ungewiß, and Ralf Michels. Refractive errors. *International Journal of Health Sciences*, 2016.

[34] Bei Yuan, Eelke Folmer, and Frederick C. Harris. Game accessibility: a survey. *Universal Access in the Information Society*, 10(1):81–100, March 2011.

[35] Aaron Gluck, Kwajo Boateng, and Julian Brinkley. Racing in the Dark: Exploring Accessible Virtual Reality by Developing a Racing Game for People who are Blind. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 65(1):1114–1118, September 2021.

[36] David Gonçalves, André Rodrigues, and Tiago Guerreiro. Playing With Others: Depicting Multiplayer Gaming Experiences of People With Visual Im-

pairments. In *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility*, pages 1–12, Virtual Event Greece, October 2020. ACM.

[37] Masaki Matsuo, Takahiro Miura, Masatsugu Sakajiri, Junji Onishi, and Tsukasa Ono. Inclusive Side-Scrolling Action Game Securing Accessibility for Visually Impaired People. In Regina Bernhaupt, Girish Dalvi, Anirudha Joshi, Devanuj K. Balkrishan, Jacki O'Neill, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2017*, volume 10516, pages 410–414. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[38] David Gonçalves, Manuel Piçarra, Pedro Pais, João Guerreiro, and André Rodrigues. "My Zelda Cane": Strategies Used by Blind Players to Play Visual-Centric Digital Games. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–15, Hamburg Germany, April 2023. ACM.

[39] John W. Creswell and J. David Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, Thousand Oaks, California, 5 edition, 2018.

[40] Pygame. `https://github.com/pygame/pygame`.

[41] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8. `https://github.com/ultralytics/ultralytics`, 2023.

[42] Natesh Bhat. pyttsx3. `https://github.com/nateshmbhat/pyttsx3`.

[43] S. Nikolenko. *Synthetic Data for Deep Learning*. 2019.