University of Southern Queensland
Faculty of Engineering and Surveying

Real-Time UAS Detection in High-Security Environments

A dissertation submitted by

Luke Webber

in fulfilment of the requirements of

ENP4111 Professional Engineer Research Project

towards the degree of

Bachelor of Engineering (Honours) (Electrical and Electronics)

Submitted December, 2024

Word Count: 17,213
Reference Count: 55
Page Count: 77

This page intentionally left blank

University of Southern Queensland

School of Engineering

ENP4111 Dissertation Project

(This is a 2-unit research project in Bachelor of Engineering Honours Program)

Limitations of Use

CERTIFICATION

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Student Name: Luke Webber

Student Number: [redacted]

Luke Webber

Signature

06/12/2024

Date

ABSTRACT

Keywords: *UAV, UAS, RPAS, Drone, Object Detection, Machine Learning, Neural Network, Prison, Security, Optuna, YOLO*

This thesis investigates the development and evaluation of a real-time drone detection system designed for high-security environments. The core objective was to ensure reliable detection of unmanned aerial systems (UAS), with a target detection accuracy of at least 50%. To achieve this, the YOLO11 object detection framework was implemented and evaluated against its variants, YOLO11m, YOLO11l, and YOLO11x, while considering trade-offs between accuracy and computational efficiency. Model training and hyperparameter optimisation were performed on a Google Cloud virtual machine to leverage its computational capabilities.

The optimised YOLO11 model delivered a mean average precision (mAP) of 0.469, a peak F1 score of 0.45, and a precision of 0.735 for the Drone class. These results reflect a considerable improvement over the baseline YOLO11 models. However, certain limitations were observed, particularly difficulties in detecting larger drones and classifying birds due to dataset constraints. Although the model effectively detected UAS across all five real-time test datasets, challenges such as inconsistent tracking in video-based tests suggest areas for refinement. Improvements could include dataset augmentation to address drone size diversity and the integration of advanced tracking algorithms.

The research concludes the optimised YOLO11 model offers a promising, efficient solution for UAS detection, particularly in high-security contexts like prison perimeter surveillance. Future efforts should prioritise building dataset diversity, stabilising tracking performance, and optimising the model for secure environments. This will ensure the model's readiness for real-world applications, where precision and reliability in drone detection are vital in maintaining security.

ACKNOWLEDGMENTS

I extend my deepest gratitude to the teaching staff at UniSQ, whose guidance and support have been invaluable in helping me complete this dissertation during an exceptionally challenging year.

This thesis would not have been possible without the unwavering support of my family. To my girlfriend, Kylie, thank you for your endless patience, love, and encouragement, especially during the countless hours I have spent wrestling with code late into the night.

Finally, I dedicate this work to the memory of my late father, Stephen Webber. From the very beginning of this degree he was my greatest supporter, and while he could not witness the completion of this capstone project, I hope he rests in peace knowing that his greatest wish of seeing me graduate is a giant leap closer to being fulfilled.

Contents

List of Figures

List of Tables

GLOSSARY

AI	Artificial Intelligence
BSM	Background Subtraction Method
CNN	Convolutional Neural Network
FN	False Negative
FP	False Positive
HPO	Hyperparameter Optimisation
IoU	Intersection of Union
mAP	Mean Average Precision
mAP50	Mean Average Precision at a 0.5 confidence level
NCC	Normalised Cross-Correlation
PR	Precision-Recall
PTZ	Pan, Tilt & Zoom (camera)
RCNN	Recurrent Convolutional Neural Network
RNN	Recurrent Neural Network
SSD	Single Shot Detector
TP	True Positive
UAS	Unmanned Aerial System
YOLO	You Only Look Once

INTRODUCTION

Aim

The aim of this research is to develop a low-cost UAS detection system that can be deployed in high-security environments. The project will investigate available detection methods, evaluating their feasibility and practicality to identify the most suitable approach for implementation, with intention of developing a scalable solution tailored for prison environments.

Background

The rise in the use of UAS, commonly known as drones, has greatly benefited the agriculture, surveillance, and logistics industries. However, the widespread availability of drones has posed unique security risks, especially in correctional facilities where controlled environments are critical for safety. UAS have been linked to illicit activities such as smuggling contraband, conducting unauthorised surveillance, and facilitating escape attempts.

In Australia, there has been an increase in incidents involving UAS attempting to breach prison security. For example, in 2020, a UAS was used to deliver prohibited items, including a knife and drugs, into a maximum-security prison (Vedelago, 2020). The same year, prison authorities were forced to relocate a high-profile inmate due to safety concerns after a UAS was suspected of gathering intelligence on the facility's layout (Usher, 2020). Criminal networks have adapted by leveraging cheap and available UAS technology to outmaneuver traditional security measures, as demonstrated by these incidents.

To address the growing threat of unauthorized UAS activity, the Civil Aviation Safety Authority (CASA) installed Passive UAS detection systems at 29 Australian airports to enhance airspace protection and secure critical infrastructure (Purtill, 2023). However, prisons present unique challenges for detection systems, as their dense infrastructure and government budgets make it difficult to deploy the same level of technology used at airports.

In May 2023, New South Wales prison officers seised over AUD$500,000 worth of contraband in one of the largest hauls in the state's history (NSW Government, 2023). While it was not confirmed that UAS played a role, the 'ingenuity' of the smuggling methods suggests the possible involvement of UAS technology. The availability of Commercial-Off-The-Shelf (COTS) UAS has amplified security risks. Modern COTS UAS offer increased payload capacities, extended flight durations, and ease of use at a fraction of the cost of older technologies. This has made them a favored tool for smuggling contraband such as drugs, weapons, and other illicit items into prison grounds (Delleji et al., 2023). As UAS technology becomes cheaper and more advanced, the need for accessible and affordable counter-UAS measures is increasing rapidly.

Research into UAS detection technologies has produced several promising solutions, but many remain cost-prohibitive or too complex for practical use in prisons. Detection

systems often relies on a combination of sensors, such as radar, thermal imaging, and optical systems, to track UAS in diverse environmental conditions. While effective, these systems require significant financial investment and technical expertise, which can be limiting for correctional facilities (Brewczyński et al., 2024). Emerging approaches, such as neural network-based detection models like the Deep Sky Monitoring System, have shown improved detection capabilities. However, their reliance on specialised hardware and software also increases costs (Delleji et al., 2023).

The threat posed by UASs is not confined to Australia, with international reports highlighting similar incidents in correctional facilities worldwide. In the United States, the Department of Justice has documented a rise in UAS-related incidents at federal prisons. Reports indicate that many such incidents go unreported due to a lack of sufficient detection capabilities and limited staff awareness (Office of the Inspector General, U.S. Department of Justice, 2020). The growing number of incidents illustrates the global nature of the problem and the pressing need for cost-effective, scalable solutions.

Mitigating the threat of UASs calls for improved strategies that include early detection, rapid response, and real-time tracking solutions. Integrated multi-sensor systems that combine radar, optical, and thermal imaging technologies have proven effective at improving detection accuracy and reducing false positives (Brewczyński et al., 2024). However, affordability remains a significant hurdle. Advanced systems like the Deep Sky Monitoring System leverage neural network models, including adaptations of YOLO-based frameworks, to enhance detection and tracking capabilities (Delleji et al., 2023). Additionally, the development of power-efficient systems, such as LoRa mesh networks, holds potential for cost-effective and energy-efficient UAS detection, making them more practical for correctional facilities (Kim et al., 2023).

The frequency of UAS-related incidents in prisons and the limitations of existing detection methods point to the need for innovative, affordable, and effective counter-UAS solutions. Correctional facilities must use a targeted approach that integrates advances in detection technology with cost constraints in mind. The adoption of sensor integration, machine learning, and energy-efficient networks, prison authorities can strengthen security and prevent UAS from undermining the integrity of correctional systems.

Challenges

This research project aims to address critical challenges rooted in both theoretical and practical aspects, particularly related to the security risks posed by UAS activities in and around prison environments.

Theoretical Challenges:

1. Assessing Current Systems: Conducting a thorough evaluation of existing UAS detection technologies to determine their effectiveness and suitability in correctional facilities. This includes identifying existing limitations, vulnerabilities, and potential areas for improvement.

2. Enhancing Cost Efficiency: Investigating methods to improve the performance and

reliability of UAS detection systems while minimising costs, ensuring that security enhancements remain financially feasible for implementation.

3. Ensuring Integration: Evaluating the compatibility of UAS detection systems with current prison security infrastructure to enable smooth integration and operational effectiveness without significant disruptions.

The primary practical challenge this research will address is the creation of a real-time UAS detection system capable of integrating with existing prison security frameworks. The research will evaluate the proposed UAS detection system in terms of detection accuracy and integration feasibility. Detection accuracy will measure the system's ability to reliably identify UASs under varying environmental conditions. The ultimate goal is to develop a reliable real-time UAS detection system tailored to the unique requirements of correctional facilities.

Objectives

Objective 1: **Comprehensive Research and Analysis** Conduct an in-depth investigation into the challenges posed by UAS activities in correctional environments. Evaluate existing detection solutions in terms of their effectiveness, integration feasibility, and compliance with legal and ethical frameworks.

Objective 2: **Identification of Optimal UAS Detection Technology** Determine the most suitable UAS detection approach (e.g., visual detection using machine learning, acoustic detection) for this application. The selection will prioritise criteria such as accuracy, reliability, cost-effectiveness, and compatibility with the operational demands of correctional facilities.

Objective 3: **Performance Evaluation of Selected Technology** Assess the chosen UAS detection technology's performance, emphasising its ability to detect UASs with high accuracy and speed across diverse environmental conditions and UAS types.

Objective 4: **Development of a Cost-Effective Prototype** Create a cost-effective prototype to demonstrate the viability of the selected technology.

Objective 5: **Deployment in a Real-World Scenario** Integrate into an existing system. This was, unfortunately, not able to be achieved in the timeframe.

LITERATURE REVIEW

Current Prison Infrastructure

Building an effective UAS detection system requires a robust assessment of the existing infrastructure within prisons and identifying points where new technology can be integrated. The shift towards 'smart prisons' has increased in recent years, particularly in response to challenges highlighted during the COVID-19 pandemic. Smart prisons are modernised facilities that use advanced technology like artificial intelligence (AI) to improve surveillance and security while also supporting the wellbeing of staff and inmates. AI-driven surveillance systems identify inmate altercations, detect unauthorized movements, and close security blind spots using biometric access controls, facial recognition systems, thermal imaging, and RFID tracking (McKay, 2022).

In Malaysia, prisons have adopted AI systems capable of analysing CCTV footage to detect physical conflicts or escape attempts (Kaun, 2020). Similarly, U.S. prisons utilise AI to monitor telephone conversations, helping identify organised crime and criminal networks within correctional facilities. In the UK, the Avigilon end-to-end security system offers behavioural and operational insights through live video monitoring; however, it does not currently support UAS detection and requires proprietary camera systems or additional software for better functionality (Goedbled, 2019). This limitation shows the potential financial burdens associated with retrofitting existing infrastructure to accommodate new detection capabilities, especially when such upgrades necessitate replacing or supplementing current systems.

AI-based technology integration extends beyond physical prison infrastructure; research conducted by Swinburne University in Melbourne explores the use of AI for home detention, employing sensors and cameras monitored by machine learning algorithms to automatically alert authorities of violations (Bagaric, 2018). These technologies aim to alleviate the financial burden associated with incarceration, a pressing issue with the rising costs of prison operation. In the 2022-2023 financial year, Australia spent over $6 billion on the construction and operation of correctional facilities, equating to approximately $147,900 per prisoner per year (Schlicht, 2023). This figure starkly contrasts the 2021 median Australian income of $54,890, highlighting the economic challenges faced by correctional institutions and reinforcing the need for cost-effective solutions such as AI and machine learning.

New prison developments prioritize security modernization by incorporating smart surveillance, automated locking mechanisms, and AI-driven inmate behavior tracking. The Southern Queensland Correctional Precinct Stage 2 (SQCPS2), the newest correctional facility under construction in Queensland, represents a significant investment of $861 million (Infrastructure Partnerships Australia, 2023). Designed to alleviate overcrowding in Queensland's correctional system, the facility incorporates the SAAB OneView interface, a comprehensive security solution that has been deployed in 20 correctional facilities since its initial implementation at Stage 1 of the project (SQCPS1) in 2012. The OneView system integrates CCTV, intercoms, access control, and perimeter detection systems, enhancing situational awareness for prison officers (SAAB Australia, 2022).

The OneView system is operated from a central command and control centre, where alarms and integrated responses are triggered when sensors detect irregularities. This includes automatic activation of pan-tilt-zoom (PTZ) cameras to track detected intrusions. Notably, SAAB's promotional material highlights the system's customisability and ability to integrate additional security subsystems, such as biometric scanners and mobile duress alarms (SAAB Australia, 2021). Despite these capabilities, UAS detection is not currently included within the OneView system, presenting an opportunity for innovation. Given its widespread deployment and adaptability, OneView provides a strong platform for integrating a UAS detection system with minimal disruption to existing infrastructure, addressing both operational needs and cost concerns.

Research by Imandeka et al. (2024) further supports the potential of smart prison infrastructure by highlighting the role of Internet of Things architectures in correctional facilities. These systems rely on layers of physical devices (e.g., sensors, biometric tools), data networks, analytics, and integrated applications to enhance connectivity and enable real-time processing. While such systems improve operational efficiency and security, they also emphasise significant challenges, including the cost of retrofitting older facilities and the need for specialised training. In Australia, these financial and operational constraints highlight the importance of leveraging adaptable, modular systems like SAAB OneView to integrate new functionalities, such as UAS detection, without requiring extensive overhauls.

Examining current prison infrastructure reveals significant potential for incorporating UAS detection systems into existing frameworks. The scalability and modularity of these systems offer a flexible approach to addressing rising unauthorised UAS activity. This integration can be achieved while remaining mindful of the financial and operational constraints that correctional facilities often face, ensuring a practical and cost-effective solution.

Methods of Detection

The detection of UASs is becoming an increasingly necessary area of research, with a variety of methods being developed and tested to overcome the unique challenges posed by these systems. The primary goal of detection methods is to accurately identify and track UASs in real-time, drawing on advancements in sensor technology, machine learning, and signal analysis. Current literature shows three predominant detection categories; acoustic detection, radio frequency (RF) detection, and vision-based detection, each offering distinct advantages and practical applications.

Acoustic detection systems use directional microphones to identify UASs by analysing the distinct sounds from their motors and propellers. Acoustic detection is effective in forested or urban areas where visual line-of-sight is blocked, though its accuracy decreases in noisy environments such as construction zones or busy streets. RF detection, by contrast, monitors signals between UASs and controllers, using either Passive interception or Active radar for identification. This method is suited to identifying UASs operating on known RF bands but may be limited by encrypted or frequency-hopping signals. Vision-based detection generally uses machine learning models, particularly Convolutional Neural Networks (CNNs), to detect UASs within image frames captured by optical or infrared

cameras. This method excels in scenarios where visual identification is critical but can face challenges with low visibility or occlusions.

Each of these detection methods has their own strengths and limitations, often shaped by environmental conditions, operational demands, and cost considerations. The following sections will provide a detailed examination of these three detection approaches, exploring their underlying mechanisms, relative effectiveness, and suitability for implementation in high-security environments, such as correctional facilities.

Acoustic Detection

Acoustic detection methods have been extensively researched as a way of identifying UAS based on the sound signatures produced by their motors and propellers. In a study evaluating various sensors and data acquisition systems, including field-deployable microphone phased arrays and spherical arrays, Ramos-Romero (2023) found that ambient noise levels exceeding 35 dB and weather condition significantly reduced detection accuracy. Experiments with UASs flying at 15 m/s at a 10-meter altitude revealed that speed, payload, and drone model significantly influenced frequency spectrum variability, highlighting the challenges in standardising acoustic detection.

Sudenov et al. (2019) developed a passive acoustic detection system, using a range of consumer-grade quadcopters and fixed-wing UAS to measure noise characteristics and apply detection algorithms. Their system achieved a maximum detection range of 250 meters using a cross-correlation method, showing the potential of acoustic analysis but also revealing its dependence on optimal conditions. Similarly, Lacava et al. (2023) explored the integration of deep learning techniques with high-performance microphone arrays, utilising CNNs, Recurrent Neural Networks (RNNs), and Recurrent Convolutional Neural Networks (RCNNs). These methods improved detection accuracy but relied on large, annotated datasets, which are resource-intensive to create, and faced delays from computationally complex machine-learning models.

Kyritsis et al. (2022) addressed acoustic detection challenges by estimating Direction of Arrival using spectrogram-based machine learning, achieving 70-meter detection ranges under ideal condition, but struggled with environmental noise and attenuation. Similarly, Paszkowski et al. (2024) utilised a neural network trained on spectrograms to detect UASs in urban environments, achieving high accuracy in controlled settings but noting significant limitations in real-world applications due to interference from ambient noise.

In another study, Aydin and Kizilay (2022) developed a lightweight CNN model that utilised Mel frequency spectrum-based pre-processing to classify UAS sounds effectively even in noisy environments. However, the model's success depended on high-quality audio datasets and faced range limitations similar to other acoustic methods. The study also emphasised the computational challenges posed by real-time processing, particularly in environments with high levels of background noise.

These studies confirm that acoustic analysis is effective for UAS detection at short ranges, but also showed significant limitations including limited range, susceptibility to environmental noise, and the computational demands of real-time processing. Correctional facilities, characterised by high ambient noise levels, expansive buildings,

and the need for large-scale microphone arrays, present additional challenges that make acoustic detection unsuitable for this application. The integration of such systems into existing prison infrastructure would require extensive retrofitting and significant financial investment, further limiting their feasibility.

Based on the above research, acoustic detection was deemed inappropriate for the correctional environment and was not explored further.

RF Detection

Radio Frequency (RF) detection is widely used for UAS detection, as drones depend on RF signals for Command and Control communication. RF detection are divided into two types: Active and Passive systems. Active RF detection transmits radar signals and analyses their reflections to generate a radar cross-section, identifying UAS features such as body shape, propeller movement, or vibrations. Passive RF detection intercepts communication signals between UASs and controllers, analyzing protocols, frequency usage, and signal strength for identification.

RF detection is widely deployed in high-security settings, such as military bases and airports, where it reliably identifies and tracks UAS activity. Active RF systems offer extended range and precision, making them particularly useful for detecting stealth UASs or those operating without controller signals. However, these systems consume significant power and require large-scale radar installations, limiting their use in smaller facilities. In contrast, Passive RF systems are cost-effective and energy-efficient but struggle to differentiate unauthorized signals in RF-dense areas, such as urban correctional facilities.

Both active and passive RF methods provide viable solutions for correctional facilities, addressing UAS intrusions despite challenges in power consumption and signal interference. The following sections will explore these methods in greater detail, examining their mechanisms, advantages, and limitations.

Active RF

Active RF detection, commonly referred to as radar, offers the advantage of tracking UAS that do not emit signals, such as autonomous UASs. Deng et al. (2023) developed 'Dr. Defender,' a system based on radar's ability to detect signal-less UASs. Their study identified a key limitation of traditional radar: difficulty discriminate between objects, which often results in false positives. Deng et al. explored UAS detection using Wi-Fi signals from home networks, using Spinning Frequency Extraction as a distinguishing feature. The study found anomaly detection more effective with multiple UASs, indicating that overlapping activity can improve detection accuracy in dense environments.

Market solutions like the Aerial Armor GroundAware system reportedly tracks UASs up to 3 miles using pulsating Doppler radar, though its accuracy and adaptability in real-world scenarios have not been extensively tested or reported. Flórez et al. (2020) identified key limitations of radar systems, including reduced reflections from carbon

fiber and plastic propellers, common in lightweight UAS. Their study showed that mmWave radar systems using MIMO techniques were effective in tracking multiple UASs within ranges of 50–150 meters, suggesting their potential for deployment in dynamic environments.

Poitevin (2017) investigated the detection of micro-UAS (weighing less than 2 kg) using military-grade forward-looking infrared (FLIR) radar systems. While FLIR systems detected micro-UASs up to 700 meters, their high cost and narrow field of view limit practical applications. The study also noted that limited directional coverage restricts the use of FLIR systems in settings like correctional facilities, where 360-degree surveillance is essential.

Zhou et al. (2023) developed networked radar systems that use interconnected nodes to enhance spatial resolution and detection accuracy, offering a significant improvement over traditional radar. Unlike traditional mono-static or multi-static radar systems, networked radar systems rely on integration between radar nodes, reducing blind spots and providing comprehensive coverage of monitored zones. The study demonstrated the adaptability and resilience of these systems, which can dynamically reconfigure nodes to maintain functionality even when parts of the system fail. Although effective in open environments, networked radar systems are less suitable for correctional facilities due to high costs and structural signal obstructions. Installing multiple radar nodes and the necessary infrastructure for data fusion and communication requires significant financial investment. Radar signals in correctional facilities are often obstructed by high walls, complex layouts, and dense vegetation, reducing system performance.

Zhang et al. (2022) developed an intrusion detection method that combines Wavelet Leader Multifractal Analysis (WLM) with Long Short-Term Memory (LSTM) networks, achieving high accuracy in controlled tests. This approach achieved high accuracy in controlled environments, successfully identifying spoofing attacks in UAS and radar systems. However, several limitations were noted:

- Computational Complexity: The WLM methodology demands significant processing resources, particularly for real-time feature extraction and classification, which may limit its deployment on standard hardware.
- False Positives: Despite achieving an overall accuracy of 90%, the system reported false positive rates of 7.9%, triggering unnecessary alerts and interventions.
- Data Dependence: The LSTM model's performance depended on high-quality training data, with insufficient or imbalanced datasets reducing detection reliability.
- Environmental Constraints: The methodology performed well in controlled scenarios, but its effectiveness in noisy and complex environments like correctional facilities remains uncertain.
- Cost and Scalability: The need for specialised hardware and high computational power makes WLM systems costly and challenging to scale in budget-limited deployments.

While Active RF detection systems are effective in UAS detection and tracking, their deployment in correctional facilities presents significant challenges. Advanced radar

systems are costly, and prisons' dense structures complicate signal reception, making these detection systems less practical for correctional use. Machine learning models require high-quality datasets and controlled testing conditions, which are difficult to replicate in the variable environment of correctional facilities. Consequently, while Active RF detection works well in controlled environments, it is poorly suited for correctional facilities due to cost, complexity, and environmental constraints.

Passive RF

Passive RF is a method of monitoring adio frequencies, known as spectral surveillance, to identify UAS activity. UAS detection methods generally targets communication in the 2.4GHz and 5GHz frequency ranges between UAS and controller, the ISM standard used in most UAS technology. Whilst these are the primary frequencies, some drones operate outside this range, reducing the effectiveness of frequency-specific detection methods.

Lv et al. (2021) found that monitoring a wide range of signals from 1MHz to 6.8GHz gave frequent false alarms as many of these signals were unrelated to UAS activity. Their analysis showed that a typical UAS had three separate signals, map transmission, remote and WiFi signals, operating in the ISM spectrum. Only the UAS map transmission signal had a significant feature with a bandwidth of 9MHz to allow it to be detected amongst the noise of the ISM band. If other frequencies are operating with a bandwidth of 9MHz, this may increase false detections. The method struggles in environments with complex electromagnetic interference, limiting its ability to filter out all non-UAS signals.

Aouladhadj et al. (2023) tested several detection methods, including algorithm gradient boosting, goodness-of-fit sensing, deep recurrent neural networks, and wavelet transform analytics achieving a 98.9% success rate in controlled environments, but ultimately found that these methods required a database of all potential RF communications to learn from, and used significant memory resources. Their findings indicate that this method only works within the Wi-Fi-based ISM band and is incompatible with DJI UAS models using OcuSync technology. With Wi-Fi bandwidths becoming crowded, UAS manufacturers are exploring frequencies beyond the ISM range to improve signal range and network stability, which requires existing detection technologies to meet new standards.

Xu et al. (2021) showed it was possible to detect features specific to UASs by using a combination of Empirical Mode Decomposition and Ensemble Empirical Mode Decomposition to classify UAS, but this was again analysing the standard 2.4GHz and 5.8GHz frequencies.

Whilst a prison has significantly less frequency noise than other facilities such as shopping centres or military bases, newer smart prisons are introducing significant wireless technologies that are likely to interfere with RF detection methods and even existing prisons have significant radio usage, and this makes Passive RF detection difficult to future proof in these environments.

Given the significant costs, infrastructure requirements, and technical limitations identified, RF detection methods were deemed unsuitable for further exploration.

Vision-Based Detection

Introduction to Vision-Based Detection

Vision-based detection uses optical sensors and cameras, combined with advanced image processing techniques, to identify, track, and monitor objects or living creatures in various environments through computer vision object detection algorithms. Vision-based detection relies on hardware such as PTZ cameras, thermal imaging, and high-resolution video systems, which enable continuous surveillance of large areas. Australian prison infrastructure employs PTZ cameras, thermal imaging, and high-resolution video systems, which form the basis of this study due to their ability to monitor airspace continuously without disrupting existing operations. Abu-Zitar et al. (2023) noted that UAS detection via existing camera infrastructure faces fewer regulatory hurdles and incurs lower costs than other detection methods. In a 2023 paper, Seidaliyeva et al. suggest the below as the key challenges in UAS detection:

- Size and speed diversity – UAS models range from expensive heavy lift UASs such as the JOUAV PH-20 with a lifting weight of 10kg and max flight speed of 20m/s (Jouav, 2024), to small, low-cost drones like the 118g Zero-X (Zero-X, 2024), highlighting the diversity in size, speed, and capability. This diversity adds complexity to UAS detection and classification tasks, as different UASs exhibit distinct flight characteristics and shapes.
- Dynamic behaviour and object differentiation - UASs exhibit unpredictable flight patterns, making it difficult to track and differentiate them from other flying objects such as birds or airplanes, complicating the detection process. Detection systems must process video frames quickly to avoid missing UASs, while maintaining accuracy to prevent misclassification. Seidaliyeva et al. (2023) mentions that the issue is so difficult that an annual challenge, called the 'Drone-Vs-Bird Detection Challenge', is held every year at the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). This dataset will be discussed later in the Methodology on Page .
- Range and Altitude - UAS are are classified as either low-altitude or high-altitude platforms. Low-altitude UASs, which are commonly used for malicious purposes, are particularly difficult due to their easy and quick deployment at a low cost. Countermeasures for these UASs must minimise collateral damage while effectively neutralising threats.
- Environmental Conditions - Environmental factors, such as adverse weather (e.g., rain, fog, wind), urban obstructions (e.g., buildings, trees), and challenging lighting conditions, significantly reduce the accuracy of UAS detection systems. Severe environmental conditions can reduce the precision and robustness of optical systems, leading to false positives or negatives. In a prison environment factors such as trees and poor lighting are less of a factor, but buildings and blind spots can affect detection accuracy.

The following literature review of vision-based detection addresses these challenges, focusing on error reduction, efficient image processing, and the development of a model that optimises both speed and accuracy.

Traditional Vision-Based Methods

Traditional vision-based methods of detection relies on rule-based, template matching and feature-based detection methods, without the use of deep learning techniques. These methods form the foundation of modern vision-based techniques, offering insight into how current approaches have evolved.

Background Subtraction Method (BSM) is a key rule-based approach that tracks motion by comparing the current frame to a reference background, detecting changes that signal movement. This is commonly implemented using algorithms such as frame difference, Mixture of Gaussian (MOG2), and k-nearest neighbour (KNN), which vary in complexity and accuracy. The process relies on background modelling to isolate moving objects, followed by feature extraction and segmentation to define object boundaries. Figure 1

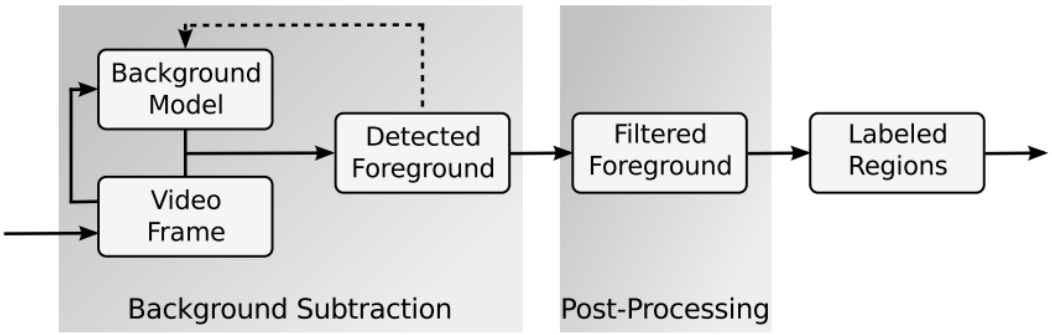


Figure 1: Background Subtraction flow with post-processing (Brutzer et al., 2011)

shows a simple flow diagram of a typical background subtraction model, illustrating how non-moving image elements are subtracted to isolate foreground regions, which can then be labelled. Brutzer et al. (2011) found that all nine BSM methods had limitations, particularly when dealing with swaying trees, moving water, variable lighting, and overlapping objects. More accurate models also had higher computational demands. This would be particularly difficult to implement in PTZ cameras that move and zoom as the background would be constantly changing.

Xiayang et al. (2013) examined frame differencing and found that existing methods were susceptible to noise and struggled to detect slow-moving objects. In the original equations for frame differencing $f_k(x, y)$ is the kth frame, $f_{k-1}(x, y)$ is the previous frame, and $D_k(x, y)$ is the difference image. Xiayang et al. introduced a grading threshold method, applying lower thresholds for small or slow-moving objects and higher thresholds for larger objects. The larger objects were stored temporarily as Motion History Images (MHI) to reduce noise detection. Combining small and large thresholds like this was given as the below set of equations, where τ is the duration of time for MHI and $M_k(x, y)$ is the detection result:

$$D_k(x,y) = f_k(x,y) - f_{k-1}(x,y) \tag{1}$$

$$D_{kL}(x,y) = \begin{cases} 1, & |D_k(x,y)| \geq \text{threshold}_L \\ 0, & |D_k(x,y)| < \text{threshold}_L \end{cases} \tag{2}$$

$$MHI_k(x,y) = \begin{cases} \tau, & D_{kL}(x,y) = 1 \\ MHI_{k-1}(x,y), & \text{else} \end{cases} \tag{3}$$

$$M_k(x,y) = \begin{cases} 1, & MHI_k(x,y) > 0 and D_k(x,y) > \text{threshold}_S \\ 0, & \text{else} \end{cases} \tag{4}$$

The conclusion of the study was that the results were satisfactory as seen in Figure 2 where image processing was minimal, but it is clear that some noise has been removed and outlines are clearer.

Figure 2: Improved Frame Difference Comparison (Xiayang et al., 2013)

Qasim et al. (2021) found that the frame difference method surpassed MOG2 and KNN, achieving an F1-score (a common testing method discussed later in the Methodology) of 81.42% and an accuracy of 89.98%. Whilst these results seem impressive, it should be noted that in all the research seen the results are of people moving in videos taking up approximately 1/3 of the image using still cameras, which shows the limits of the technology.

Annaby and Fouda (2024) presented a method for fast template matching, otherwise known as a sliding window detector, using φ-correlation and binary circuits. Template matching is a method in image processing that involves comparing a predefined template image to a target image to locate regions with a high degree of similarity. The technique works by sliding the template across the target image and calculating similarity measures at each position. Commonly used metrics include normalised cross-correlation (NCC) and zero-mean normalised cross-correlation (ZNCC). Traditional methods like NCC and ZNCC rely on pixel-by-pixel comparisons, which become computationally intensive for large datasets or high-resolution images. The study also showed that the Boolean circuit approach showed the most promise due to its combination of speed, accuracy, and robustness, while the φ-correlation coefficient offered a compelling balance of computational efficiency and precision. The models faced scalability issues, particularly in high resolution images and reliance on binary bit-plane slicing which can result in loss of detail. It was also noted that implementation may require specialised hardware, particularly in the boolean circuit algorithm.

Feature-based methods offer an alternative to traditional detection techniques. Lin et al. (2021) reviewed the evolution of feature detection and description methods in image matching, transitioning from hand-crafted techniques like SIFT and SURF to deep learning-based approaches. They highlighted the critical steps in feature-based image matching, including feature detection, affine shape estimation, orientation assignment, and feature description, which collectively address the challenges of matching images under varying scales, rotations, and lighting conditions. The study emphasised the increasing adoption of deep learning, with techniques like Siamese and triplet CNNs providing significant advancements in descriptor learning and robustness against geometric and radiometric variations.

Although these advancements improved matching accuracy, several limitations remain. Current methods struggle with high inter-class variability, large-scale negative pair comparisons, and limited exploration of matching pair variability. The transferability of deep learning-based methods across different imaging domains, such as aerial and close-range scenes, remains under-explored. Lin et al. concluded by emphasising the need for integrating domain knowledge with deep neural networks to enhance robustness and address these challenges.

Traditional methods like rule-based detection, template matching, and feature-based approaches form the basis of image detection, but they struggle in dynamic, real-world scenarios. These methods often struggle with scalability, robustness to noise, computational efficiency, and adaptability to diverse environments, especially with modern challenges like variable lighting, background movement, and high inter-class variability. Relying on manual rules or descriptors reduces their ability to work effectively across different datasets or imaging environments. In contrast, deep learning approaches use neural networks and large datasets to learn robust features automatically, resulting in improved accuracy and adaptability in complex scenes. The following section examines how deep learning methods overcome these limitations, making them a leading approach in vision-based applications.

Deep Learning for Vision-Based Detection

Janiesch et al. (2021) and Ren and Wang (2022) both explore the evolution of deep learning algorithms, explaining their impact on tasks like object detection and classification compared to traditional methods. Janiesch et al. (2021) demonstrated how CNNs automate feature extraction for object detection tasks by applying filters to image data, identifying edges, shapes, and textures critical to object classification, reducing the need for manual feature engineering. Similarly, Ren and Wang highlight the evolution of object detection from traditional techniques to CNN-based methods, outlining advancements from recurrent CNN (RCNN) and its successors to modern algorithms like YOLO, Single Shot Detectors (SSDs), and RetinaNet, which have significantly improved detection speed and accuracy for real-time use.

CNNs are a class of deep learning models widely used for image processing tasks. They consist of several key layers:

1. Input Layer - Receives raw image data.

2. Convolutional Layers: Apply filters to the input to extract features such as edges and textures. These layers utilize local receptive fields, sparse weights, and parameter sharing, enabling translation and scale invariance.

3. Activation Layers (e.g., ReLU): Introduce non-linearity to the model, allowing it to learn complex patterns.

4. Pooling Layers: Reduce the spatial dimensions of the data, which decreases computational load and helps prevent overfitting.

5. Fully Connected Layers: Integrate features learned by previous layers to perform classification or regression tasks.

The layered structure of CNNs enables the learning of hierarchical features, which improves performance in image classification and object detection.

Both studies highlight the effectiveness of CNNs in feature extraction, classification, and hierarchical representation learning from raw data. Innovations like anchor boxes, feature pyramids, and adaptive loss functions, have improved the performance of these methods. However, the authors also acknowledge shared challenges, including high computational demands, interpretability issues, and susceptibility to biases in training data. Ren and Wang identify additional challenges, such as detecting small objects, processing multi-scale images, and optimizing models for low-resource devices. Despite these limitations, advancements like transfer learning and architectural improvements have expanded the accessibility and applicability of deep learning. BBoth studies emphasise the need for effective data management, model explainability, and bias reduction to maximise deep learning's potential in intelligent systems.

Fieres et al. (2006) proposed a method for training CNNs with binary threshold neurons, optimised for low-power mixed-signal hardware. Using clustering methods for convolutional layers and Perceptron learning for output layers, the approach was tested on MNIST and traffic sign datasets. This approach proved robust to hardware inaccuracies, competitive performance with simplified weight quantisation (-1, 0, +1), and effective feature extraction. The study concluded that CNNs with threshold neurons are viable for low-power hardware, achieving strong performance while simplifying computation. However, its accuracy falls short of backpropagation-trained networks, and it faces scalability challenges in larger systems.

Nalamati et al. (2019) focused on detecting small UASs in long-range surveillance videos, where UASs often blend into complex backgrounds and resemble birds or other objects. The study evaluates several CNN-based object detection models, including Faster RCNN with ResNet-101 and Inception v2 architectures, as well as the SSD. Due to limited datasets, pre-trained models and transfer learning techniques were utilised. The study found that Faster RCNN with ResNet-101 performed best, achieving an average precision (AP) of 40.99%, while SSD exhibited the fastest performance but the poorest detection accuracy. The authors highlighted the need for further improvements in detection time for real-time applications and suggested that future work focus on optimising the models for speed and scalability in practical scenarios.

Z Deng et al. (2017) suggests an aerial vehicle identification system with a specially designed vehicle detection system comprised of an Accurate Vehicle Proposal Network

(AVPN) and Vehicle Attributes Learning Network (VALN). The AVPN and VALN aim to overcome the shortcomings in RCNN methods, suggesting that traditional RCNN methods are too slow for real-time detection and suggesting the 'faster-RCNN' model, with near real-time detection, is not appropriate for aerial imaging techniques. The AVPN takes inspiration from the Zeiler and Fergus model, a five-layer approach to create a map cupe from the actual input and training model, named the 'hyper feature map', which was used in conjunction with a sliding window to generate bounding boxes. This was promising, but had occasional false positives and missing detections, especially in occluded or complex scenes and was computationally demanding.

Mubarak et al. (2022) focused on detecting UASs using the Mask RCNN deep learning model, which combines object detection with instance segmentation for improved performance. The study addresses challenges such as small object size, occlusion, and variable flight dynamics in real-world scenarios. Mask RCNN's architecture leverages region proposal networks and multi-task learning to accurately segment UASs from backgrounds. The findings show that Mask RCNN effectively detects UASs with high precision and recall, even in cluttered or dynamic environments. However, the model's computational demands pose a challenge for real-time applications, and its performance degrades under significant environmental variations or with very small UASs. Future improvements include optimising Mask RCNN for faster inference and exploring advanced data augmentation techniques to improve robustness.

Tang et al. (2024) provided a comprehensive survey of deep learning-based object detection methods for UASs, emphasising their applications in fields such as monitoring, precision agriculture, and disaster response. The study shows the unique challenges of UAS object detection, including small object sizes, complex backgrounds, object rotation, scale variations, and category imbalance. It compares one-stage and two-stage detection methods, noting that one-stage algorithms, such as YOLO and SSD, are faster and better suited for real-time applications, while two-stage algorithms, like Faster RCNN, achieve higher accuracy. Figure 3 shows the high-level process for a SSD like YOLO, while Figure 4 shows two-stage detection such as RCNN.

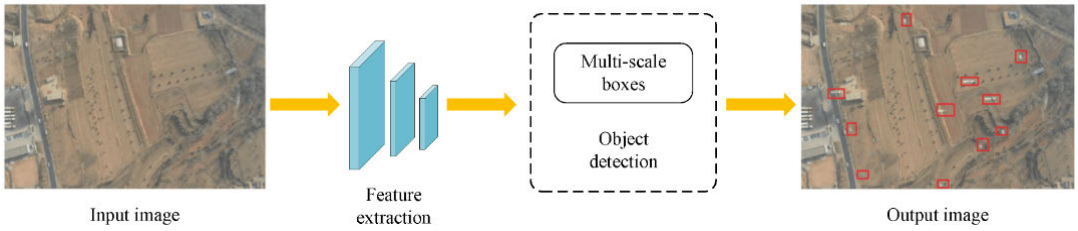


Figure 3: Framework for single-stage detection (Tang et al., 2024)

Deep learning methods have improved UAS object detection, but challenges remain with detecting small objects, handling scale changes, and managing complex backgrounds. Future directions include optimising models for real-time performance, improving performance in cluttered environments, and addressing computational limitations to better meet the demands of UAS applications.

Laurito (2020) demonstrated by using off-the-shelf components and a pinhole camera they could detect UAS in real time out to 35 metres. This result is impressive since it relied on older algorithms and a camera with a 1600-pixel resolution, which has since

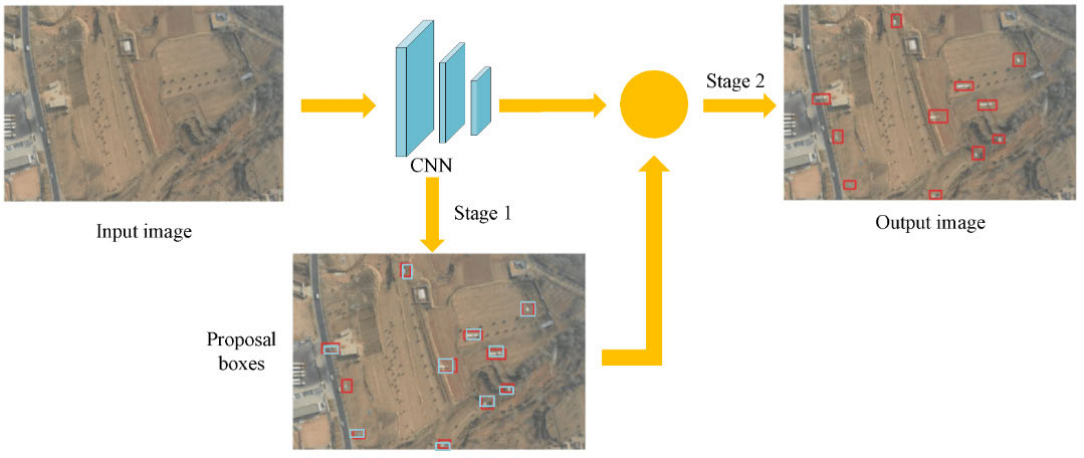


Figure 4: Framework for two-stage detection (Tang et al., 2024)

seen significant improvements. All processing was conducted on a custom-built UAS with a stripped onboard Intel i7 NUC, showing this is capable of producing promising results with low processing power. This design used the YOLO3 SSD algorithm with pre-trained weight files for transfer learning and used relative distance estimation and deflection angle estimates. The model initially overestimated the bounding box and miscalculated the distance, so an alternate method was used by calibrating with known distances to improve accuracy.

Alsanad et al. (2022) proposed a modified YOLO3 algorithm to improve real-time UAS detection, focusing on small object recognition, complex backgrounds, and computational efficiency.. improved YOLO3 with a lighter CNN architecture (Darknet-49), dense connectivity to enhance feature reuse, and a multi-scale detection module for better small-object detection. The modified model was tested on a custom dataset derived from UAS videos. The modified YOLO3 achieved superior performance compared to the standard YOLO3, with an accuracy of 95.60% and an average precision of 96%. It also demonstrated faster processing speeds (60.3 FPS) due to the reduced complexity of the Darknet-49 architecture and noted it was effective in real-time environments, but struggled with scalability.

The theme from these studies is that two-stage detectors offer higher accuracy but require more computation, while single-stage detectors generate location and category data directly from images, providing faster performance but struggling with smaller objects. This paper will continue to investigate SSD methods of object detection for high-speed UAS, specifically around the YOLO framework.

YOLO Framework Overview

The YOLO framework has improved significantly over the years from the original YOLO model to the current YOLO11 model released on September 27, 2024. Figure 5 shows the rapid progression of the YOLO framework over the last 10 years, and although this research was conducted this year, it still does not include the latest YOLO11 model. Figure 5 shows that the YOLO model has been in development for almost 10 years, and the majority of research in UAS detection using the YOLO framework are focused on YOLO3, YOLO4 and YOLO8, released in 2018, 2020 and 2023 respectively.

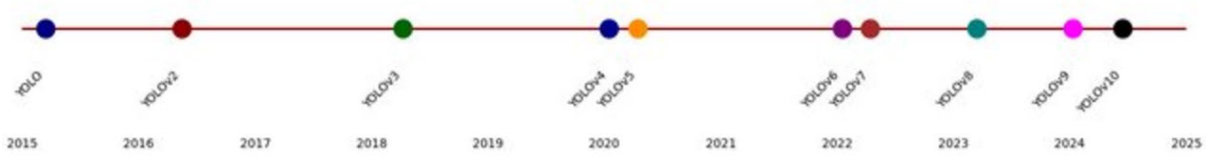


Figure 5: YOLO Timeline (Sapkota et al., 2024)

Sapokta et al. (2023) provided a detailed review of the YOLO framework's development. YOLO1, introduced by Joseph Redmon et al. in 2016, was the first model to unify object detection into a single neural network pass. Unlike region proposal networks like Faster RCNN, YOLO1 divided the image into a grid and directly predicted bounding boxes and class probabilities. This design enabled real-time detection with speeds previously unattainable in object detection models. However, small and overlapping objects posed challenges for YOLO1 due to its coarse grid-based predictions. Localisation accuracy and bounding box precision were also less robust compared to multi-stage detectors, highlighting key areas for improvement.

YOLO2 improved YOLO1 by introducing anchor boxes, a concept borrowed from Faster RCNN. Anchor boxes improved the detection of small and overlapping objects by providing predefined shapes for bounding boxes. YOLO2 also incorporated batch normalisation to reduce overfitting and accelerate training. Multi-scale training also enabled YOLO2 to maintain strong performance across images of varying resolutions. YOLO2 introduced YOLO9000, which merged classification and detection datasets using hierarchical classification, enabling the detection of over 9,000 classes. These changes improved the accuracy and versatility of the model while maintaining real-time performance. YOLO3 further improved accuracy and versatility, partisularly in smaller objects, by introducing the feature pyramid network (FPN), used the Darknet-53 backbone previously discussed and improved bounding box predictions.

Released in 2020, YOLO4 optimized YOLO3 with advanced techniques to balance speed and accuracy, including CSPDarknet53 and PANet for better feature aggregation. Techniques such as Mosaic data augmentation, DropBlock regularisation, and CIOU (Complete Intersection over Union) loss were added under the 'Bag of Freebies' to improve generalisation without increasing inference time. The 'Bag of Specials' included Mish activation and Spatial Pyramid Pooling (SPP) for improved detection performance. With PANet for better feature aggregation and spatial context, YOLO4 achieved excellent results in both efficiency and accuracy, making it one of the most balanced models in the series.

Figure 6 below shows the structure of the YOLO4 network, and shows the increasing complexity of the SSD model.

YOLO5 differed from the previous models in the YOLO series due to its development by the open-source community rather than the original creators. YOLO5 prioritised usability, modularity, and efficiency, making it a popular choice for real-world

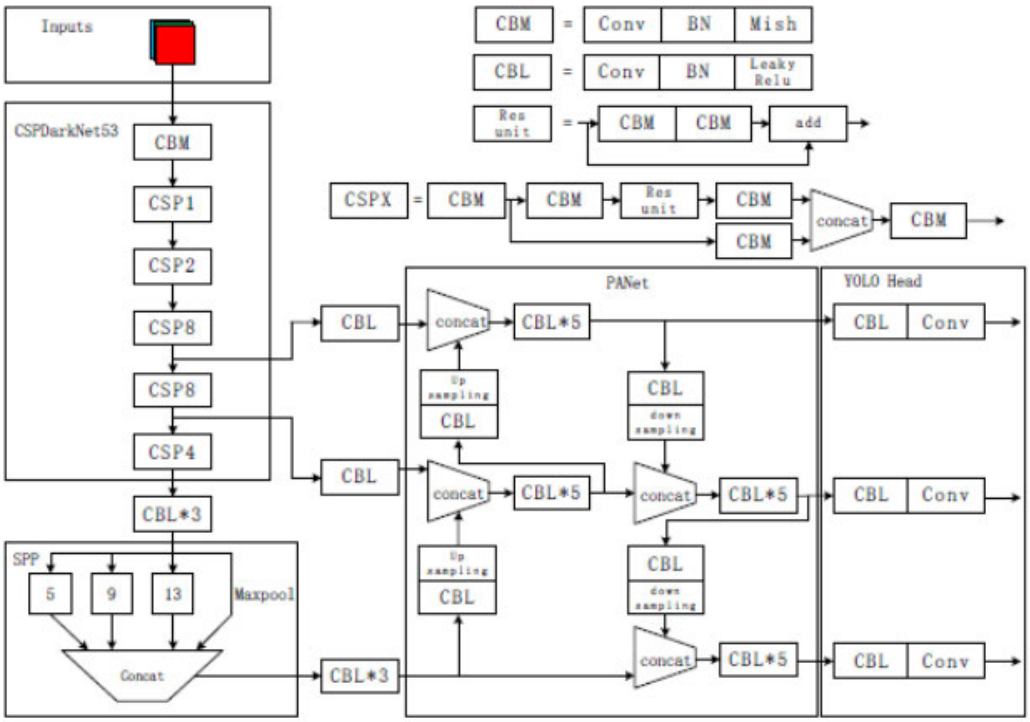


Figure 6: YOLO4 structure

applications. Written in PyTorch, YOLO5 offered greater flexibility and simpler deployment than the original Darknet framework. IT introduced mixed precision training to reduce memory usage and speed up training, while auto-learning of anchor boxes enhanced detection without manual adjustments. YOLO5 also supported multi-GPU training and included pre-trained models optimised for lightweight, medium, and large configurations, making it suitable for real-time and resource-limited scenarios.

YOLO6 optimised models for industrial applications by improving efficiency in resource-constrained environments. It used techniques such as decoupled head architecture and advanced post-processing algorithms, reducing latency during inference. While maintaining competitive accuracy, YOLO6 was tailored for edge devices, offering lightweight variants that excelled in scenarios requiring low power and high speed.

YOLO7 targeted model performance improvements through training efficiency and accuracy for small object detection. It introduced extended E-ELAN (Enhanced Expandable Layered Aggregation Networks) for better feature extraction and model scaling. YOLO7 also improved its feature pyramid and path aggregation strategies, delivering higher accuracy for challenging detection tasks while keeping inference times low.

YOLO8 expanded on previous models by using recent advancements in object detection. It introduced anchor-free detection, removing predefined anchor boxes to enhance flexibility and lower computational costs. Decoupled detection heads were employed to separate classification and regression tasks, improving precision and speed. YOLO8 also used advanced loss functions, such as dynamic label assignment and focal loss, to address class imbalance and refine detection accuracy. Its streamlined architecture allowed it

to perform exceptionally well in diverse applications, enabling superior performance in applications requiring real-time object detection.

Zamri et al. (2024) optimized the YOLO8n model for small UAS detection by incorporating attention mechanisms and architectural changes. The authors added attention modules such as CBAM, GAM, ECA, and ResCBAM to the model's neck, along with a high-resolution detection head for better small-object detection. They also developed a new dataset, BirDrone, with images of small UASs and birds to train the model for real-world scenarios. The P2-YOLO8n-ResCBAM model improved mAP from 90.3% to 92.6%, while maintaining a real-time processing speed of 166 frames per second (fps). The model successfully distinguished between UASs and birds, even at long distances, though it struggles with occluded objects and relies on high-performance hardware for optimal operation.

Zamri et al. (2024) also published a separate paper specifically for the BirDrone dataset to address the unique challenges in distinguishing UASs from birds and detecting small targets in a range of conditions. This dataset includes 2,970 high-quality images annotated for UASs and birds, incorporating varied lighting and complex backgrounds. Pre-processing involved auto-orientation, resizing, and auto-contrast adjustment, while data augmentation increased image diversity to reflect real-world conditions. This was tested using YOLO9, and showed higher UAS detection accuracy and fewer false alarms compared to other datasets. The dataset proved effective in improving detection capabilities, especially in security and surveillance applications.

Zhai et al. (2024) aimed to improve UAS detection by improving the YOLO8 model, addressing challenges like tiny target sizes, complex airspace backgrounds, and varying light conditions. The model features a high-resolution detection head for small targets and uses SPD-Conv and GAM attention mechanisms to enhance feature extraction and fusion. These modifications significantly improved the precision, recall, and mAP by 11.9%, 15.2%, and 9%, respectively, while reducing model parameters and size by nearly 60%. The improved model outperformed other YOLO versions and baseline models, achieving better detection accuracy and speed suitable for edge devices. However, the model faced two main limitations: increased inference time due to added complexity and reduced recall in complex airspace scenarios.

YOLO9 introduced advanced spatial-channel decoupling techniques to further improve feature extraction efficiency. It incorporated dual assignment strategies, improving detection accuracy for overlapping objects and complex scenarios. YOLO9 was optimized for large-scale datasets, ensuring fast real-time processing.

YOLO10 refined model precision and adaptability, combining hybrid anchor-free and anchor-based detection to improve flexibility and robustness. With improved label assignment and multi-task learning, YOLO10 excelled in diverse detection tasks, supporting applications in autonomous vehicles and surveillance.

Ultralytics (2024), the developer of the YOLO11 model, states that YOLO11 features an improved backbone and neck architecture to enhance feature extraction for precise object detection. It achieves higher efficiency and speed through improved designs and training pipelines, balancing detection accuracy with processing speed. Ultralytics suggests that the YOLO11m demonstrates greater accuracy with 22% fewer parameters than YOLO8m on the COCO dataset, improving computational efficiency. Khanam

and Hussein (2024) state the key innovations include the C3k2 block for faster and more efficient feature extraction, the SPPF (Spatial Pyramid Pooling - Fast) module, and the C2PSA (Convolutional block with Parallel Spatial Attention) for improved attention mechanisms. These blocks are shown below in Figure 7, split purposefully into backbone, neck and head sections for clarity. These architectural changes improve detection performance, especially in challenging scenarios involving small or occluded objects. YOLO11 shows improvements in computational efficiency, achieving high mAP scores with fewer parameters than earlier versions

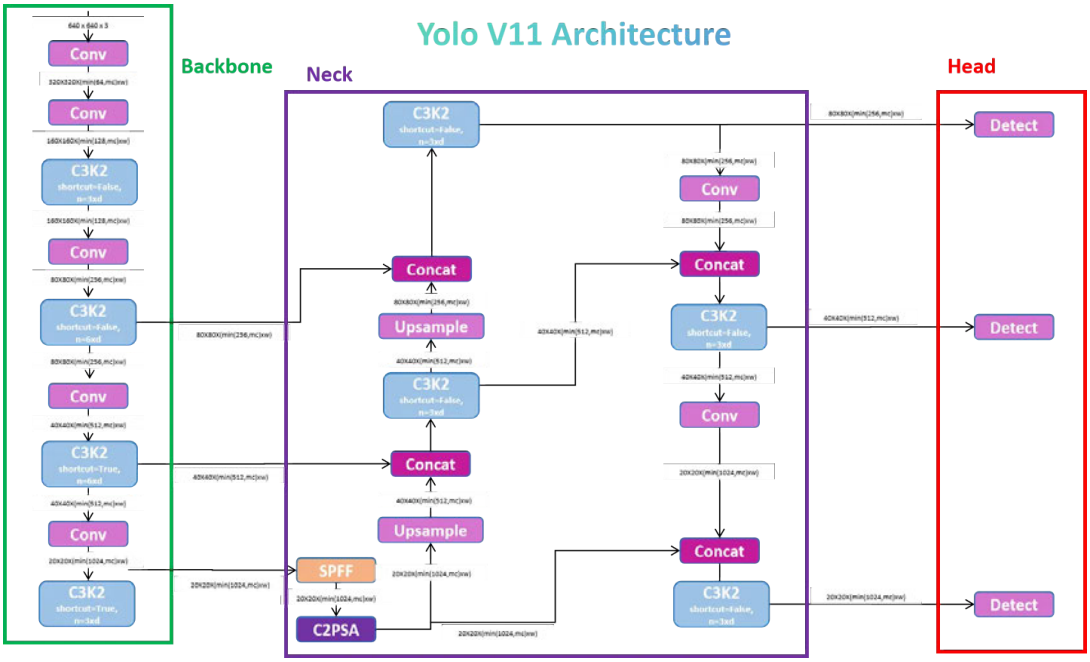


Figure 7: YOLO11 Architecture (Rao, 2024)

It should be noted that the YOLO framework has variants within the model that can be used depending on the balance needed between speed and accuracy as seen in Table 1 with different variants for Nano (n), Small (s), Medium (m), Large (l), and Extra Large (x). The information shown in Table 1 is reflected in Figure 5 where it is important to note that for all models, as the size of the variant increases so does the accuracy (mAP^{val}) and computation time (ms/image). The performance of these models will be discussed more in the Methodology.

Model	**Size**	mAP^{val} **50-95**	**Speed** **T4 TensorRT10** (ms)	**Speed** **CPU ONNX** (ms)	**Params** (M)	**FLOPs** (B)
YOLO11n	640	39.5	1.5 ± 0.0	56.1 ± 0.8	2.6	6.5
YOLO11s	640	47.0	2.5 ± 0.0	90.0 ± 1.2	9.4	21.5
YOLO11m	640	51.5	4.7 ± 0.1	183.2 ± 2.0	20.1	68.0
YOLO11l	640	53.4	6.2 ± 0.1	238.6 ± 1.4	25.3	86.9
YOLO11x	640	54.7	11.3 ± 0.2	462.8 ± 6.7	56.9	194.9

Table 1: Performance comparison of YOLO11 models (Ultralytics, 2024)

The only peer-reviewed literature found for YOLO11 implementation is from Sharma et al. (2024), where they evaluated and compared the performance of YOLO8, YOLO9, YOLO10, YOLO11, and Faster RCNN for detecting multiple weed species in agricultural settings. The authors created an annotated image database with bounding boxes for species such as cocklebur, dandelion, and Palmer amaranth, totalling 2,348 images. These images were processed and augmented to ensure robustness under diverse conditions. The study concluded that:

- YOLO11 achieved the fastest inference time (13.5 ms) while maintaining high accuracy (mAP50 of 0.921 during testing).

- YOLO9 had the highest mAP50 (0.935) but required more computational resources (54.2 ms inference time).
- YOLO8 and YOLO10 balanced accuracy and speed, with YOLO8 demonstrating robust real-time capabilities.
- Faster RCNN lagged in both speed (63.8 ms) and accuracy (mAP50 of 0.821), highlighting its inefficiency for real-time applications.

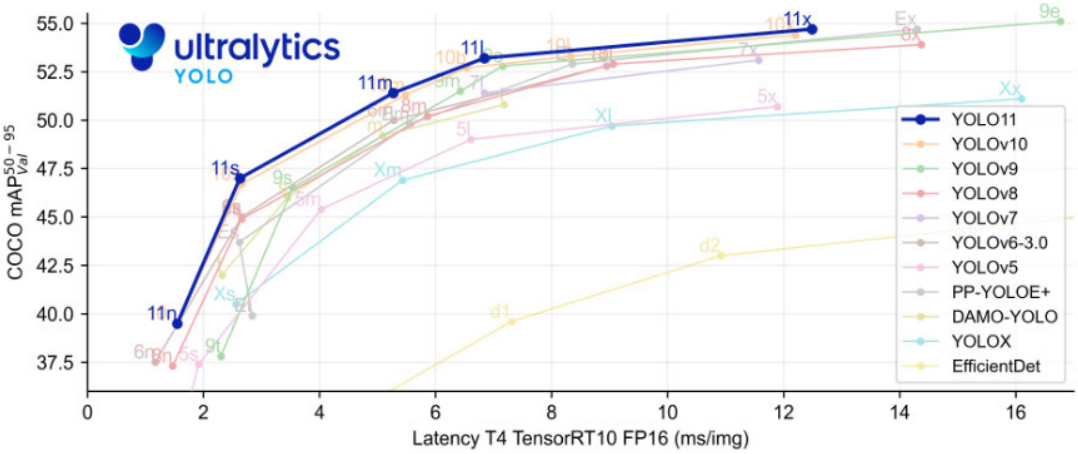


Figure 8: YOLO Model Performance Comparison (Ultralytics, 2024)

The study concluded that YOLO models, particularly YOLO11 and YOLO8, are well-suited for real-time weed detection, offering a viable solution for precision agriculture. More importantly, it demonstrated the efficacy of YOLO11 in real-time object detection with high accuracy.

Hyperparameter Tuning

Hyperparameter tuning is a critical component of machine learning model development, as it directly influences model accuracy, convergence speed, and generalisation. Hyperparameters differ from model parameters in that they are set before training begins and control the learning process, affecting aspects such as learning rate, batch size, and model architecture. Effective hyperparameter tuning is critical in optimising performance of machine learning models, particularly in complex systems like YOLO11.

Traditionally, hyperparameter tuning was conducted manually through methods such as grid search and random search. In grid search, a fixed range of hyperparameter values is predefined, and exhaustive combinations of these values are tested. Random search samples values from the hyperparameter space at random. While simple and effective for small search spaces, these methods have large trade offs such as high computaional cost, inefficient exploration and human intervention to set ranges, initiate trials and monitor results.

To overcome limitations, modern approaches use automated hyperparameter optimisation (HPO) tools like Optuna, which use intelligent search strategies and pruning techniques

to optimise hyperparameters more efficiently. Unlike grid and random search, Optuna leverages a Tree-Structured Parzen Estimator to model the search space, focusing exploration on high-potential regions for faster convergence. Optuna also supports pruning algorithms that stop underperforming trials early, cutting computational costs.

Almarzooq and Waheed (2024) demonstrated that Optuna outperforms manual tuning methods. Their work on seismic data inversion models found that Optuna outperformed grid search on key performance metrics, with the Optuna-based models achieving lower root mean square error (RMSE) and faster convergence. The Optuna approach also required fewer trials to identify optimal parameters, which directly translates to reduced computational time and resource usage. The authors stated that dynamic trial selection and model importance analysis were critical for optimising performance.

A key feature of Optuna is its ability to prune unpromising trials using Successive Halving Pruning, a technique designed to improve computational efficiency. Pruning stops trials early if their intermediate performance does not meet a specified threshold, allowing computational resources to be reallocated to more promising trials. Unlike traditional early stopping, which halts training for overfitting prevention, successive halving focuses on computational efficiency, ensuring that unpromising trials are eliminated early. Successive halving is particularly useful for high-cost models like YOLO11, where training time can be significant.

This approach, as demonstrated by Akiba et al. (2019), significantly accelerates the search process. Their work found that pruning reduced the total number of fully trained trials while maintaining or improving final model performance. By halving the number of active trials at each stage, computational resources are focused on only the most promising configurations. For example, in one study involving a subnetwork of the AlexNet architecture, pruning enabled the exploration of over 1200 trials, of which 1271 trials were pruned, compared to only 35 full trials completed under grid search.

Optuna's use in YOLO modelling is well-supported by its computational efficiency and tuning capabilities. YOLO models require tuning hyperparameters such as learning rate, batch size, momentum, and weight decay, all of which are computationally expensive to tune manually. Optuna efficiently explores hyperparameters, improving performance with minimal manual effort. This process promotes reproducibility and transparency, while reducing the risk of human error in hyperparameter tuning.

Literature Conclusions

Studies like Zamri et al. (2024) and Sharma et al. (2024) show YOLO's effectiveness in challenging applications, including UAS detection. Zamri et al. showed that YOLO8 could distinguish UASs from birds using improved datasets like BirDrone, even under variable environmental and lighting conditions. Similarly, Sharma et al. showed that YOLO11 achieves the fastest inference times (13.5 ms) while maintaining high accuracy (mAP50 of 0.921), outperforming earlier models like YOLO8, YOLO9, and YOLO10 in real-time detection.

Although YOLO9 had higher accuracy in some scenarios, its computational demands restricted its use in resource-constrained environments.YOLO11 offers a balance of speed,

accuracy, and computational efficiency, supporting use on edge devices, cloud platforms, and high-performance systems. By incorporating features such as high-resolution detection heads and advanced attention mechanisms, YOLO11 addresses small-object detection, cluttered backgrounds, and computational overhead.

YOLO11 has key features that support real-time UAS detection, including:

1. **Enhanced Small Object Detection:** YOLO11 improves small-object detection by using high-resolution detection heads and the C2PSA module, which strengthens feature extraction and increases detection accuracy for small, fast-moving objects.
2. **Real-Time Performance:** YOLO11's optimized architecture delivers fast inference times, with Sharma et al. (2024) reporting a speed of 13.5 ms. This speed is essential for UAS detection, where rapid detection prevents security threats or operational delays.
3. **Robust Feature Extraction:** The use of C3k2 blocks and improved neck designs enables YOLO11 to identify objects in cluttered or dynamic backgrounds, a common issue in airspace surveillance where UASs may blend with birds, clouds, or other objects.
4. **Adaptability:** YOLO11 is designed to operate efficiently across various environments, including edge devices, cloud systems, and NVIDIA GPUs. This adaptability ensures that the model can be deployed in a range of UAS detection scenarios, from fixed surveillance cameras to mobile UAS tracking systems.
5. **Proven Results in Related Applications:** Sharma et al. show that YOLO11 achieves strong results in detection tasks, indicating its suitability for broader UAS detection contexts.

In conjuction with training using the BirDrone dataset, the YOLO11 seems the obvious choice for exploration in real-time detection of UAS. The knowledge gap that is present is the efficacy of YOLO11 in UAS detection.

METHODOLOGY

Overview

The development of the UAS detection model followed a structured, multi-phase process designed to ensure success in identifying UAS under diverse conditions in correctional facilities. This process was divided into four key stages:

1. **Setup** - Preparing the hardware, datasets and environment.
2. **Benchmarking** - Training and evaluating multiple YOLO variants to determine the most suitable model.
3. **Optimisation** - Fine-tuning hyperparameters to maximise the performance of the selected YOLO model.
4. **Testing** - Evaluating the final optimised model's generalisation, robustness, and real-time detection capabilities.

This is visualised best in Figure 9. The benchmarking and optimisation stages were both trained from scratch, with the benchmarking variants being trained and validated once, and the optimisation model being trained and validated over 50 trials of varying hyperparameters. This was eventually done again on a further refined set of hyperparameters over 10 trials, but this will be discussed later.

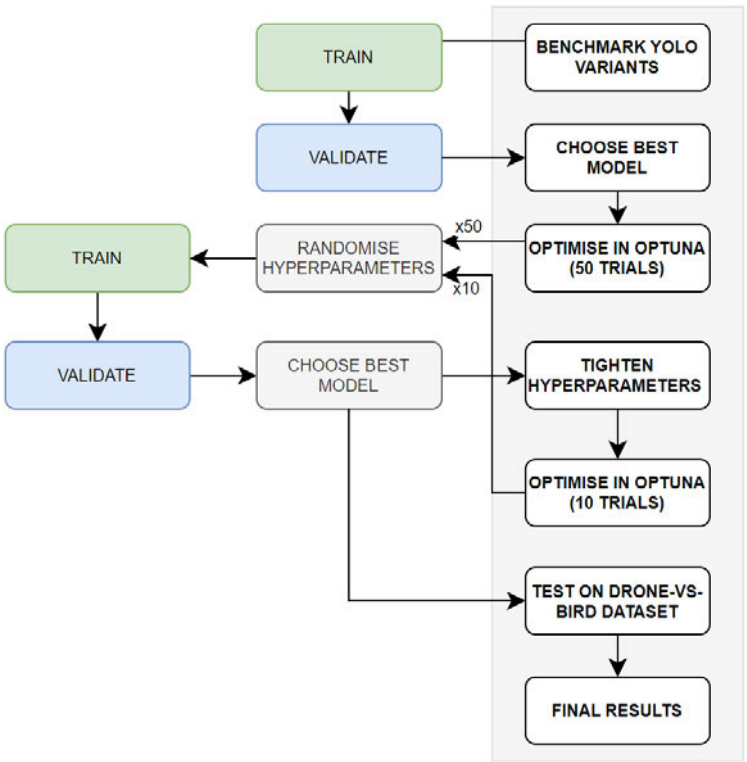


Figure 9: Workflow of Benchmarking, Optimising and Testing the YOLO model

Hardware Setup

To optimise both the training and testing phases of the UAS detection models, two distinct hardware configurations were employed: a high-performance desktop computer for intensive computational tasks and a portable laptop for mobile testing scenarios.

Desktop Computer

- **OS:** Windows 11
- **Processor:** Intel Core i7-12700K
- **Memory:** 32GB RAM, 1TB SSD
- **GPU:** NVIDIA GeForce RTX 3080

This system was utilised for computationally intensive tasks such as training and validating the models, leveraging the better GPU for accelerated processing. It was thought the desktop would be more than capable of the model training, but it became evident later that this was not the case.

Mobile Hardware

- **OS:** Windows 11
- **Processor:** AMD Ryzen 9 5900HX
- **Memory:** 16GB RAM, 1TB SSD
- **GPU:** NVIDIA GeForce RTX 3070Ti Mobile

The laptop enables flexibility in testing the model in various settings, reflecting real-world deployment scenarios. It was mostly used for smaller graphics tasks, like the YOLO11 benchmark testing over 75 epochs.

Virtual Machine - Google Cloud

- **OS:** Google Cloud VM - Debian 11, Python 3.10, with PyTorch 2.4 and fast.ai preinstalled
- **Processor:** Intel® Xeon® Scalable Platinum 8173M (2 cores only)
- **Memory:** 7.5GB RAM, 100GB SSD
- **GPU:** NVIDIA V100

It became evident during training the desktop computer was not powerful enough for the Optuna trails. To optimise training times and leverage the computational power of an NVIDIA V100 GPU, a cloud-based virtual machine on Google Cloud was used. This GPU, over twice as powerful as the NVIDIA 3080, significantly improved the efficiency of

training the YOLO11 models. The process began by uploading the training datasets and Python scripts to the virtual machine and setting up the Python environment for YOLO. Once setup, the training files were executed via Secure Shell (SSH), enabling remote control and monitoring of the training process. This allowed for efficient utilisation of cloud resources and faster model convergence.

Dataset Preparation

Training/Validation Dataset - BirDrone

The training of the YOLO11 model, including its variants YOLO11m, YOLO11l, and YOLO11x, was conducted using the BirDrone dataset. This dataset provides diverse imagery of UAS across various environmental conditions, which is crucial for ensuring robust and generalised model performance. The BirDrone dataset includes annotated images of UASs in multiple scenarios, including varying altitudes, backgrounds, and lighting conditions. These variations ensure that the models can learn to detect UASs in real-world environments, such as correctional facility perimeters. The dataset was split into 70% for training, 15% for validation, and 1% for testing, balancing diversity across all subsets.

Testing Dataset - Drone-vs-Bird

Collucia et al. (2023) developed the Drone-vs-Bird Detection Grand Challenge dataset to advance research in UAS detection, particularly in distinguishing UASs from birds in complex visual environments. The dataset was introduced as part of the IEEE ICASSP 2023 conference to encourage the development of robust detection algorithms capable of performing under real-world conditions. To achieve this, the dataset includes video footage captured with both static and moving cameras, featuring a range of UAS models, such as DJI Inspire and DJI Phantom, as well as smaller custom-built UASs. The footage was recorded in diverse settings, including urban areas, forests, and maritime environments, and under varying weather conditions like sun, cloud, and low light. This variety ensures that detection models are tested against the types of environmental variability likely to be encountered in practice.

One of the defining features of the dataset is the inclusion of birds, which presents a unique classification challenge. Drones and birds often exhibit similar shapes, sizes, and flight patterns, making it difficult for detection models to differentiate between them. The dataset also includes frames with small UAS representations, from as few as 15 pixels to over a million pixels in size. This variation ensures that detection algorithms are tested on both near and distant UAS sightings, simulating realistic surveillance conditions.

The decision to use the Drone-vs-Bird Detection Grand Challenge dataset for testing the YOLO11 optimised model is well-supported as the training dataset, BirDrone, is also focused on UAS and bird classes. The dataset allows the performance of the YOLO11 model to be tested under conditions that closely mimic real-world UAS detection challenges, ensuring it is both robust and reliable. This research piece was lucky enough to be granted access to this dataset, and five of the provided videos were used as the real-time testing dataset.

Performance Quantifiers

The success of the YOLO11-based UAS detection system was determined by evaluating its performance against predefined thresholds across critical metrics. These criteria were carefully selected to ensure that the model aligned with the operational and security requirements of high-security environments. The following benchmarks formed the foundation for the evaluation:

Mean Average Precision (mAP)

Mean Average Precison (mAP) evaluates the model's performance in object detection tasks by averaging precision values across multiple Intersection-over-Union (IoU) thresholds. IoU measures the overlap between predicted bounding boxes and ground truth bounding boxes, ensuring accurate localisation of detected objects, defined as:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Average Precision (AP) is determined by plotting a PR curve for each class and calculating the area under the curve. mAP is the mean of the AP values across all classes (in this case, primarily the UAS class). For the validation dataset, predictions are compared to ground-truth annotations at multiple IoU thresholds (e.g., 0.5, 0.75). Higher mAP values reflect the model's ability to simultaneously achieve high detection accuracy and precise localisation. This is particularly crucial in detecting small, fast-moving UASs, where minor localisation errors could render the detection ineffective. mAP is also sensitive to both classification accuracy and bounding box quality, making it an essential metric for fine-tuning object detection systems like YOLO11.

Precision

Precision measures the proportion of true positive detections (correctly identified UASs) among all positive detections made by the model. This metric focuses on avoiding false positives—incorrectly classifying objects as UASs when they are not. In a prison environment, false alarms can have significant consequences, such as wasting valuable resources or diverting attention away from real threats. For instance, false positives triggered by birds, debris, or other benign objects could lead to unnecessary disruptions or erode confidence in the detection system. Achieving high precision ensures that alarms are credible and actionable, maintaining operational efficiency and reducing unnecessary interventions. Precision is calculated using the formula:

$$\text{Precision} = \frac{TP}{\text{TP + FP}}$$

Recall

Recall quantifies the proportion of true positives correctly identified out of all actual positive cases in the dataset. In the context of UAS detection, this metric is critical for ensuring no UAS escapes detection, as undetected UASs pose severe security risks. High

recall ensures the system's reliability in identifying threats under various conditions, such as poor lighting or occluded views. Recall is calculated as:

$$\text{Recall} = \frac{TP}{\text{TP} + \text{FN}}$$

F1-Score

The F1-score is the harmonic mean of precision and recall, balancing the trade-off between avoiding false positives and ensuring no true positives are missed. This is particularly important in scenarios where the dataset is imbalanced, as in UAS detection, where UASs (positives) are rare compared to non-UAS objects (negatives). The F1-score provides a single measure that combines the strengths of precision and recall, ensuring the model performs well in both avoiding false alarms and maintaining detection reliability. The formula is:

$$\text{Recall} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-score is particularly useful when there is an imbalance in the dataset (e.g., fewer positive samples relative to negatives), ensuring that both precision and recall are equally considered.

Model Testing Criteria

These metrics collectively ensure that the detection system is both reliable and efficient for real-time UAS detection. Precision guarantees minimal false alarms, preserving the integrity of security operations. Recall ensures no UAS goes undetected, maintaining the facility's safety. The F1-score balances these two aspects, while mAP provides an overall assessment of detection performance, including localisation and classification. Together, these metrics ensure that the proposed UAS detection system is optimised for the complex and dynamic requirements of correctional facilities.

mAP was chosen as the primary metric for this study, 50% across IoU thresholds selected as the baseline, with a target of 50% or higher. This is called mAP50. This metric was chosen to evaluate the model's ability to accurately localize UASs within complex and varied prison environments. High mAP values provided confidence in both the detection accuracy and the precision of object localisation, which are critical in distinguishing UASs from other objects. This was the primary metric for validation, with a result of greater that 0.4 considered good for tough datasets, and above 0.5 being considered excellent.

F1-score, precision and recall were also used to determine a models efficacy. These metrics ensured that the detection systems were rigorously evaluated across dimensions of accuracy, speed, and operational reliability. These benchmarks were designed to validate the system's suitability for deployment in high-security environments, emphasizing both its technical capabilities and practical application.

Model Training

After selecting the hardware and preparing the dataset, the next step was the benchmarking of the YOLO11 variants. This process involved both pre-trained weights

and training from scratch to explore the capabilities and limitations of various model configurations. The goal of this training phase was to benchmark the models and determine which was best for future optimisation.

Using the information given for YOLO11 performance in Figure 8 all models exceed the performance requirement with the slowest being 11.2 ± 0.2ms in the YOLO11x model. While this has the greatest detection accuracy, the gains in accuracy over the YOLO11l model are minimal in comparison for almost double the processing time. YOLO11x was chosen to be tested to see if the gains are negligible for the computation time.

Using this information, the benchmarking process was conducted using 3 different configurations:

- YOLO11m – Medium. Prioritises inference speed and resource efficiency.
- YOLO11l – Large. Balance of medium and extra-large models.
- YOLO11x – Extra Large. Prioritises detection accuracy at the expense of computational power.

Training Environment

The training environment was set up using Anaconda, using the Spyder IDE v5.5.1, known for its integrated development environment for scientific computing and machine learning. Python v3.12.7 was used due to its compatibility with the required libraries and frameworks.

The first step involved creating a dedicated virtual environment within Anaconda to isolate dependencies and avoid conflicts with existing installations. Once activated, the necessary dependencies were installed using conda and pip. Key libraries included PyTorch for model training, Ultralytics for YOLO implementations, and supporting tools such as Optuna. Installing PyTorch required GPU compatibility as the NVIDIA CUDA Toolkit v12.1 was used to enable GPU acceleration. The Ultralytics package was also installed via PIP. The Anaconda environment was tested to ensure all dependencies were correctly installed and functioning. Sample scripts were run to validate GPU availability and confirm that PyTorch and YOLO were correctly configured.

With the environment prepared, the training phase began by initialising the YOLO11 models. Each model variant was trained separately from scratch using the unique characteristics of the BirDrone dataset. Hyperparameter tuning was conducted in the validation phase using Optuna.

With so many models being trained for performance comparison, all with automatically generated sub-folders, it became necessary to create a structured folder. The naming convention chosen for models was 'abbreviated type of training_model_reverse date_time'. For example, a training script of the YOLO11m model on October 21st 2024, 10:10 am would be 'train_yolo11m_20241021_1010'. Folders were split into the structure shown in Figure 10, allowing for a structure that could be easily recalled later. For optimisation runs, which will be discussed later in Hyperparameters on Page 35, each trial within a run required its own folder and naming convention which was given as 'trial_trial

number_time', also shown in Figure 10. As this folder each had 50 trials, the trial number and time was important for later performance comparisons.

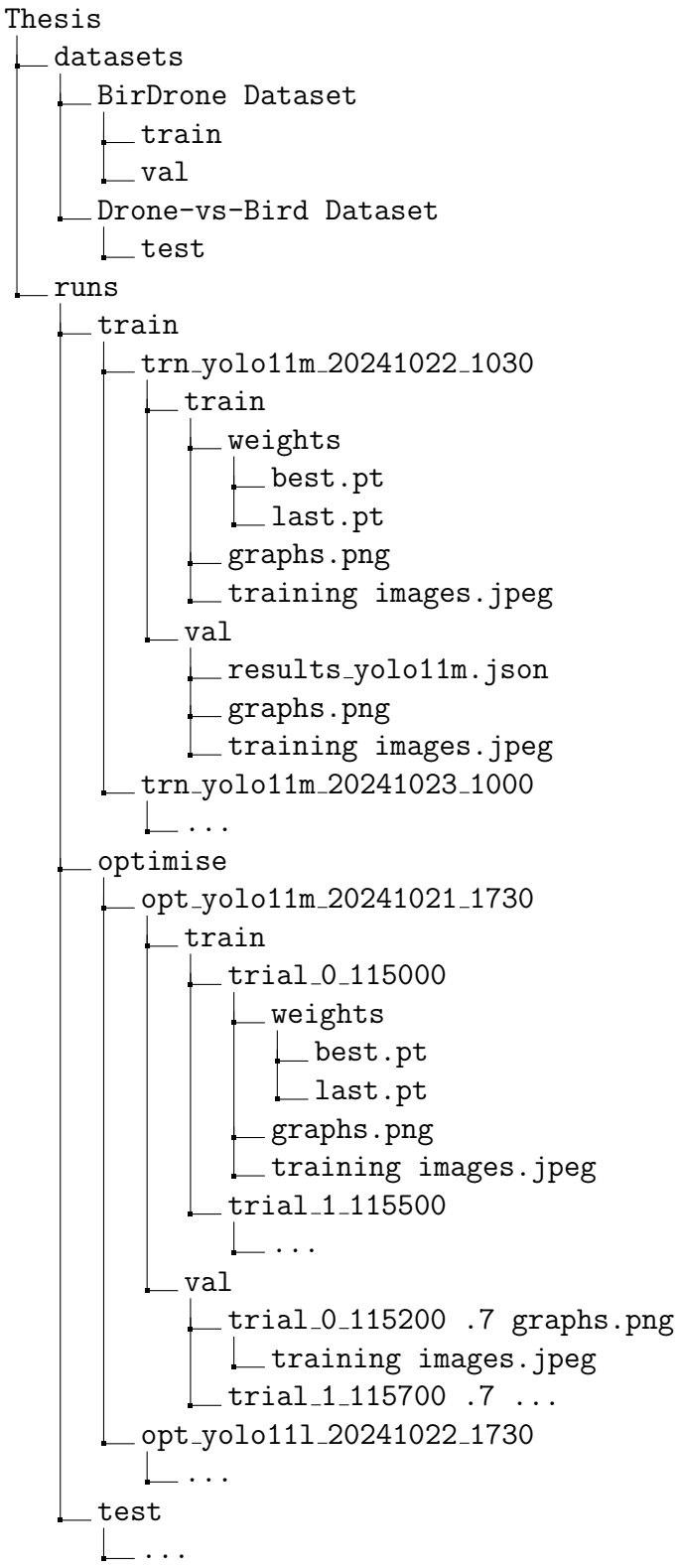


Figure 10: Example Folder Directory Structure

Monitoring and Evaluation

Throughout the training, the command line remained open showing real-time progress information, with mAP50 reported after every trial. Figure 11 shows an example of the command line interface, showing the initialisation of the folder and folder structure, parameters, and current epoch number with completion, ETA and iteration per second. This particular example only had a batch number of 8 correlating to the low GPU memory of 4.39GB. A batch size of 16 would more than double the GPU memory at around 10.4GB, increasing efficiency. Regular checkpoints were saved to allow for recovery and intermediate evaluations of the data.

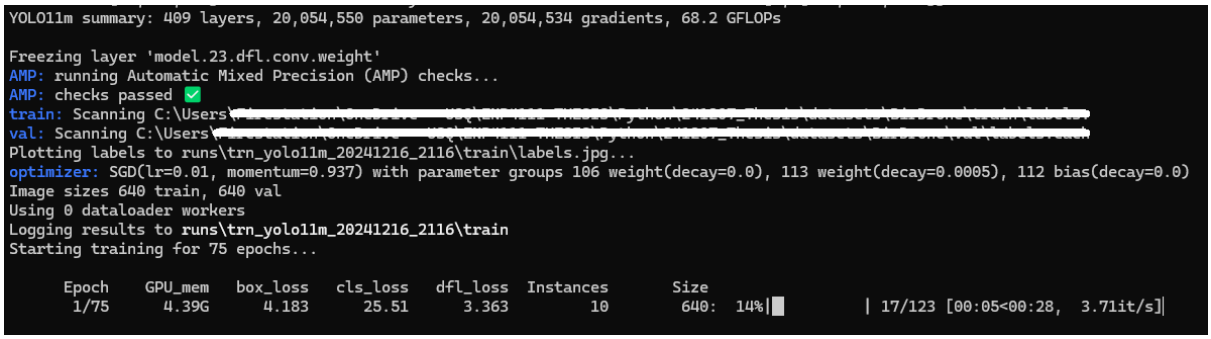


Figure 11: Anaconda Command Line - YOLO11m Example

Model Validation

Validation evaluates the model's performance during training on a separate dataset (validation set) that the model has not seen before. This helps fine-tune hyperparameters (e.g. learning rate, batch size, epochs) and assess whether the model is overfitting or underfitting.

Validation of the YOLO11 and YOLO8 models was conducted using the tools provided by YOLO itself, specifically the 'val' mode. This mode facilitates a structured and efficient evaluation of model performance against the validation dataset, which was curated separately from the training data to ensure an unbiased assessment. The validation dataset of BirDrone was designed to simulate real-world conditions, encompassing varied lighting scenarios, altitudes, and backgrounds to challenge the model's robustness.

The validation process began by ensuring consistency between training and validation configurations. The image preprocessing techniques applied during training. This alignment minimised discrepancies between the two phases and ensured that the performance metrics reflected the model's real-world deployment capabilities accurately.

Validation was executed using the model.val() function in the YOLO python library. This command automated the evaluation pipeline, comparing the model's predictions against ground truth annotations. Precision, recall, F1 score, mAP, and inference time were calculated directly, providing a comprehensive view of the model's detection and localisation capabilities. Metrics such as mAP were evaluated across multiple IoU thresholds (e.g., mAP50, mAP50-95), reflecting the model's accuracy in detecting UASs with varying degrees of overlap with ground truth bounding boxes. mAP50 was the

primary metric here, with the graphs for PR, F1-score, performance results and confusion matrixes used for information.

Model validation was completed automatically after the training algorithm, but was stored in a separate folder labelled 'val'. Once all three of the initial models were trained for benchmarking, hyperparameter tuning below was conducted on the best performing model.

Hyperparameter Tuning

Hyperparameter tuning is a critical component of machine learning model development, as it directly influences model accuracy, convergence speed, and generalisation. Hyperparameters differ from model parameters in that they are set before training begins and control the learning process, affecting aspects such as learning rate, batch size, and model architecture. Effective hyperparameter tuning is essential for optimising the performance of machine learning models, particularly in complex systems like YOLO11.

Traditionally, hyperparameter tuning was done manually through methods such as grid search and random search. In grid search, a fixed range of hyperparameter values is predefined, and exhaustive combinations of these values are tested. Random search, on the other hand, samples values from the hyperparameter space at random. While simple and effective for small search spaces, these methods have significant drawbacks such as high computational cost, inefficient exploration and human intervention to set ranges, initiate trials and monitor results.

To overcome the limitations of manual methods, researchers developed automated HPO tools. These tools automate the process of selecting, testing, and refining hyperparameters, significantly reducing manual intervention. One such tool is Optuna, as discussed in the Literature Review, a modern framework for dynamic hyperparameter tuning. Unlike grid and random search, Optuna uses a Tree-Structured Parzen Estimator to model the hyperparameter space and prioritize more promising regions. This approach enables more efficient exploration while converging to optimal hyperparameter values faster.

This method operates in the following stages:

- Initial Allocation: All trials start with an equal allocation of computational resources (e.g., number of epochs).
- Evaluation Checkpoints: At specific checkpoints, trials are ranked by their performance.
- Elimination of Underperformers: The lowest-performing trials are 'pruned' or terminated (in practice this did not end up working effectively with the YOLO model, but was included here for information anyway)
- Resource Reallocation: Resources are reallocated to higher-performing trials, allowing them to continue with increased computational resources.

The Optuna trials took up a majority of this experiment, as 50 initial trials were completed using the below hyperparameters. These trials then allowed a tightened set of

hyperparameters for a further 10 trials, but this will be discussed in the Results section. The below parameters were chosen based on known-good parameters from previous YOLO models pre-trained on the COCO dataset.

The following parameters were evaluated:

1. **Learning Rate** Controls how much the model's weights are adjusted in response to the calculated error during training. It dictates the size of the steps the optimisation algorithm takes toward minimising the loss function. If the learning rate is too high the model might overshoot the optimal solution, leading to unstable training. If it is too low training becomes very slow and might not converge within a reasonable number of epochs. The benchmarking value for the learning rate was 0.01, with values between 0.01 and 0.0001 tested with Optuna in the 50 trials.

2. **Batch Size** The number of training examples processed together in one forward and backward pass through the model. Small batch sizes use less memory and introduce noise that can help generalisation but might make training less stable, where large sizes provide smoother and more stable gradient updates but require more memory and computational resources. It was found that batch size had the largest performance impact in training, and significant time was spent fine tuning this for the Desktop Computer.

   The batch size was set to 16 for the initial benchmarking, and 8 to 16 in steps of 4 was chosen for the Optuna testing. Larger batch sizes (greater than 16) caused memory overflow on the GPUs during training of the larger models (YOLO11l and YOLO11x), while smaller batch sizes slowed training and introduced noise in gradient updates.

3. **Number of Epochs** An epoch is one complete pass through the entire training dataset. In each epoch, the model sees all the training dataset samples once. Too few epochs might result in underfitting, where the model has not learned enough from the data, and too many epochs can lead to overfitting, where the model learns the noise in the training data, reducing its ability to generalize to new data.

   Initial benchmark training was conducted for 75 epochs, with Optuna trials conducted between 75 and 150 epochs, as initial benchmarking showed 75 epochs may not be adequate. This strategy prevented overfitting and ensured efficient use of computational resources.

4. **Momentum** Momentum is a hyperparameter that influences the update of model weights by incorporating the prior direction of gradient descent. It helps the model maintain its direction of movement, smoothing out fluctuations in weight updates and accelerating convergence. Higher momentum values encourage larger updates, while lower values slow the progression of training.

   Initial benchmarking momentum was chosen as 0.937 (the YOLO11 default setting), with Optuna testing between 0.9 to 0.96.

5. **Weight Decay** Weight decay is a regularisation technique used to prevent overfitting by penalising large weight values in the model. By applying an additional penalty to the loss function, weight decay encourages smaller, more generalizable

weights. This helps avoid models that 'memorize' the training data instead of generalizing to unseen data.

Initial benchmarking value for weight decay was 5e−4, with values between 1e−3 and 1e−6 chosen for Optuna testing. Larger values (e.g., 0.01) resulted in underfitting, while smaller values (e.g., 0.0001) provided minimal regularisation, leading to overfitting on the training data.

To initiate the optimisation, Optuna's create_study() function was used with the Successive Halving Pruner enabled. This pruner halved the number of active trials at each checkpoint, eliminating trials that failed to achieve promising results relative to other ongoing trials. The process ensured that only the most promising hyperparameter configurations were explored in depth, thereby significantly reducing computational time.

The objective function is central to the HPO process, defining the optimisation steps and driving the search for the best hyperparameters. The objective function trained the YOLO11 model and returned the mAP as the evaluation metric for each trial, where a higher mAP indicated better model performance. The process began with hyperparameter sampling, where Optuna randomly sampled values (within the set limits) for learning rate, batch size, number of epochs, momentum, and weight decay from predefined search spaces. Next, the model training phase was started, where the YOLO11 model was configured using a YAML file and trained using the sampled hyperparameters. The model was, as also done in benchmarking, trained on the BirDrone dataset using Stochastic Gradient Descent (SGD) as the optimiser, with the model weights saved in a unique directory for each trial, enabling post-training validation and figure plotting. Once training finished, the model's last.pt file, which represents the final weights, was loaded for validation and evaluation. The model was then evaluated on the BirDrone validation dataset, with mAP50 extracted as the key performance metric. If the last.pt file was missing due to training failure, a default mAP of 0.0 was assigned to ensure trial continuity. Finally, the mAP value was returned to Optuna, which used this feedback to refine its hyperparameter sampling strategy for subsequent trials, allowing the optimisation process to progressively improve over time.

This process was then conducted for 50 trials, refining and optimising the sampled values to improve mAP over time. The optimisation process was conducted for the chosen YOLO model only. The best-performing hyperparameter set was extracted and saved for further fine-tuning.

Tightened Hyperparameter Tuning

After the 50 trials had been completed, it was noticed that the breadth of the hyperparameters meant that a conclusive could not be reached. A futher 10 trials with significantly tightened and optimised hyperparameters were conducted following the same method as above, and this allowed for a higher mAP value to be reached.

Model Testing

The final phase of the algorithm development involved testing the Optuna optimised trained and validated YOLO11 model to evaluate its performance in realistic scenarios.

Unlike benchmarking and optimisation, which focus on optimising and fine-tuning the model, testing serves as an objective evaluation of the model's ability to generalise to new data. This stage was crucial for verifying whether the optimised model could reliably detect UAS in diverse and operationally challenging conditions representative of their intended deployment.

Deploying the optimised YOLO11 model marked the milestone of validating the model in a controlled testing to real-world application. This phase focused on processing video feeds and detecting UAS in dynamic environments. The deployment process was designed to reflect the practical conditions under which the model would operate, testing for readiness in real-world scenarios.

The system processed five video streams provided by the Drone-vs-Bird dataset (Coluccia et al. 2023). As discussed, this dataset was specifically chosen for it's relevance and known difficulty. Live testing of the model involved streaming the five video's separately and observing its ability to detect and track UASs in real time. A variety of scenarios were tested, including different lighting conditions, varied UAS altitudes, clear and occluded backgrounds, single and clusters of UAS, untrained objects such as planes, and backgrounds with potential visual distractions. The model was evaluated on its ability to maintain accurate and consistent detections under these conditions.

To evaluate performance during live testing, identification, tracking and confidence levels were measured. Detections were compared against known ground truth annotations derived from manually reviewed footage or controlled test cases. Inference times were monitored to ensure the system consistently met the real-time requirement of processing frames faster than the input footage, which was 60FPS. Special attention was given to edge cases, such as low-contrast environments or partially obscured UASs, as these represent critical scenarios in practical applications.

This phase of the project ensured that the models were tested under realistic conditions and could operate effectively in their intended environment. The deployment process provided valuable insights into the practical performance of the detection system, including areas where further refinement might improve its reliability or scalability. By combining algorithm deployment with live testing, this methodology bridged the gap between controlled experimentation and real-world use, ensuring the system was ready for practical application.

RESULTS

Benchmarking

The benchmarking models were trained under the same hyperparameters, all trained on the Laptop computer while the Desktop computer worked on the optimisation models.

The benchmarking dataset use the below hyperparameters:

- Epochs: 75
- Batch: 8 (to accommodate for GPU memory errors during YOLO11x testing)
- Image Size: 640
- Optimiser: SGD
- IOU: 0.7
- Initial Learning Rate: 0.01
- Final Learning Rate: 0.01
- Momentum: 0.937
- Weight Decay: 5e−4

All results from modelling exported with a CSV file, a compilation of the training batch and validation batch images with bounding boxes, F1-confidence, confusion matrices, and PR curves. The above parameters are shown in the python code used for training the YOLO11 model in Appendix A.

It should be noted that in all testing, the classes were separated into Drones and Birds, with Class 0 being birds, and Class 1 being Drones. As the dataset was the same for all training, the dataset training image exports were identical. This is shown in Figures 12, 13, 14 and 15 which show that the training continues to 7997 batches. The bounding boxes are pre-defined, as shown with dark blue boxes for Class 0 Birds and cyan boxes for Class 1 Drone. It also shows how the augmentations are applied to the images for a more robust training dataset, with various overlaps, rotations, scaling changes and distortions within the images.

Figure 12: Training Batch 00, with bounding boxes

Figure 14: Training Batch 7995, with bounding boxes

Figure 13: Training Batch 02, with bounding boxes

Figure 15: Training Batch 7997, with bounding boxes

YOLO11m

The YOLO11m model demonstrated average performance in detecting UASs, with an overall mAP of 0.2847, which is considered low. While the model shows an ability to identify UASs in various scenarios, the results show clear limitations, particularly in precision and recall for the Bird class.

The F1-Confidence Curve in Figure 16 shows a peak F1 score of 0.33 at a confidence threshold of 0.233, indicating the model struggles to maintain a consistent balance between precision and recall. The Drone class achieves higher F1 scores across confidence values, but the Bird class underperforms significantly, with scores remaining below 0.2 for most thresholds. This suggests the model has difficulty distinguishing between birds and background, leading to frequent misclassification.

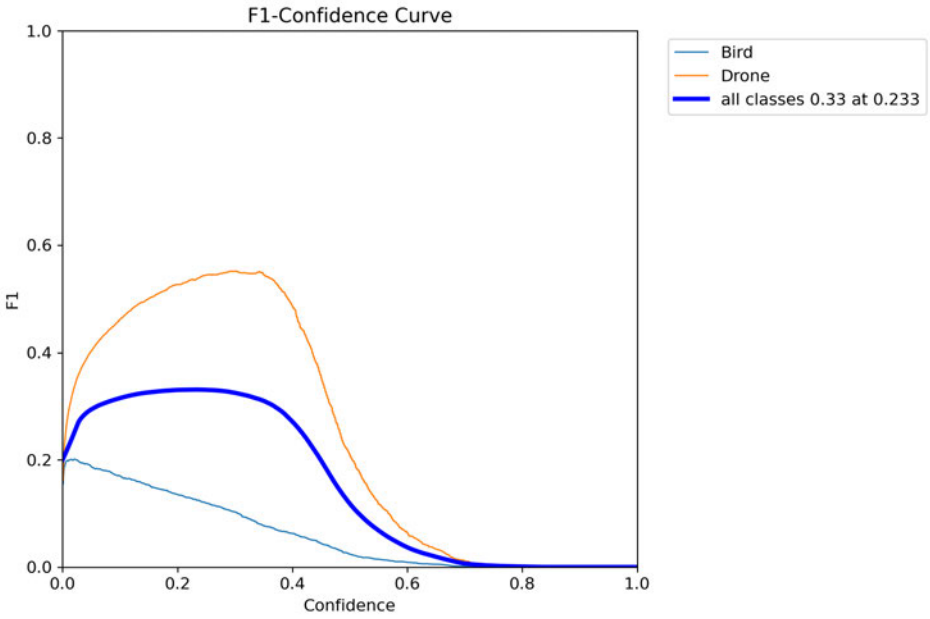


Figure 16: YOLO11m Benchmark - F1 Curve

The PR Curve in Figure 17 further shows the model's strengths and weaknesses. For the Drone class, precision reached 0.445, showing the model can identify UASs with moderate confidence. However, precision declines steeply as recall increases, suggesting that the model sacrifices accuracy when attempting to detect more objects. The Bird class again proves challenging with a low precision of 0.121. This indicates a higher rate of false positives, which reduces the model's reliability when distinguishing birds from irrelevant features in the images. The overall PR curve for all classes reflects the model's limitations in achieving consistent performance across different detection thresholds.

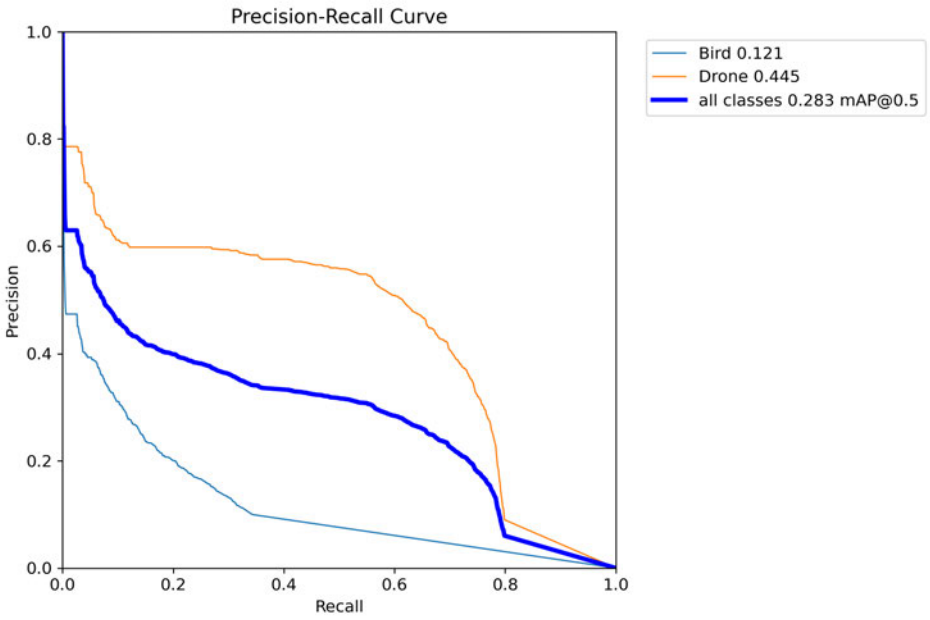


Figure 17: YOLO11m Benchmark - PR Curve

The confusion matrix shows the YOLO11m model's detailed performance across classes. For the Drone class, the model correctly identifies 1731 true positives, but a notable number of drones are misclassified as background or other objects. Similarly, birds are frequently misidentified, with a significant proportion of true Bird labels being classified as background or drones. This further supports the observation that the model struggles with the Bird class, resulting in lower precision and recall for this category. Additionally, the background class shows a high number of correct classifications but also contains a non-negligible number of false positives.

The training and validation losses (results in Figure 19) provide additional insights into the model's performance. While the box loss, classification loss, and distribution focal loss show steady downward trends during training, the loss values remain relatively high compared to an ideal model. This suggests that YOLO11m struggles to fully converge, particularly for more complex or ambiguous objects. Precision and recall metrics improve steadily over training epochs but exhibit significant fluctuation, highlighting instability in the learning process. The final metrics indicate that while the model makes progress during training, there is room for further optimisation.

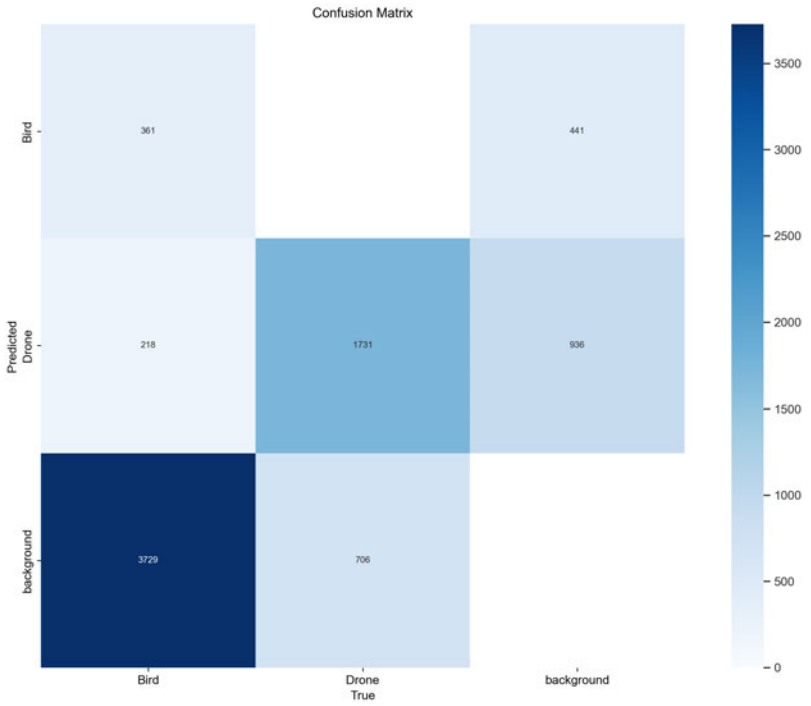


Figure 18: YOLO11m Benchmark - Confusion Matrix

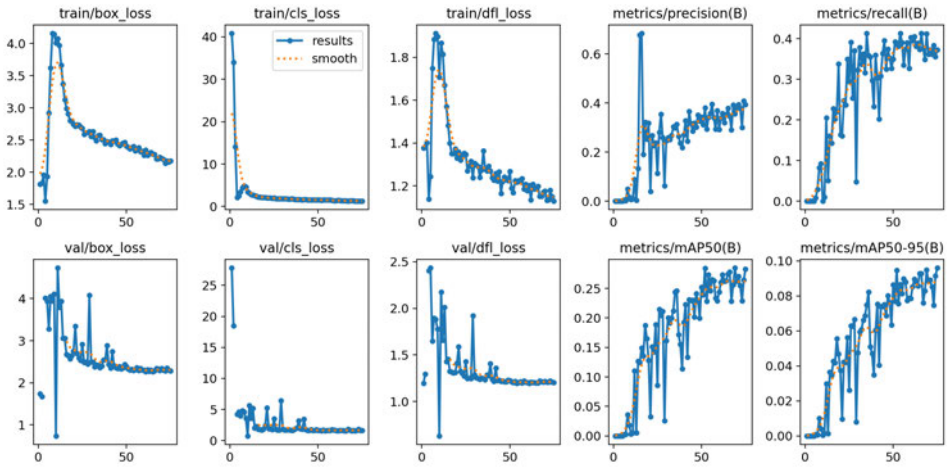


Figure 19: YOLO11m Benchmark - Results

Figure 20 displays a comparison between the model-predicted bounding boxes and the fully annotated training image (the Truth image) utilised for validation. Although the model does identify the UAS within the scene, it shows limited confidence in its detection. Given that the confidence threshold is set at 0.5, the UAS's presence is acknowledged but not assigned a decisive label. This suggests that while the YOLO11m models fundamental object recognition ability is in place, further training or parameter tuning might be required to improve its confidence in consistently classifying targets.

(a) Truth (b) Predicted

Figure 20: YOLO11m comparison of truth against predicted images

Overall, the YOLO11m model establishes a moderate baseline for UAS detection with questionable success. However, the performance for the Bird class remains poor, with frequent misclassifications and low precision. The high loss values and fluctuating training metrics suggest that the model requires further optimisation to achieve greater stability and accuracy. These results highlight the need for targeted improvements, particularly in class-specific detection and confidence calibration, to enhance the model's overall performance.

YOLO11l

The python file for training this model can be seen in Appendix A. The results of the YOLO11l model demonstrates a notable improvement in performance compared to the previous YOLO11m model. The mAP50 achieved by YOLO11l was 0.346, reflecting a significant increase of approximately 21.5% over the 0.2847 recorded for YOLO11m. This improvement shows the capability of the YOLO11l model to detect and classify objects, particularly UASs, with greater accuracy. The F1-Confidence Curve shown in Figure 21 further supports this, with YOLO11l achieving a peak F1 score of 0.38 at a confidence threshold of 0.316, compared to YOLO11m's lower peak of 0.33. While the Drone class demonstrated consistently higher F1 scores, the Bird class continued to underperform, with its F1 score remaining below 0.25 across most thresholds.

An analysis of the training and validation losses further reinforces these findings. Figure 22 shows that YOLO11l demonstrated faster convergence and greater stability compared to YOLO11m, with reduced box loss, classification loss, and distribution focal loss throughout the training process. The metrics for precision and recall displayed a clear upward trajectory over the training epochs, confirming improved generalisation performance. This consistency across training and validation metrics suggests that the YOLO11l model benefits from better optimisation and more efficient learning.

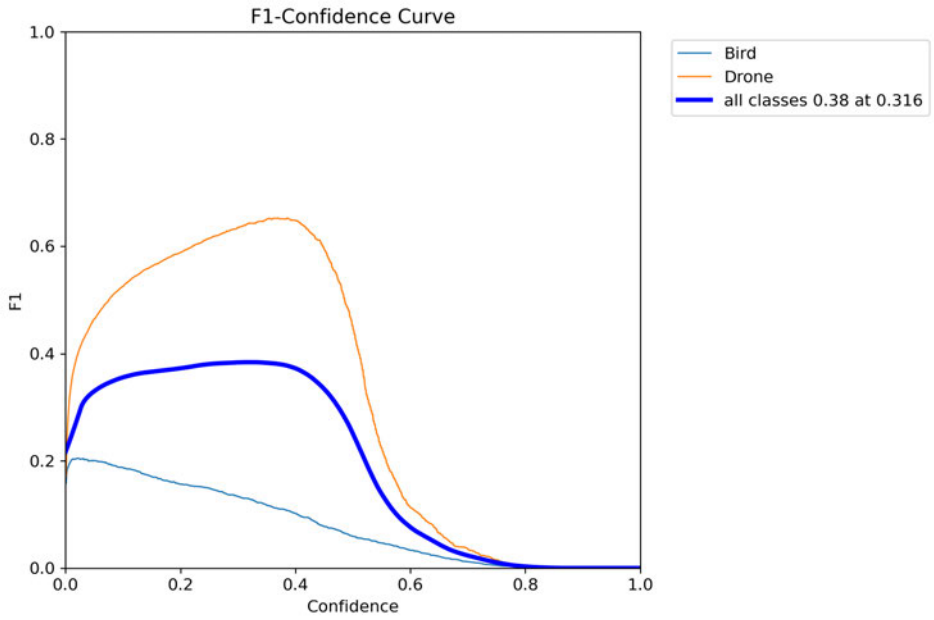


Figure 21: YOLO11l Benchmark - F1 Curve

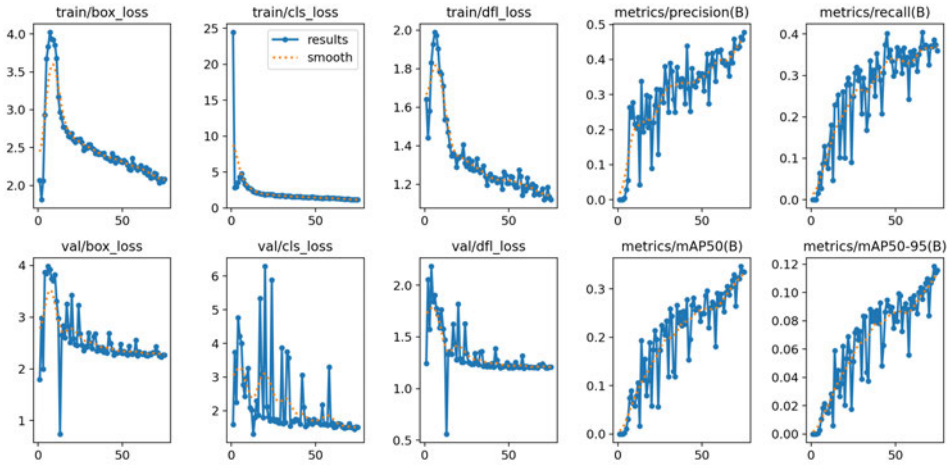


Figure 22: YOLO11l Benchmark - Results

The Precision-Recall Curve in Figure 23 shows clear advancements, particularly for the Drone class, which achieved a precision of 0.569 in YOLO11l, up from 0.445 in YOLO11m. The YOLO11l model maintains a higher precision across varying recall values, showing improved ability to reduce false positives for UAS. In contrast, the Bird class achieved a precision of only 0.124, showing birds are more difficult to detect that UAS. The overall PR balance has improved, showcasing the YOLO11l model's robustness and better trade-off between precision and recall.

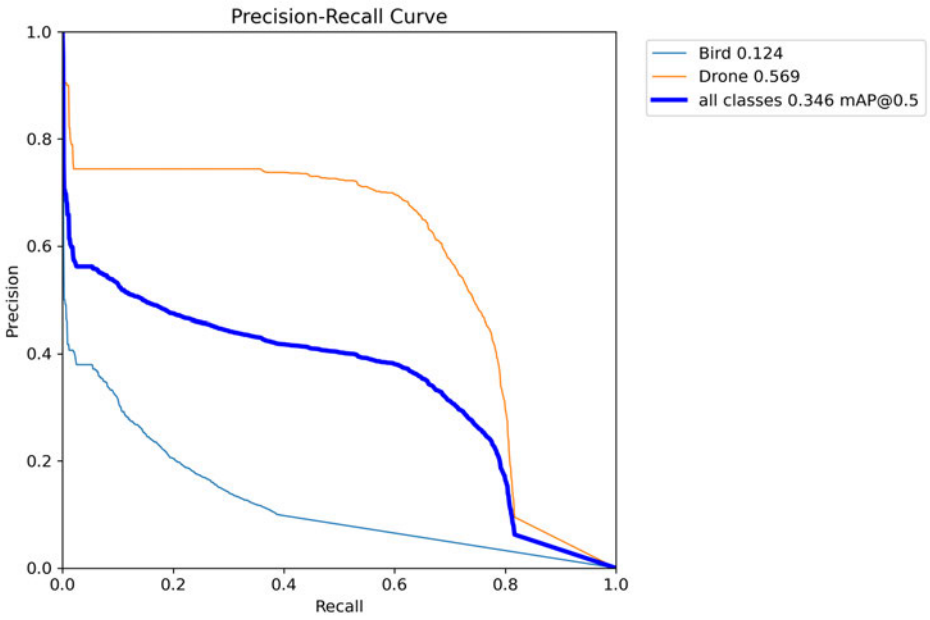


Figure 23: YOLO11l Benchmark - PR Curve

The confusion matrix in Figure 24 shows the advancements in true positive classifications for the Drone class. YOLO11l correctly identified 1921 drones, an improvement over the 1731 recorded by YOLO11m. Additionally, there were fewer misclassifications into the background category, indicating that the model is better at distinguishing UASs from irrelevant objects. Again, the bird class is shown as difficult to detect.

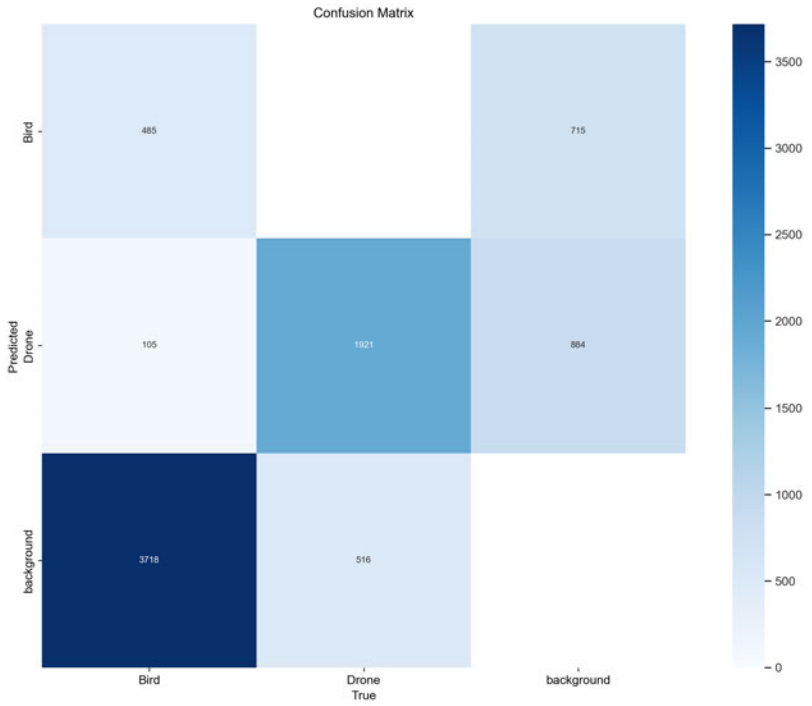
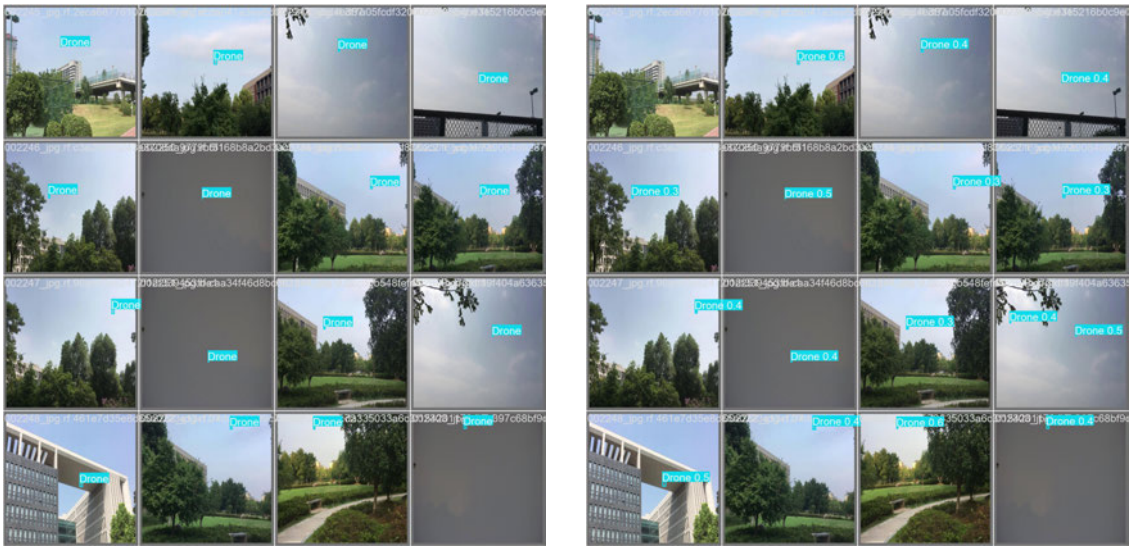


Figure 24: YOLO11l Benchmark - Confusion Matrix

Figure 25 shows again the truth and predicted images for the YOLO11l model, showing

it successfully detects most UASs with predicted boxes closely aligning with the ground truth labels. However, confidence scores vary, ranging from 0.3 to 0.5, which indicates uncertainty in some predictions. This suggests that while the model can identify UASs effectively, it still struggles with consistency in confidence, particularly in more challenging or ambiguous scenes.



(a) Truth (b) Predicted

Figure 25: YOLO11l comparison of truth against predicted images

The YOLO11l model demonstrates clear improvements over YOLO11m, particularly in its ability to detect UASs with higher precision and recall. The increase in mAP, coupled with the more stable training metrics and better classification performance, suggests that YOLO11l is a more robust and accurate model.

YOLO11x

The YOLO11x model demonstrates slight improvements over the YOLO11l model, particularly in terms of precision and mAP. The mAP50 for YOLO11x is 0.342, which is slightly lower than the 0.346 achieved by YOLO11l. While the difference in overall mAP is marginal, the PR analysis and confusion matrix indicate noteworthy differences in class-specific performance.

The F1-Confidence Curve in Figure 26 shows that the YOLO11x model achieves a peak F1 score of 0.37 at a confidence threshold of 0.288, which is slightly lower than YOLO11l's peak F1 score of 0.38. However, YOLO11x maintains a more gradual decline in F1 scores for the UAS class, indicating more consistent performance across a range of confidence thresholds. The Bird class continues to struggle, with F1 scores remaining low, similar to YOLO11l, but the overall curve for both classes suggests a slight reduction in false positives at higher confidence thresholds.

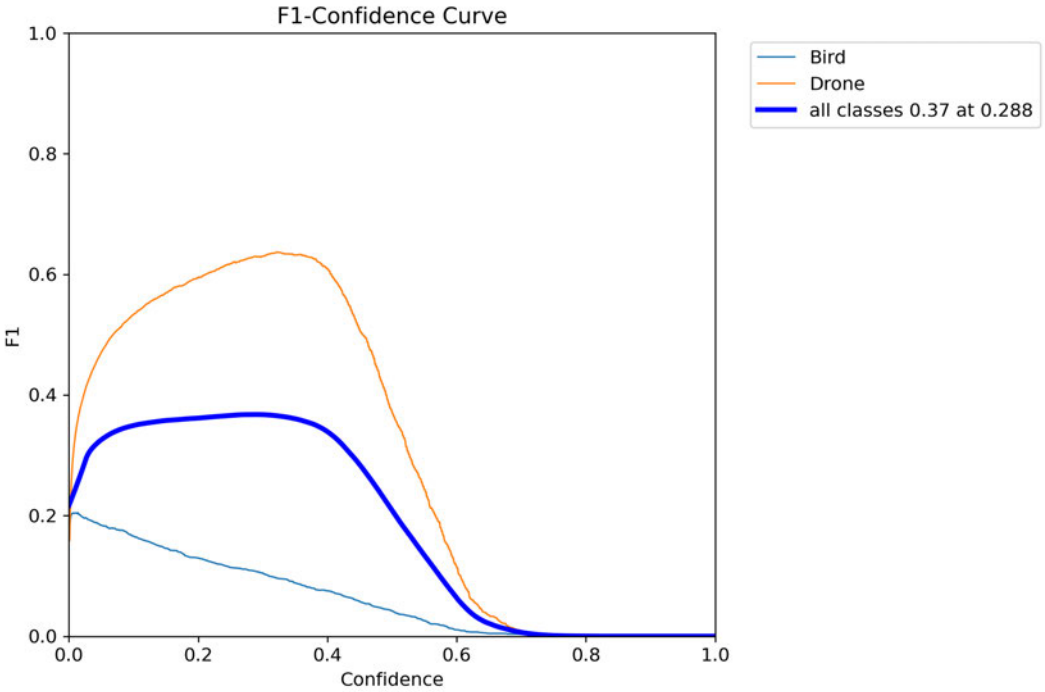


Figure 26: YOLO11x Benchmark - F1 Curve

The Precision-Recall (PR) curve reveals that precision for the Drone class in YOLO11x is 0.555, slightly lower than YOLO11l's 0.569. However, YOLO11x demonstrates improved precision for the Bird class, with a precision of 0.126, compared to YOLO11l's 0.124. While the improvement is minor, it indicates that YOLO11x performs slightly better in distinguishing birds from background noise. The overall PR curve for all classes in YOLO11x appears smoother and shows a more stable precision rate at moderate recall levels, which suggests better consistency in predictions.

The confusion matrix in Figure 24 shows the comparative performance of YOLO11x. For the Drone class, YOLO11x correctly identifies 1784 true positives, slightly lower than the 1921 detected by YOLO11l. However, YOLO11x shows a reduction in false positives for UASs, with fewer misclassifications into the background class. Additionally, YOLO11x improves slightly in identifying birds, with fewer Bird labels misclassified as background compared to YOLO11l. These results indicate that YOLO11x, while not achieving a higher overall mAP, performs more consistently across both classes and reduces specific types of errors.

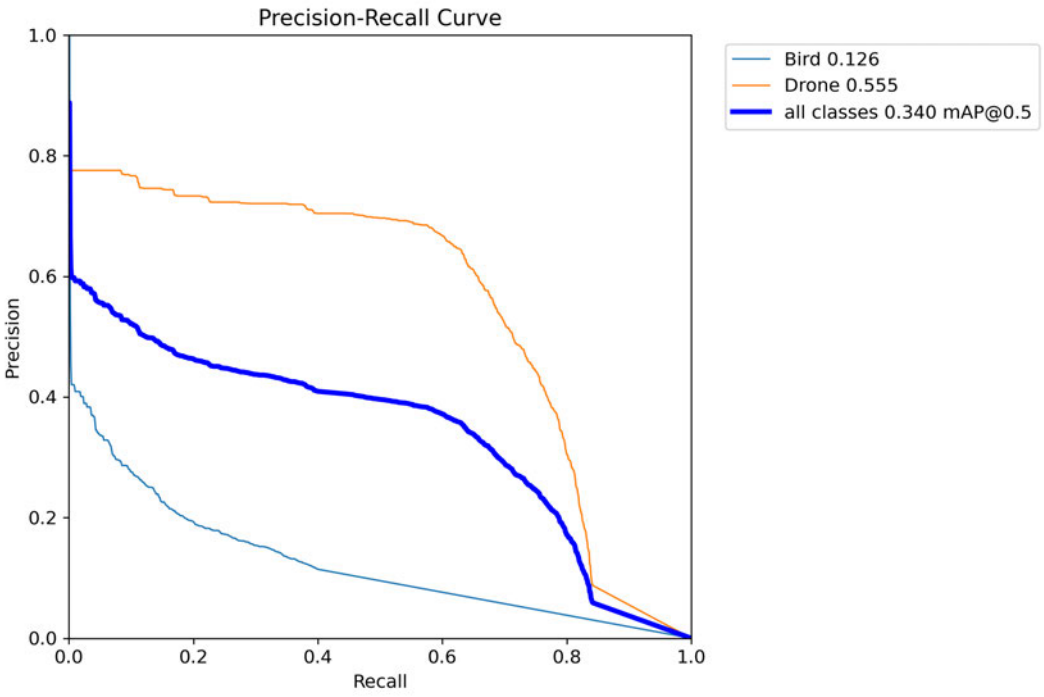


Figure 27: YOLO11x Benchmark - PR Curve

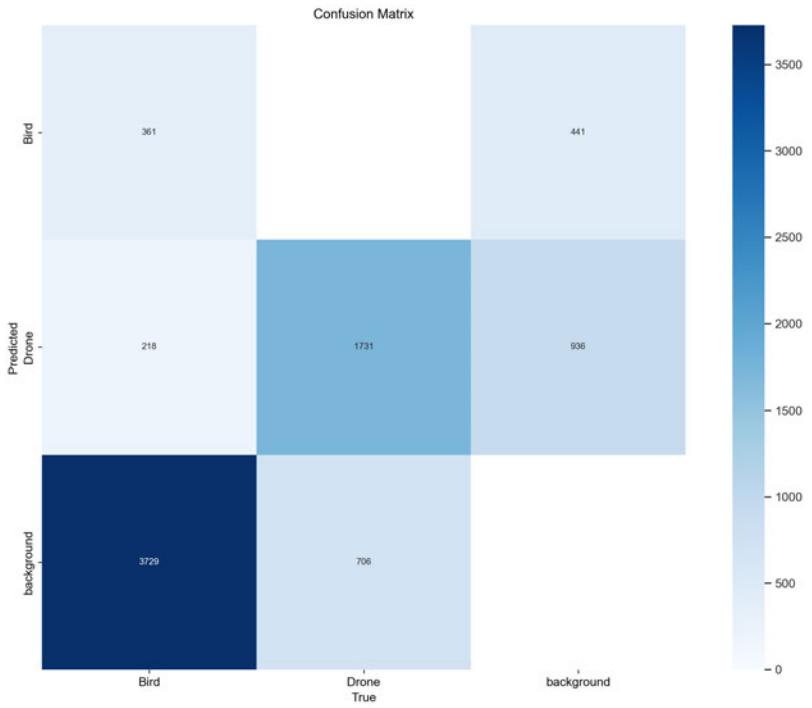


Figure 28: YOLO11x Benchmark - Confusion Matrix

The training and validation loss curves in Figure 29 for YOLO11x show improved stability during training compared to YOLO11l. The box loss, classification loss, and distribution focal loss decrease more smoothly, with fewer fluctuations, indicating better convergence. Precision and recall metrics for YOLO11x also show steady improvement across epochs,

similar to YOLO11l, but with slightly less variability, particularly in recall. This stability is likely a result of the larger model size and greater parameter capacity, which allows YOLO11x to learn more nuanced features in the dataset.

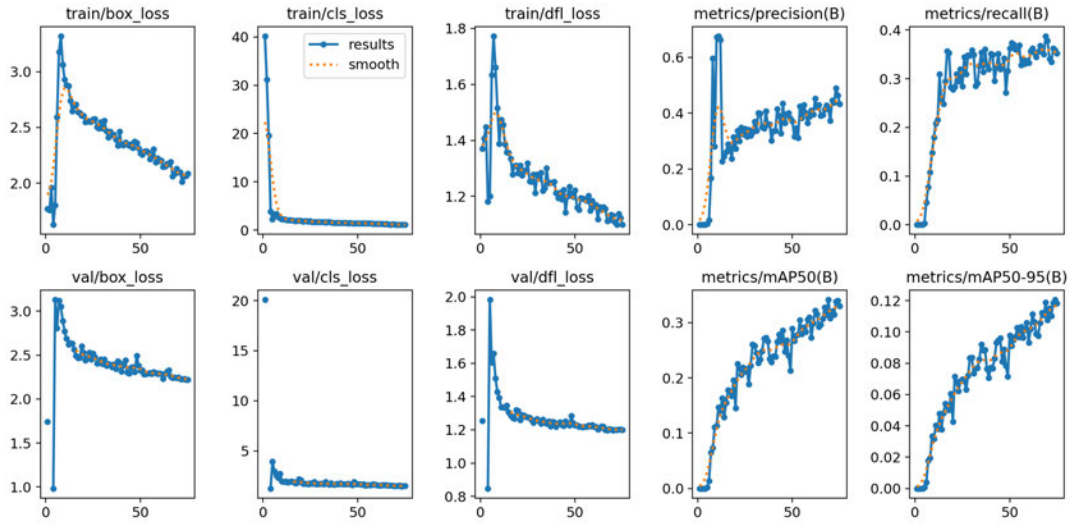


Figure 29: YOLO11x Benchmark - Results

Figure 30 again shows the predicted againt truth images. When compared to the corresponding YOLO11m and YOLO11l predictions, the YOLO11x model's detected bounding boxes appear more consistently placed over the actual UAS locations. However, while YOLO11x might show slightly improved alignment, both models still demonstrate moderate confidence scores rather than strong certainty. YOLO11x predictions are more accurate in pinpointing UASs than YOLO11m, but both models retain a measure of uncertainty that could benefit from further refinement or additional training data.

(a) Truth (b) Predicted

Figure 30: YOLO11x comparison of truth against predicted images

While YOLO11x achieves slightly lower mAP compared to YOLO11l, it demonstrates

improved stability during training and more consistent performance across precision and recall metrics. The model reduces false positives for UASs and shows minor improvements in bird detection, addressing some of the shortcomings observed in YOLO11l, although it was overall worse at detecting UAS than YOLO11l. These results suggest that YOLO11x offers a more balanced and reliable detection model, particularly for challenging or ambiguous scenarios, despite the marginal trade-off in overall mAP.

It should also be noted that the training time for YOLO11x was significantly longer, taking 23,722 seconds (6.6 hours) to complete. In comparison, YOLO11l required only 2.12 hours, and YOLO11m completed training in 1.76 hours. This shows a trade-off between performance and computational cost, with YOLO11x demanding far more training time for only marginal improvements.

Benchmarking Conclusions

Based on the results of the YOLO11m, YOLO11l, and YOLO11x models, the YOLO11l model is the clear suitable choice for deployment. While the YOLO11x model offers slight improvements in stability and marginal reductions in false positives, these gains are accompanied by a significantly higher computational cost and worsened UAS detection rate. The training time for YOLO11x was 6.6 hours (23,722 seconds), which is over three times longer than YOLO11l's 2.12 hours, and nearly four times longer than YOLO11m. This substantial increase in computation time doesn't justify marginal gains in confidence while maintaining a lower mAP of 0.342, compared to the improved 0.346 for YOLO11l.

It is also important to note that the minor gaps in precision and stability seen in YOLO11l could likely be addressed through hyperparameter tuning. By optimising parameters it is reasonable to expect further improvements without the need for the increased computational resources demanded by YOLO11x. For these reaosns, YOLO11l was chosen as the model to be optimised in the following results below.

Optimised Hyperparameter Dataset

The next step was to optimise the hyperparameters of the chosen model (YOLO11l) and train a new model on the same BirDrone dataset. The training of this optimised model was extensive, taking over 2 days to complete all 50 trials. It should also be noted that the Desktop computer which was originally chosen for processing was too slow, estimating that the computations would have taken greater than 6 days to complete. As a result, the Google Cloud VM was used as the primary computations resource.

Please refer to the Methodology on Page 35 for the chosen hyperparameter windows of the below results. The hyperparameter tuning was done using the Optuna HPO framework, which made automating the tuning incredibly easy in comparison to manual methods. The Optuna Successive Halving Pruner was included in the code, but unfortunately did not work as the training for each trial completed in it's entirety before the pruner had a chance to work. This was unable to be rectified, and thus pruning was unsuccessful.

The results for the trials, in Table 2, show successful changes to the batch size, epochs, momentum, learning rate and weight decay, with a final value (mAP50) to quantify the

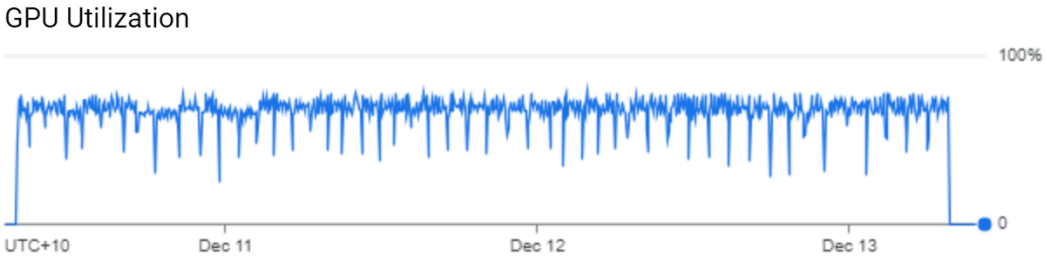
success of the experiment. The python file for training this model can be seen in Appendix B.

Trial	Batch Size	Epochs	Momentum	Learning Rate	Weight Decay	Value
21	16	102	0.9246	1.40E-03	7.90E-06	0.3770
22	16	116	0.9312	5.00E-04	4.68E-06	0.4522
23	16	114	0.9325	5.27E-04	6.24E-06	0.4176
24	12	125	0.9244	6.07E-04	1.56E-05	0.4334
25	16	134	0.9108	3.63E-04	1.57E-06	0.4589
26	16	133	0.9107	3.62E-04	1.12E-06	0.4611
27	16	146	0.9121	1.80E-04	1.19E-06	0.4356
28	12	134	0.9105	3.57E-04	1.46E-06	0.4490
29	16	150	0.9137	1.00E-04	1.07E-06	0.4466
30	12	87	0.9005	7.53E-04	9.07E-06	0.4054
31	16	134	0.9216	1.33E-03	2.51E-05	0.4236
32	16	102	0.9264	3.64E-04	4.99E-06	0.4347
33	16	113	0.9191	6.61E-04	4.18E-06	0.4504
34	16	79	0.958	2.47E-04	1.92E-06	0.4188
35	16	139	0.9107	2.09E-04	3.50E-06	0.4526
36	16	139	0.9093	2.04E-04	1.64E-06	0.4418
37	16	144	0.915	1.39E-04	3.72E-06	0.4324
38	12	131	0.9071	3.68E-04	2.80E-06	0.4495
39	16	137	0.9013	1.07E-03	2.29E-05	0.4305
40	12	134	0.9046	1.45E-04	1.08E-05	0.4377
41	16	142	0.9168	1.89E-04	6.91E-06	0.4396
42	16	125	0.9201	5.86E-04	4.66E-06	0.4460
43	16	104	0.933	4.09E-04	3.31E-06	0.4366
44	16	137	0.9074	2.33E-04	1.74E-06	0.4536
45	16	150	0.9075	2.45E-04	1.00E-06	0.4441
46	16	137	0.9119	1.25E-04	1.86E-06	0.4340
47	16	129	0.9043	2.12E-04	1.44E-06	0.4209
48	16	140	0.9155	3.01E-04	2.52E-06	0.4553
49	16	146	0.9072	7.91E-04	2.25E-06	0.4217
50	16	142	0.9142	2.99E-04	5.22E-05	0.4331

Table 2: Optuna Optimization Results - Top 30 Trials

Figure 31 was added here for information only as it provides interesting insights. Each dip in performance is a change into validation of trials, and where it peaks again is the beginning of a new trial for the full 50 trials. GPU utilisation remains fairly uniform, but GPU memory changes significantly. It is quite obvious in the GPU memory where batch size changes occur between trials, with 16 batches being the highest and 8 being the lowest, with a corresponding slight change in GPU utilisation. This shows that the 16 batches was significantly more efficient. Figure 32 also shows that 16 batches was more efficient. While the majority of the trials used a batch size of 16, this was due to

increased efficiency of GPU processing as determined by the Optuna HPO. Whilst an actual figure of performance improvement is difficult to obtain, say via processing time, is because the number of epochs also greatly affects the processing time and this was also changed between trials.



(a) GPU Utilisation

(b) GPU Memory Utilisation

Figure 31: GPU Utilisation and Memory Usage during Optuna trials

For all of the below figures where value is given on the x-axis, value is the mAP50 result from each individual trial. Table 2 itself doesn't how a patten, so the below figures were split into each individual testing hyperparameter against value to see any correlation.

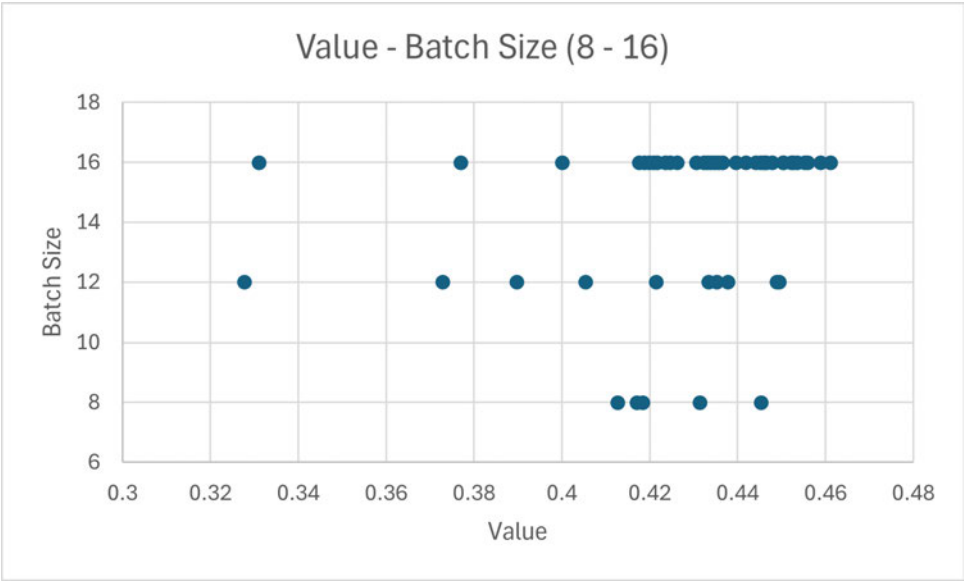


Figure 32: First Optuna Trials - Batch Size

The number of epochs was chosen to be tested in the range of 75 to 150, and the top 25 results from Figure 33 does indicate that epochs greater than 110 did have a significant increase in mAP50, with the best results being around 130 epochs. Due to this, it was decided the tightened testing parameter window for Optuna should be between 110 and 150 epochs.

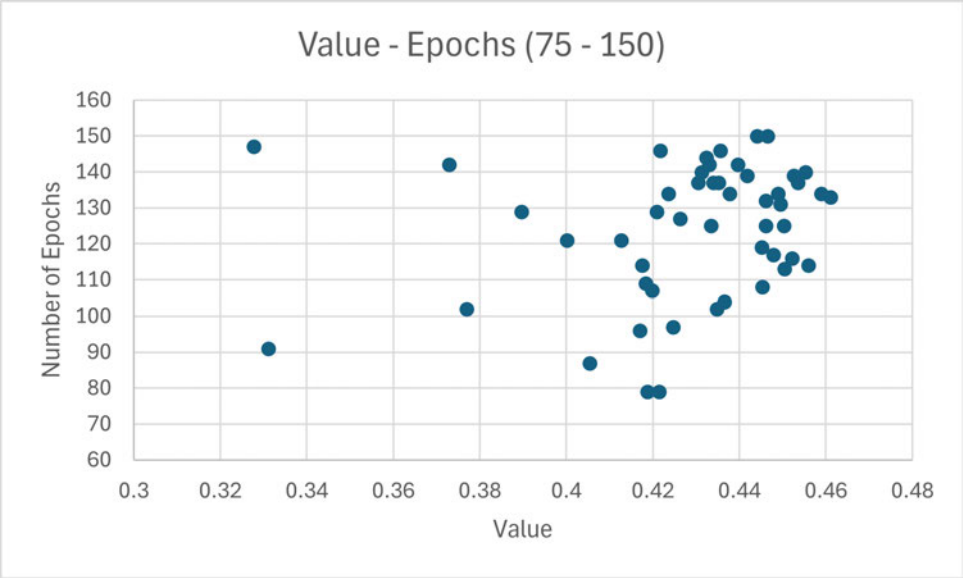


Figure 33: First Optuna Trials - Number of Epochs

The momentum shown in Figure 34 shows that as value increase it narrows in on a single point around 0.91. It was also noted using the data from Table 2, that lower number of epochs combined with lower learning rates and higher momentums (0.34) also provided respectable mAP results. Because of this the tightened testing parameter window for momentum were chosen as 0.91 to 0.935.

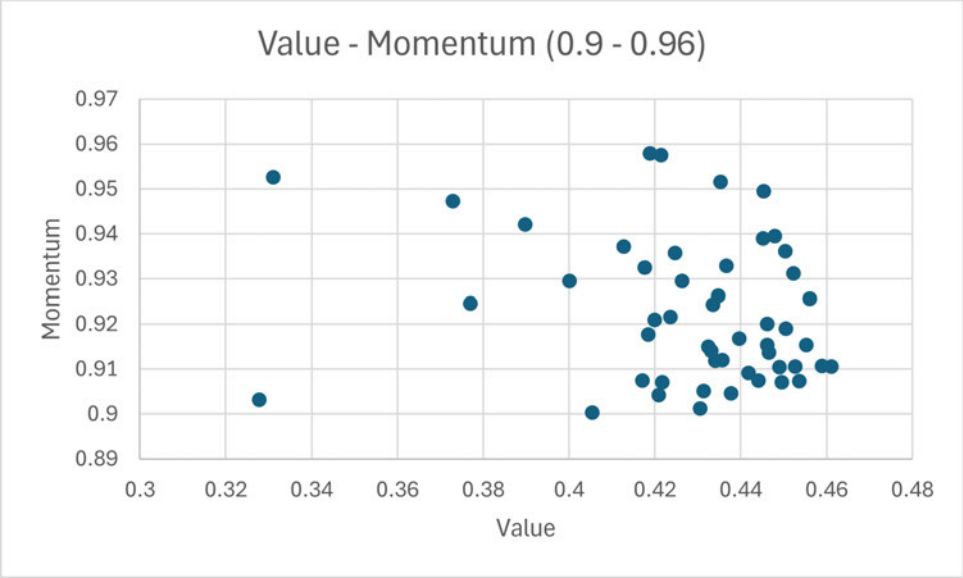


Figure 34: First Optuna Trials - Momentum

The learning rate in Figure 35 clearly shows that the top two learning rates, around 3.62e−4, had higher mAP than the other trials. This acknowledged, it was also noted that lower number of epochs combined with a higher learning rate, around 6e−4, also had impressive mAP results and this should be included in testing. Using this, tightened testing parameter window for learning rate was chosen as 2.5e−4 and 7e−4.

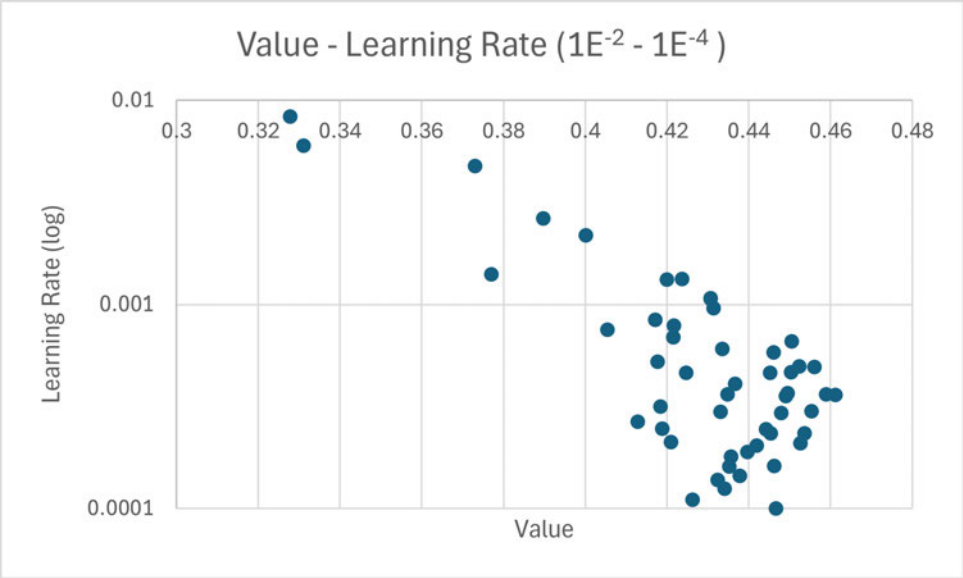


Figure 35: First Optuna Trials - Learning Rate

The final metric tested was weight decay, and Figure 36, along with Table 2, shows that weight decay below 5e−6 significantly outperformed higher weight decay, with lower weight decay producing better results. As a result, tightened testing parameter window for weight decay was chosen as 1e−6 to 4.7e−6.

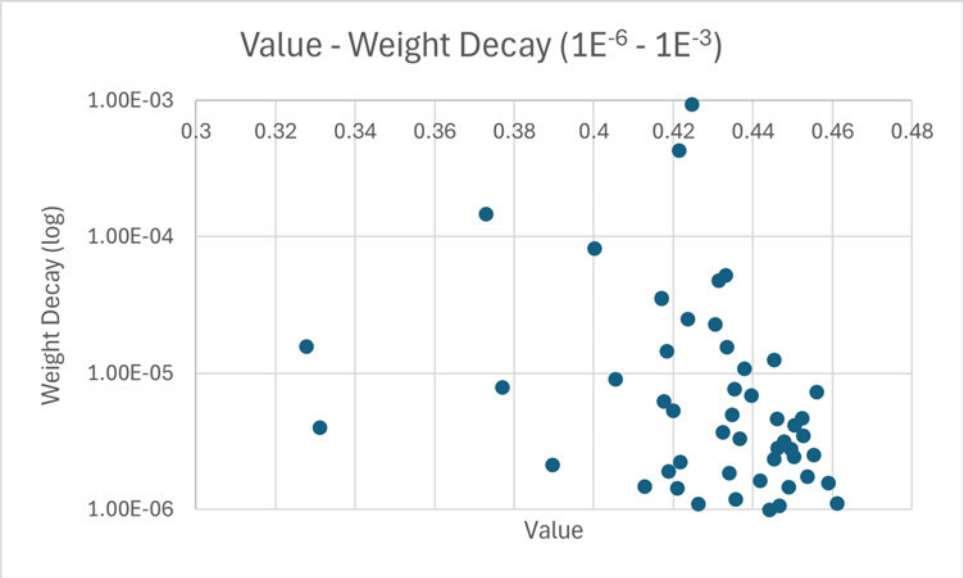


Figure 36: First Optuna Trials - Weight Decay

Optimised Hyperparameter Dataset - Tightened Parameters

With a tightened set of hyperparameters, another 10 trials were conducted using Optuna to try and get a better mAP value than 0.4611 as in Table 2.

Table 3 shows the final Optuna results, with a highest mAP of 0.46996 (0.47). Figures 37 through 40 really show how each individual parameters affects the performance of the model. The lowest score had high momentum, a high learning rate, low weight decay and the lowest epochs. The best result, known as 'Trial 8', had a low momentum, low learning rate, mid-range weight decay and surprisingly low epochs. It's also interesting to note the two best results had similar weight decay and momentum, but the second best trial had a higher learning rate and more epochs but was still unable to beat the lower learning rate. The python file for training this model can be seen in Appendix C.

Trial	Epochs	Momentum	Learning Rate	Weight Decay	mAP50
1	115	0.9325	4.93E-04	4.20E-06	0.44007
2	113	0.931	5.64E-04	1.48E-06	0.41703
3	133	0.9247	3.14E-04	2.03E-06	0.45817
4	132	0.9314	5.84E-04	3.26E-06	0.43187
5	115	0.9179	5.98E-04	2.72E-06	0.4458
6	114	0.9138	6.25E-04	3.08E-06	0.442
7	138	0.9264	6.19E-04	1.84E-06	0.45814
8	120	0.912	2.77E-04	2.71E-06	0.46996
9	125	0.912	4.18E-04	2.96E-06	0.46596
10	142	0.9117	3.92E-04	2.20E-06	0.45753

Table 3: Optuna Optimization Results (with tightened parameters)

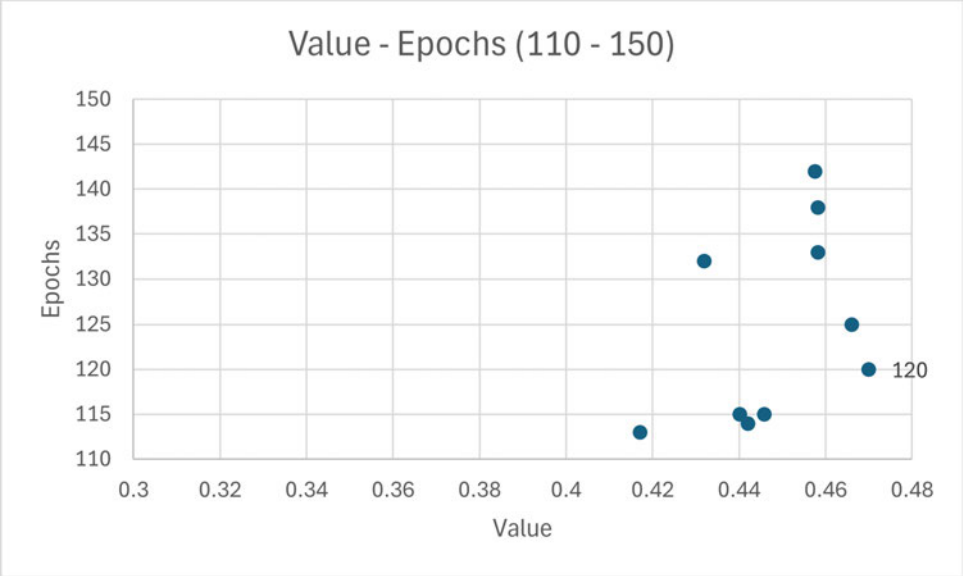


Figure 37: Second Optuna Trials - Number of Epochs

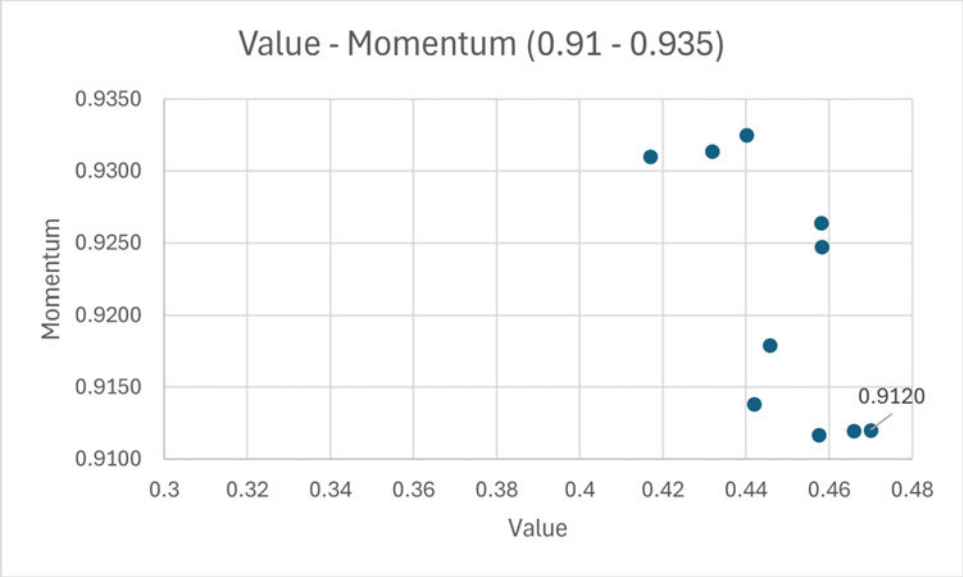


Figure 38: Second Optuna Trials - Momentum

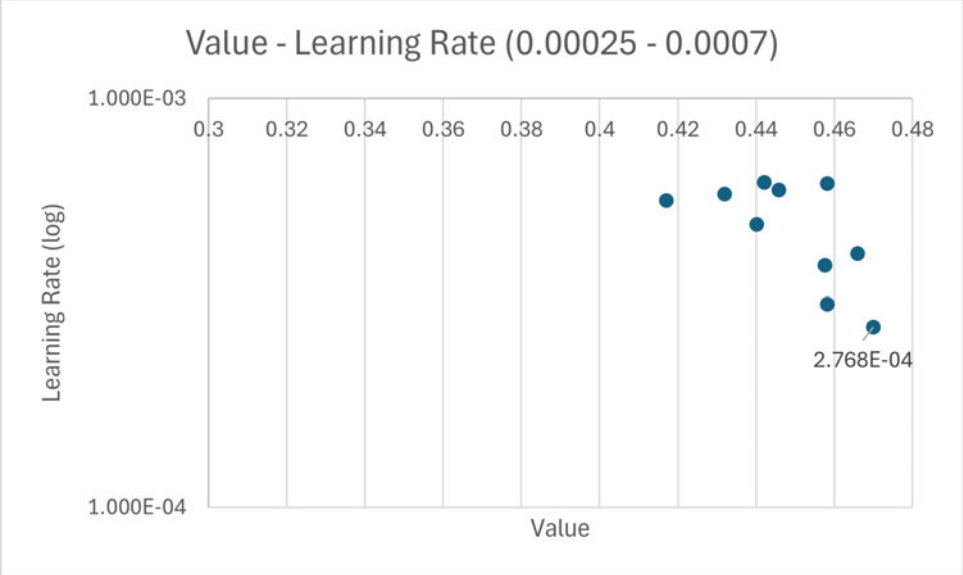


Figure 39: Second Optuna Trials - Learning Rate

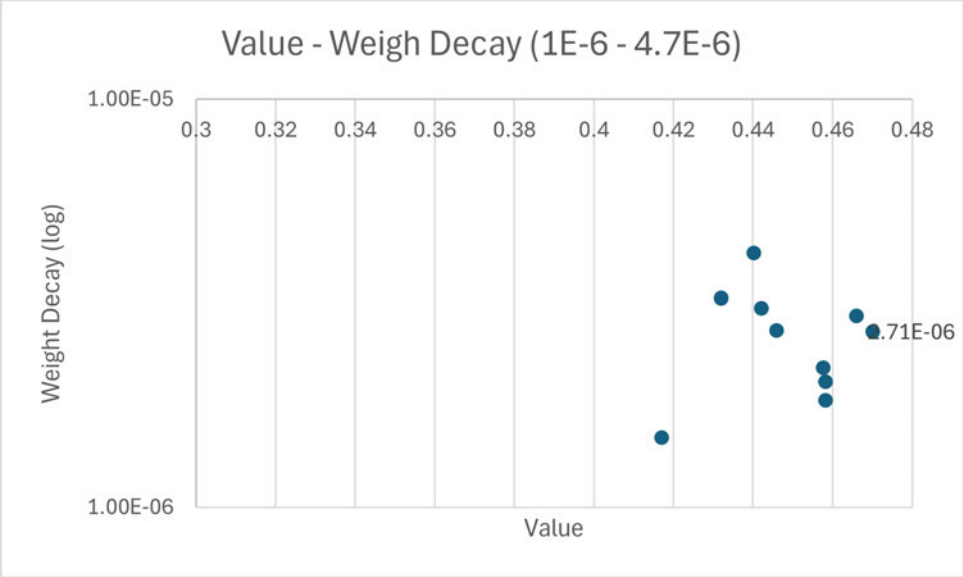


Figure 40: Second Optuna Trials - Weight Decay

Finalised Model

As discussed, of the second round of Optuna modelling the best results were from 'Trial 8' with an mAP of 0.47. The hyperparameter-optimised YOLO11 model shows significant improvements compared to the benchmarked YOLO11m, YOLO11l, and YOLO11x models. The results indicate enhancements in both precision and recall, particularly for the Drone class, while also improving overall detection stability.

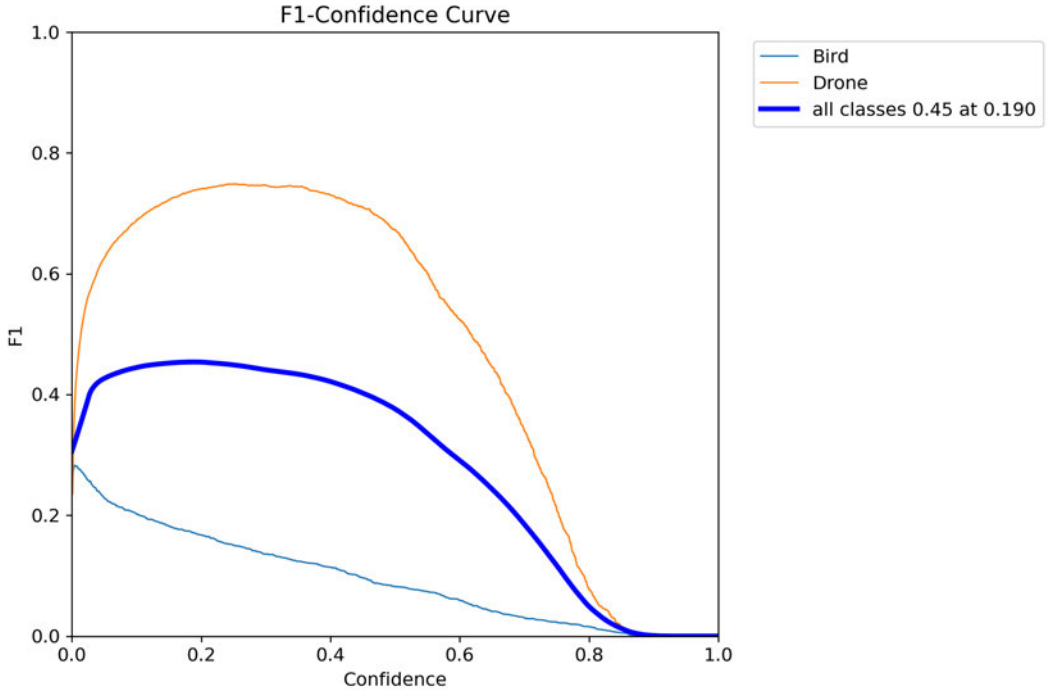


Figure 41: Trial 8 - F1 Curve

The F1-Confidence Curve in Figure 41 shows a peak F1 score of 0.45 at a confidence threshold of 0.19, which is the highest observed among the YOLO11 models. The Drone class significantly outperforms the Bird class, achieving an F1 score approaching 0.75. The overall F1 curve for all classes demonstrates a more gradual decline, indicating that the model maintains a strong balance between precision and recall across various confidence levels. This improvement suggests that the HPO has effectively addressed issues related to false positives and inconsistent confidence scores.

The Precision-Recall Curve in Figure 42 shows that the hyperparameter-optimised model achieves a mAP50 of 0.469, which is a substantial improvement over YOLO11l (0.346) and YOLO11x (0.342). The Drone class reaches a precision of 0.735, demonstrating a significant boost in accurate UAS detection, while the Bird class precision improves to 0.204, up from the previously lower values (0.124). The PR curve for the Drone class also maintains a smoother decline compared to earlier models, highlighting improved consistency across different recall thresholds.

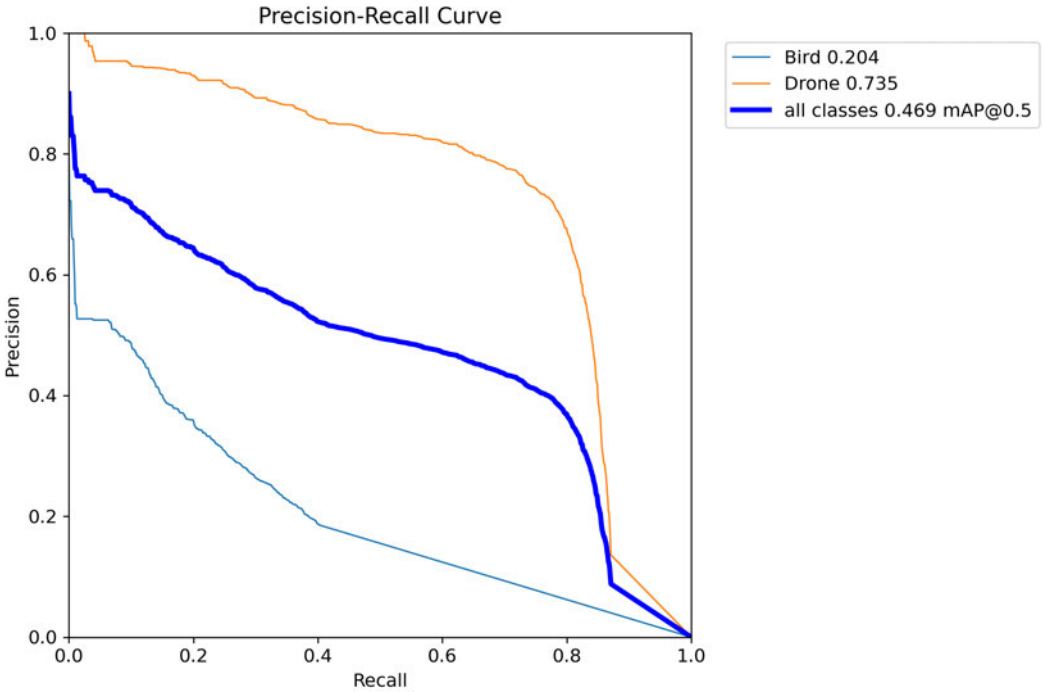


Figure 42: Trial 8 - PR Curve

The confusion matrix in Figure 43 shows a significant reduction in misclassifications and improved detection accuracy. The Drone class achieves 1986 true positives, an improvement over the YOLO11l model's 1921 and YOLO11x's 1784. False positives for drones have also decreased, with fewer background objects being misclassified. For the Bird class, while challenges remain, there are clear improvements, with 433 true positives and fewer misclassifications compared to earlier models. The overall distribution of predictions reflects the model's improved ability to distinguish between classes.

The training and validation loss curves in Figure 44 provide further evidence of improved performance and stability. The box loss, classification loss, and distribution focal loss decrease smoothly over 100+ epochs, with reduced variability compared to the earlier

models. Metrics for precision and recall consistently improve throughout the training process, achieving higher overall values by the end of training. The mAP50-95 metric also shows steady improvement, indicating enhanced model generalisation across different IoU thresholds.

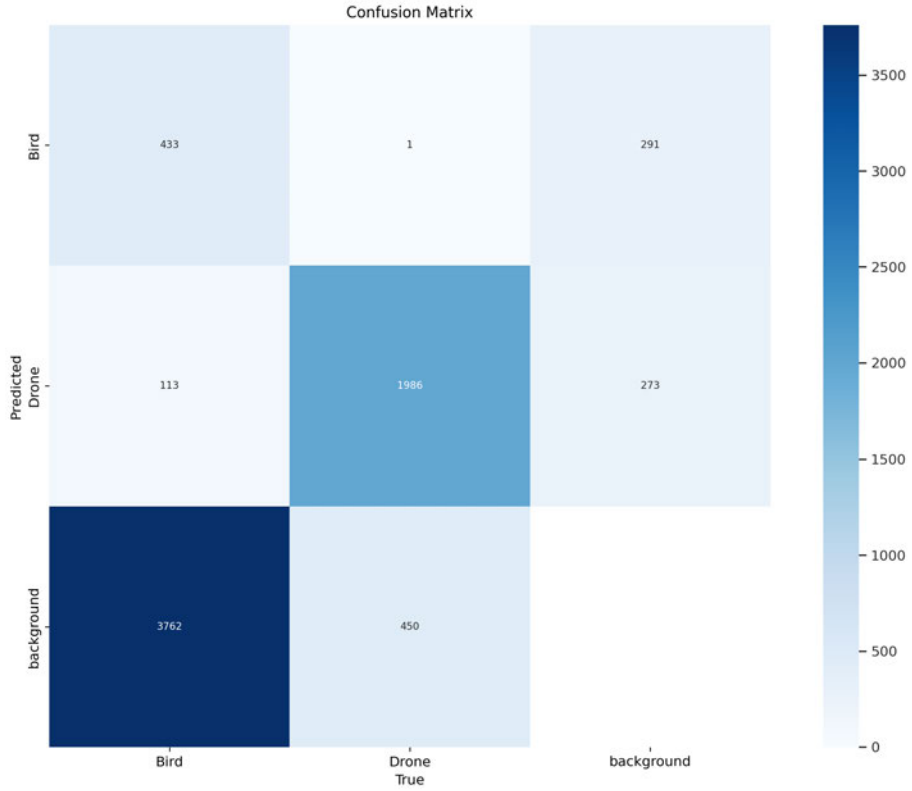


Figure 43: Trial 8 - Confusion Matrix

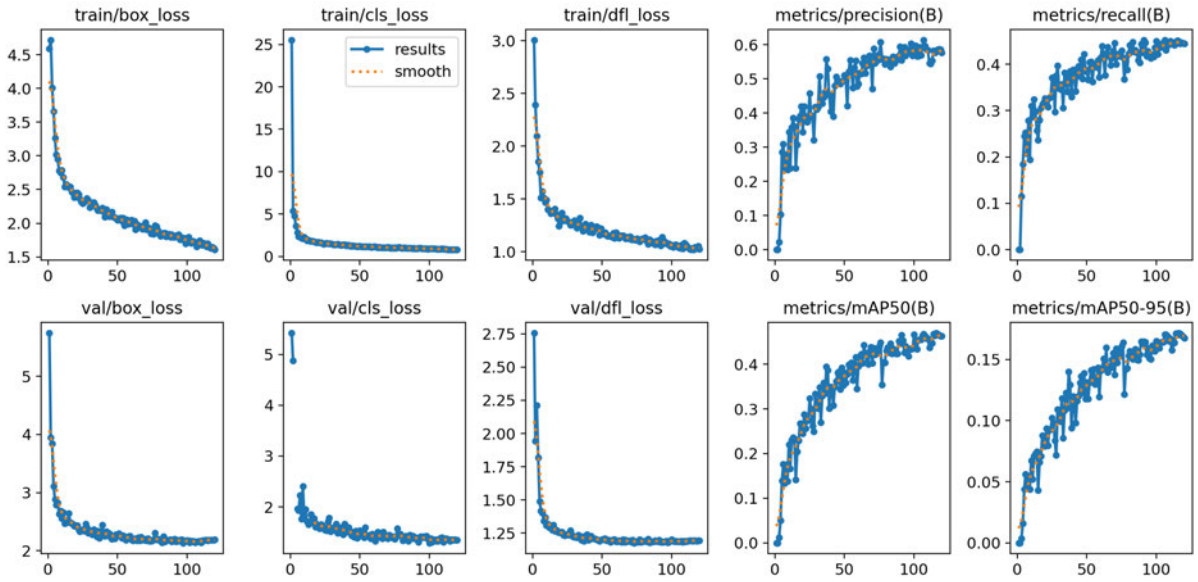


Figure 44: Trial 8 - Results

The comparison between the Trial 8 optimised model in Figure 45 and the baseline YOLO11 model in Figure 25 demonstrates clear improvements in detection performance, particularly for the Drone class. The optimised model consistently predicts bounding boxes with higher confidence scores, often ranging between 0.5 and 0.8, compared to the lower confidence scores of 0.3 to 0.5 seen in the benchmark YOLO11 predictions. This increased confidence reflects the model's improved ability to distinguish drones, even in challenging scenarios such as partial occlusion or reduced visibility. Additionally, the optimised model successfully detects nearly all ground-truth objects, with fewer missed detections and improved localisation accuracy. False positives are noticeably reduced, as the optimised model avoids misclassifying background features, a weakness occasionally observed in the YOLO11 outputs. The predictions from the optimised model are more consistent across all images, with better alignment of bounding boxes to the ground-truth annotations.

(a) Truth (b) Predicted

Figure 45: Trial 8 comparison of truth against predicted images

Video Dataset Results

The video dataset, provided by Coluccia et al. (2023), consisted of five training videos in various settings. Whilst the actual videos, with predicted bounding box generation provided by the YOLO11 model, are unable to be provided, random still frames of the resulting videos are shown in the Figures below.

Figure 46 shows an individual UAS in a darkened scene and various cloud formations. The difficulty of detection is quite high, as the UAS is very small, so a confidence result of 0.31 is considered a success. The confidence did reach 0.67 when the UAS was closer in the image, but the still frame shows success at long ranges and more realistic scenarios of smaller UAS in the image.

Figure 46: Bird vs. Drone Dataset - Test Video 1, Single UAS

Figure 47 shows a larger UAS against a calmer background with a high confidence level detection. This particual video test had a tracking rate of 85.7%, showing the algorithm is able to detect UAS in uncluttered environments.

Figure 47: Bird vs. Drone Dataset - Test Video 2, Single UAS

Figure 48 shows a much more complicated scene with a UAS cluster of four UASs and a pixelated, blurry image. The YOLO11 testing algorithm was able to successfully capture and track all four UASs with high confidence levels, showing it is able to successfuly detect more than a single UAS in a frame.

Figure 48: Bird vs. Drone Dataset - Test Video 3, UAS Cluster

Figure 49 shows two UAS and a plane. Whilst the plane is quite small in the image, located approximately halfway down on the second-third, right side of the image, it is larger than the UAS. This particular video did have false positives where it detected the plane as a UAS, but the algorithm tracked the UAS with much higher confidence levels showing it is able to understand the difference.

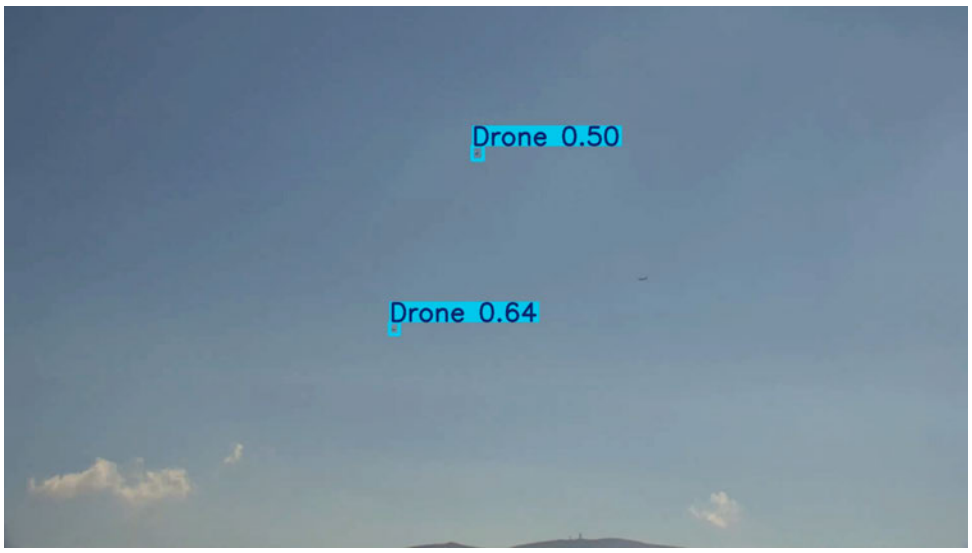


Figure 49: Bird vs. Drone Dataset - Test Video 4, Two UAS and a plane

Figure 50 shows a single UAS on a cluttered background of a hillside, which makes detection more difficult. The algorithm was able to detect the UAS with a tracking rate of

92.1%, showing the algorithm actually detects at a higher rate in cluttered environments. This test video also moved the camera significantly, which meant the UAS left the frame, and was immediately detected upon re-entering the frame.

Figure 50: Bird vs. Drone Dataset - Test Video 5, Single UAS against mountain

This testing was incredibly successful, showing that the algorithm was able to detect with a high level on accuracy in real-time scenarios. More importantly, the algorithm was able to perform comfortably at a resolution of 1080p, 60fps on the Desktop computer.

DISCUSSION AND CONCLUSIONS

Evaluation of Algorithm Performance

The hyperparameter-optimised YOLO11 model demonstrates a clear and significant improvement over the baseline YOLO11m, YOLO11l, and YOLO11x models, achieving the best balance between accuracy, precision, and recall. The model attained a mAP50 of 0.469, a peak F1 score of 0.45, and an impressive precision of 0.735 for the Drone class, showcasing its ability to accurately detect and distinguish drones from other objects. The improvements are particularly evident in the Drone class, where the model consistently reduced false positives while increasing true positive detections, even in challenging scenarios such as occlusions or low visibility. The optimised model also displayed enhanced training stability, with smoother convergence in box loss, classification loss, and distribution focal loss over the training period.

The comparison of predictions against ground-truth annotations further shows the models effectiveness. The optimised YOLO11 successfully detected and tracked the UAS in all five testing drone-vs-bird datasets, a task that was challenging for the baseline models. The improved model produced bounding boxes with higher confidence scores and greater alignment with ground-truth labels, while minimising misclassifications and inconsistencies. This consistent detection performance reflects the impact of hyperparameter tuning in refining the model's precision and generalizability.

Notably, these improvements were achieved without a significant increase in computational costs compared to larger models like YOLO11x. By using targeted HPO, the model maintains a practical training time while delivering quality results.

Limitations

While the model did perform better than expectations, it was not without its limitations. The model was very good at detecting UAS from a distance, particularly in small pixel ranges, but the training dataset, BirDrone, lacked a significant amount of large UASs. Upon reflection, this could have been overcome with better augmentation of the datatsets during training such as zooming of images, or including an entirely different dataset that had larger images of UASs included. An mAP50 of 0.469, while respectable for a particularly difficult dataset such as BirDrone, could definitely be improved to above 50 with better training and tighter parameters.

The model also struggled significantly with the bird class, and whilst this wasn't entirely an issue as the aim wasn't to detect birds, more to not misclassify UAS as birds, it would have been a lot better to see the model classify them correctly instead of misidentifying them. This again could have been fixed with a larger training dataset.

It is also of note that during the testing of a real-time video dataset, notably the drone-vs-bird testing videos, the model was unable to maintain a flawless tracking of UASs. This again isn't a large issue as the intention was always to detect the UAS, rather than flawlessly track it.

The main limitation of this study was unfortunately time.

Future Work

While the hyperparameter-optimised YOLO11 model performed better than anticipated, there are several avenues for improvement to enhance its robustness and prepare it for real-world deployment. One key limitation observed was the model's performance on larger UASs. The BirDrone dataset used for training lacked a significant number of larger UAS images, which led to a bias in the model's ability to detect UASs primarily at small pixel ranges and from greater distances. To address this, future work could involve augmenting the existing dataset with techniques such as image zooming, which would simulate larger UASs by scaling the training images. Additionally, incorporating a more diverse dataset that includes high-resolution images of larger UASs at closer ranges would help improve the model's detection accuracy across varying scales and distances.

To prepare the model for real-world deployment, further testing in actual operational environments, such as correctional facilities or open-air prisons, is essential. Such trials would provide valuable insights into the model's performance under varying lighting conditions, backgrounds, and UAS flight behaviors that cannot be fully replicated in synthetic datasets. Additionally, optimising the model for edge devices, such as low-power GPUs or embedded systems, would be crucial for real-time deployment in resource-constrained environments. This could involve techniques such as model quantisation and pruning to reduce the model size while maintaining its performance.

Future work should focus on improving the training dataset to include larger UASs and diverse bird classes, enhancing the model's real-time tracking capabilities, and conducting real-world trials to validate its robustness and reliability. These steps, combined with deployment optimisations for edge devices, would ensure the YOLO11 model is fully equipped for practical applications in UAS detection and monitoring.

Bibliography

1. Abu Zitar, R, Al-Betar, M, Ryalat, M & Kassaymeh, S 2023, 'A review of UAV Visual Detection and Tracking Methods', <https://ui.adsabs.harvard.edu/abs/2023arXiv230605089A, <http://arxiv.org/pdf/2306.05089>.

2. Almarzooq, H & bin Waheed, U 2024, 'Automating hyperparameter optimization in geophysics with Optuna A comparative study', Geophysical Prospecting, vol. 72, no. 5, pp. 1778-88.

3. Almarzooq, H & bin Waheed, U 2024, 'Automating hyperparameter optimization in geophysics with Optuna A comparative study', Geophysical Prospecting, vol. 72, no. 5, pp. 1778-88.

4. Alsanad, HR, Sadik, AZ, Ucan, ON, Ilyas, M & Bayat, O 2022, 'YOLO-V3 based real-time drone detection algorithm', Multimedia Tools and Applications, vol. 81, no. 18, pp. 26185-98.

5. Annaby, M & Fouda, Y 2024, 'Fast template matching and object detection techniques using ϕ-correlation and binary circuits', Multimedia Tools and Applications, vol. 83, no. 3, pp. 6469-96.

6. Aouladhadj, D, Kpre, E, Deniau, V, Kharchouf, A, Gransart, C & Gaquière, C 2023, 'Drone Detection and Tracking Using RF Identification Signals', Sensors (Basel, Switzerland), vol. 23, no. 17, p. 7650.

7. Australia, S 2021, 26 February 2021, 'Saab Australia securing the nation's largest correctional facilities', SAAB, <https://www.saab.com/markets/australia/news/stories/2022/Southern_Queensland_Correctional_Precinct_Stage_2_project>.

8. Australia, S 2022, 18 August 2022, 'Work is well underway installing our OneView integrated security solution at the Southern Queensland Correctional Precinct Stage 2', SAAB, <https://www.saab.com/markets/australia/news/stories/2022/Southern_Queensland_Correctional_Precinct_Stage_2_project>.

9. Australia, IP 2023, 2023, 'Major Contract - Southern Queensland Correctional Precinct Stage Two', Australia New Zealand Infrastructure Pipeline, <https://infrastructurepipeline.org/project/southern-queensland-correctional-precinct-gatton>.

10. Bagaric, M, Hunter, D & Wolf, G 2018, 'Technological Incarceration and the End of the Prison Crisis Criminal Law', J. Crim. L. & Criminology, vol. 108, p. 73.

11. Brewczyński, KD, Życzkowski, M, Cichulski, K, Kamiński, KA, Petsioti, P & De Cubber, G 2024, 'Methods for Assessing the Effectiveness of Modern Counter Unmanned Aircraft Systems', Remote sensing (Basel, Switzerland), vol. 16, no. 19, p. 3714.

12. Brutzer, S, Hoferlin, B & Heidemann, G 'Evaluation of background subtraction techniques for video surveillance', IEEE, pp. 1937-44, <https://ieeexplore.ieee.org/document/5995508/>.

13. Chen, L, Rottensteiner, F & Heipke, C 2021, 'Feature detection and description for image matching from hand-crafted design to deep learning', Geo-spatial information science, vol. 24, no. 1, pp. 58-74.

14. Coluccia, A, Fascista, A, Sommer, L, Schumann, A, Dimou, A & Zarpalas, D 2023, 'The Drone-vs-Bird Detection Grand Challenge at ICASSP 2023 A Review of Methods and Result', IEEE Open Journal of Signal Processing.

15. Corrections, NGMf 2023, Huge contraband haul at Australia's biggest remand prison, NSW Government, NSW Government.

16. Delleji, T, Slimeni, F, Lafi, M, Ayadi, A & Chtourou, Z 2023, 'Deep Sky Monitoring System for Mini-drone Detection and Tracking', 2023 IEEE International Conference on Advanced Systems and Emergent Technologies (IC_ASET), pp. 1-6.

17. Deng, Z, Sun, H, Zhou, S, Zhao, J & Zou, H 2017, 'Toward Fast and Accurate Vehicle Detection in Aerial Images Using Coupled Region-Based Convolutional Neural Networks', IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 10, no. 8, pp. 3652-64.

18. Deng, J, Ji, X, Wang, B, Wang, B & Xu, W 2023, 'Dr. Defender Proactive Detection of Autopilot Drones based on CSI', IEEE transactions on information forensics and security.

19. Fieres, J, Schemmel, J & Meier, K 2006, 'Training convolutional networks of threshold neurons suited for low-power hardware implementation', The 2006 IEEE International Joint Conference on Neural Network Proceedings, pp. 21-8, <https://ieeexplore-ieee-org.ezproxy.usq.edu.au/document/1716065, <https://ieeexplore.ieee.org/document/1716065/>.

20. Flórez, J, Ortega, J, Betancourt, A, García, A, Bedoya, M & Botero, JS 2020, 'A review of algorithms, methods, and techniques for detecting UAVs and UAS using audio, radiofrequency, and video applications', Tecno - Lógicas (Instituto Tecnológico Metropolitano), vol. 23, no. 48, pp. 269-85.

21. Goedbled, B 2019, 'Artificial intelligence in a prison environment', Technology in Corrections Conference Digital Transformation, International Corrections & Prisons Association, Cyprus, <https://rm.coe.int/benny-goedbloed-ai-in-a-prison-environment-cdpps2019/168094ac84>.

22. Imandeka, E, Hidayanto, AN & Mahmud, M 2024, 'Smart prison technology and challenges a systematic literature reviews', IAES International Journal of Artificial Intelligence (IJ-AI); Vol 13, No 2 June 2024.

23. Janiesch, C, Zschech, P & Heinrich, K 2021, 'Machine learning and deep learning', Electronic Markets, vol. 31, no. 3, pp. 685-95.

24. Jouav 2024, PH-20 Heavy Payload Multirotor Drone, <https://www.jouav.com/products/ph-25.html>

25. Kaun, A & Stiernstedt, F 2020, 'Doing time, the smart way? Temporalities of the smart prison', New Media & Society, vol. 22, no. 9, pp. 1580-99.

26. Khanam, R & Hussain, M 2024, 'YOLOv11 An Overview of the Key Architectural Enhancements', p. arXiv 2410.17725, <https://ui.adsabs.harvard.edu/abs/2024arXiv241017725K, <http://arxiv.org/pdf/2410.17725>.

27. Kim, G, Dhuper, K, Kwon, S, Chander, P, Kim, N, Matson, ET, Kim, N & Smith, T 2023, 'Power Efficient Long Range Drone Networking System for UAV Detection', 2023 14th International Conference on Information and Communication Technology Convergence (ICTC), pp. 381-6.

28. Kyritsis, A, Makri, R & Uzunoglu, N 2022, 'Small UAS Online Audio DOA Estimation and Real-Time Identification Using Machine Learning', Sensors (Basel, Switzerland), vol. 22, no. 22, p. 8659.

29. Lacava, G, Mercaldo, F, Martinelli, F, Santone, A & Pizzi, M 2022, 'Drone Audio recognition based on Machine Learning Techniques', Procedia Computer Science, vol. 207, pp. 848-57.

30. Laurito, G, Fraser, B & Rosser, K 2020, 'Airborne Localisation of Small UAS using Visual Detection A Field Experiment', 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1435-43, <https://ieeexplore.ieee.org/document/9308605/>.

31. Lv, H, Liu, F & Yuan, N 2021, 'Drone Presence Detection by the Drone's RF Communication', Journal of physics. Conference series, vol. 1738, no. 1, p. 12044.

32. McKay, C 2022, 'The Carceral Automaton Digital Prisons and Technologies of Detention', International Journal for Crime, Justice and Social Democracy, vol. 11, no. 1, pp. 100-19.

33. Mubarak, AS, Vubangsi, M, Al-Turjman, F, Ameen, ZS, Mahfudh, AS & Alturjman, S 2022, 'Computer Vision Based Drone Detection Using Mask R-CNN', 2022 International Conference on Artificial Intelligence in Everything (AIE), pp. 540-3, <https://ieeexplore.ieee.org/document/9898777/>.

34. Office of the Inspector General, USDoJ 2020, Audit of the Department of Justice's Efforts to Protect Federal Bureau of Prisons Facilities Against Threats Posed by Unmanned Aircraft Systems, Do Justice, Washington, DC.

35. Paszkowski, W, Gola, A & Świć, A 2024, 'Acoustic-Based Drone Detection Using Neural Networks – A Comprehensive Analysis', Advances in science & technology, research journal, vol. 18, no. 1, pp. 36-47.

36. Poitevin, P, Pelletier, M & Lamontagne, P 2017, 'Challenges in detecting UAS with radar', 2017 International Carnahan Conference on Security Technology (ICCST), pp. 1-6, <https://ieeexplore.ieee.org/document/8167852/>.

37. Purtill, J 2023, 'How prisons, drug smugglers and the war in Ukraine made Australia's DroneShield a global success', ABC News, <https://www.abc.net.au/news/science/2023-01-18/droneshield-selling-drone-counter-measures-detection-guns/101852562>.

38. Qasim, S, Nawaz Khan, K, Yu, M & Salman Khan, M 2021, 'Performance Evaluation of Background Subtraction Techniques for Video Frames', EasyChair Preprint, vol. 5335.

39. Ramos-Romero, C, Green, N, Torija, AJ & Asensio, C 2023, 'On-field noise measurements and acoustic characterisation of multi-rotor small unmanned aerial systems', Aerospace Science and Technology, vol. 141, p. 108537.

40. Rao, SN 2024, YOLOv11 Architecture Explained Next-Level Object Detection with Enhanced Speed and Accuracy, <https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71>.

41. Sapkota, R, Qureshi, R, Flores-Calero, M, Badgujar, C, Nepal, U, Poulose, A, Zeno, P, Vaddevolu, UBP, Yan, H & Karkee, M 2024, YOLOv10 to Its Genesis A Decadal and Comprehensive Review of The You Only Look Once Series.

42. Schlicht, M 2023, The Cost of Prisons in Australia 2023, Institute of Public Affairs, <https://ipa.org.au/wp-content/uploads/2023/07/IPA-Cost-of-Prisons-Report.pdf>.

43. Sedunov, A, Haddad, D, Salloum, H, Sutin, A, Sedunov, N & Yakubovskiy, A 2019, 'Stevens Drone Detection Acoustic System and Experiments in Acoustics UAV Tracking', 2019 IEEE International Symposium on Technologies for Homeland Security (HST), pp. 1-7.

44. Seidaliyeva, U, Ilipbayeva, L, Taissariyeva, K, Smailov, N & Matson, ET 2023, 'Advances and Challenges in Drone Detection and Classification Techniques A State-of-the-Art Review', Sensors (Basel, Switzerland), vol. 24, no. 1, p. 125.

45. Sharma, A, Kumar, V & Longchamps, L 2024, 'Comparative performance of YOLOv8, YOLOv9, YOLOv10, YOLOv11 and Faster R-CNN models for detection of multiple weed species', Smart agricultural technology, p. 100648.

46. Song, D, Yuan, F & Ding, C 2022, 'Track Foreign Object Debris Detection based on Improved YOLOv4 Model', 2022 IEEE 6th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), pp. 1991-5, <https://ieeexplore.ieee.org/document/9930100/>.

47. Tang, G, Ni, J, Zhao, Y, Gu, Y & Cao, W 2024, 'A Survey of Object Detection for UAVs Based on Deep Learning', Remote sensing (Basel, Switzerland), vol. 16, no. 1, p. 149.

48. Ultralytics 2024, Ultralytics YOLO11 Overview, <https://docs.ultralytics.com/models/yolo11/>, viewed 24 October.

49. Xu, C, He, F, Chen, B, Jiang, Y & Song, H 2021, 'Adaptive RF Fingerprint Decomposition in Micro UAV Detection based on Machine Learning', ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 7968-72, <https://ieeexplore.ieee.org/document/9414985/>.

50. Zamri, FNM, Gunawan, TS, Yusoff, SH, Mustafah, YM, Kartiwi, M & Yusoff, NM 'BirDrone A Novel Dataset for Enhanced Drone and Bird Detection Using YOLOv9', pp. 1-6.

51. Zamri, FNM, Gunawan, TS, Yusoff, SH, Alzahrani, AA, Bramantoro, A & Kartiwi, M 2024, 'Enhanced Small Drone Detection Using Optimized YOLOv8 With Attention Mechanisms', IEEE access, vol. 12, pp. 90629-43.

52. Zero-X, 2024, Osprey Tech + Specs, <https://www.zero-x.com.au/product-osprey>.

53. Zhai, X, Huang, Z, Li, T, Liu, H & Wang, S 2023, 'YOLO-Drone An Optimized YOLOv8 Network for Tiny UAV Object Detection', Electronics (Basel), vol. 12, no. 17, p. 3664.

54. Zhang, R, Condomines, J-P & Lochin, E 2022, 'A Multifractal Analysis and Machine Learning Based Intrusion Detection System with an Application in a UAS/RADAR System', Drones, vol. 6, no. 1, p. 21.

55. Zhou, Z, Zeng, C, Wang, H & Liao, G 2023, 'Networked Radar System A More Advanced Radar Detection Platform', 2023 3rd International Conference on Frontiers of Electronics, Information and Computation Technologies (ICFEICT), pp. 506-12.

Appendix A: Training Script

The below is the training script for the YOLO11l model. It is identical to the YOLO11m and YOLO11x training scripts, with the model changed to reflect the model being trained.

train.py

```
from ultralytics import YOLO
import datetime
import json
import os
import sys

# Define YOLO algorithm and folder names
alg = "yolo11l"
study_name = f"trn_{alg}_{datetime.datetime.now().strftime('%Y%m%d_%H%M')}"
train_path = f"runs/{study_name}"
val_path = f"runs/{study_name}"

# Check the paths
try:
    os.makedirs(train_path, exist_ok=True)
    os.makedirs(val_path, exist_ok=True)
except Exception as e:
    print(f"Failed to create directories: {e}")
    sys.exit(1)


# Load model
model = YOLO(f"{alg}.yaml")

# Train the model
results = model.train(
    data="config.yaml",
    batch=8,
    epochs=75,
    device=0,
    workers=0,
    project=train_path,
    amp=True,
    optimizer="SGD",
    name = "train"
)

# Validation
last_checkpoint_path = f"{train_path}/train/weights/last.pt"
if not os.path.exists(last_checkpoint_path):
    print(f"Warning: {last_checkpoint_path} not found!")
    sys.exit(1)

model = YOLO(last_checkpoint_path)
metrics = model.val(
    project=val_path,
    name = "val"
)

val_metrics = {
        "mAP@50-95": float(metrics.box.map),
        "mAP@50": float(metrics.box.map50),
        "mAP@75": float(metrics.box.map75),
        "Precision": float(metrics.box.mp),
        "Recall": float(metrics.box.mr),
        "F1": float(metrics.box.f1[0]),
        "Class-wise mAP": metrics.box.maps.tolist()
    }

# Save metrics as a JSON file
try:
```

```
    metrics_path = f"{val_path}/{alg}_metrics.json"
    with open(metrics_path, "w") as f:
        json.dump(val_metrics, f, indent=4)  # Save to JSON
    print(f"Validation metrics saved to {metrics_path}")
except Exception as e:
    print(f"Failed to save validation metrics: {e}")
```

Appendix B: Optimisation Script

The below is the optimisation script for the YOLO11l model. It is identical to the YOLO11m and YOLO11x training scripts, with the model changed to reflect the model being trained.

optimise.py

```python
import optuna
import os
import datetime
import json
from ultralytics import YOLO

# Define variables, YOLO algorithm and folder names
trialsn = 50  # Number of trials for Optuna
alg = "yolo11l"
study_name = f"opt_{alg}_{datetime.datetime.now().strftime('%Y%m%d_%H%M')}"
train_path = f"runs/{study_name}/train"
val_path = f"runs/{study_name}/val"

def objective(trial):
    # Sample hyperparameters
    learning_rate = trial.suggest_float("learning_rate", 1e-4, 1e-2, log=True)
    epochs = trial.suggest_int("epochs", 75, 150)
    batch_size = trial.suggest_int("batch_size", 8, 16, step=4)
    momentum = trial.suggest_float("momentum", 0.9, 0.96)
    weight_decay = trial.suggest_float("weight_decay", 1e-6, 1e-3, log=True)

    # Generate unique trial folder name
    trial_name = f"trial_{trial.number}_{datetime.datetime.now().strftime('%H%M%S')}"

    # Train the YOLO model
    model = YOLO(f"{alg}.yaml")
    model.train(
        data="config.yaml",
        epochs=epochs,
        batch=batch_size,
        lr0=learning_rate,
        momentum=momentum,
        weight_decay=weight_decay,
        workers=8,
        device=0,
        amp=True,
        project=train_path,
        name=trial_name,
        optimizer="SGD",
        close_mosaic=10,
    )

    # Extract mAP50 for last.pt
    last_checkpoint_path = f"runs/{study_name}/train/{trial_name}/weights/last.pt"
    if not os.path.exists(last_checkpoint_path):
        print(f"Warning: {last_checkpoint_path} not found!")
        return 0.0  # Assign default mAP if last.pt is missing

    # Validate
    model = YOLO(last_checkpoint_path)
    results = model.val(
        project=val_path,
        save_json=True,
        name=trial_name,
    )
    map50 = results.box.map50
    print(f"Trial {trial.number} mAP50 (last.pt): {map50:.4f}")
    return map50


```

```python
if __name__ == '__main__':
    # Create directories before starting
    os.makedirs(train_path, exist_ok=True)
    os.makedirs(val_path, exist_ok=True)

    # Using a local SQLite database file "optuna_study.db" in case the trial fails
    storage = f"sqlite:///{os.path.join('runs', study_name, 'optuna_study.db')}"

    pruner = optuna.pruners.SuccessiveHalvingPruner()
    study = optuna.create_study(
        study_name=study_name,
        direction="maximize",
        pruner=pruner,
        storage=storage,        # SQLite storage
        load_if_exists=True     # Load if the study already exists
    )
    study.optimize(objective, n_trials=trialsn)

    # Print the best trial's hyperparameters
    best_trial = study.best_trial
    print(f"Best trial parameters: {best_trial.params}")
    print(f"Best mAP: {best_trial.value}")

    # Save study results as CSV
    study.trials_dataframe().to_csv(f"runs/{study_name}/val/optuna_results.csv", index=False)

    # Save study results as JSON
    try:
        trials_data = []
        for trial in study.trials:
            trial_data = {
                "number": trial.number,
                "state": str(trial.state),
                "value": trial.value,
                "duration": (trial.datetime_complete - trial.datetime_start).total_seconds()
                    if trial.datetime_complete else None,
                "params": trial.params,
                "user_attrs": trial.user_attrs,
                "system_attrs": trial.system_attrs
            }
            trials_data.append(trial_data)

        json_path = f"runs/{study_name}/val/optuna_results.json"
        with open(json_path, "w") as f:
            json.dump(trials_data, f, indent=4)
        print(f"Optuna results saved to {json_path}")
    except Exception as e:
        print(f"Failed to save Optuna results: {e}")
```

Appendix C: Refined Optimisation Script

The below is the optimisation script for the YOLO11l model. It is identical to the YOLO11m and YOLO11x training scripts, with the model changed to reflect the model being trained.

optimise2.py

```python
import optuna
import os
import datetime
import json
from ultralytics import YOLO

# Define variables, YOLO algorithm and folder names
trialsn = 10 # Number of trials for Optuna
alg = "yolo11l"
study_name = f"opt_{alg}_{datetime.datetime.now().strftime('%Y%m%d_%H%M')}"
train_path = f"runs/{study_name}/train"
val_path = f"runs/{study_name}/val"

def objective(trial):
    # Tightened hyperparameters
    learning_rate = trial.suggest_float("learning_rate", 2.5e-4, 7e-4)
    epochs = trial.suggest_int("epochs", 110, 150)
    momentum = trial.suggest_float("momentum", 0.91, 0.935)
    weight_decay = trial.suggest_float("weight_decay", 1e-6, 4.7e-6)

    # Generate unique trial folder name
    trial_name = f"trial_{trial.number}_{datetime.datetime.now().strftime('%H%M%S')}"

    # Train the YOLO model
    model = YOLO(f"{alg}.yaml")
    model.train(
        data="config.yaml",
        epochs=epochs,
        batch=16,
        lr0=learning_rate,
        momentum=momentum,
        weight_decay=weight_decay,
        workers=8,
        device=0,
        amp=True,
        project=train_path,
        name=trial_name,
        optimizer="SGD",
        close_mosaic=10,
    )

    # Extract mAP50 for last.pt
    last_checkpoint_path = f"runs/{study_name}/train/{trial_name}/weights/last.pt"
    if not os.path.exists(last_checkpoint_path):
        print(f"Warning: {last_checkpoint_path} not found!")
        return 0.0  # Assign default mAP if last.pt is missing

    # Validate
    model = YOLO(last_checkpoint_path)
    results = model.val(
        project=val_path,
        save_json=True,
        name=trial_name,
    )
    map50 = results.box.map50
    print(f"Trial {trial.number} mAP50 (last.pt): {map50:.4f}")
    return map50


if __name__ == '__main__':
```

```
    # Create directories before starting
    os.makedirs(train_path, exist_ok=True)
    os.makedirs(val_path, exist_ok=True)

    # Using a local SQLite database file "optuna_study.db" in case the trial fails
    storage = f"sqlite:///{os.path.join('runs', study_name, 'optuna_study.db')}"

    pruner = optuna.pruners.SuccessiveHalvingPruner()
    study = optuna.create_study(
        study_name=study_name,
        direction="maximize",
        pruner=pruner,
        storage=storage,        # SQLite storage
        load_if_exists=True     # Load if the study already exists
    )
    study.optimize(objective, n_trials=trialsn)

    # Print the best trial's hyperparameters
    best_trial = study.best_trial
    print(f"Best trial parameters: {best_trial.params}")
    print(f"Best mAP: {best_trial.value}")

    # Save study results as CSV
    study.trials_dataframe().to_csv(f"runs/{study_name}/val/optuna_results.csv", index=False)

    # Save study results as JSON
    try:
        trials_data = []
        for trial in study.trials:
            trial_data = {
                "number": trial.number,
                "state": str(trial.state),
                "value": trial.value,
                "duration": (trial.datetime_complete - trial.datetime_start).total_seconds()
                    if trial.datetime_complete else None,
                "params": trial.params,
                "user_attrs": trial.user_attrs,
                "system_attrs": trial.system_attrs
            }
            trials_data.append(trial_data)

        json_path = f"runs/{study_name}/val/optuna_results.json"
        with open(json_path, "w") as f:
            json.dump(trials_data, f, indent=4)
        print(f"Optuna results saved to {json_path}")
    except Exception as e:
        print(f"Failed to save Optuna results: {e}")
```