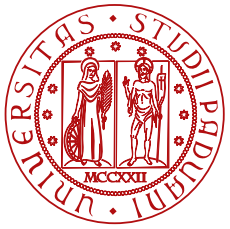


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Information Engineering

Performance Analysis and Evaluation of Object Detection Algorithms for Drone Networks

Master Candidate

Nisan Karsan

Student ID 2040499

Supervisor

Prof. Marco Giordani

University of Padova

Co-supervisor

Asst. Prof. Filippo Campagnaro and Roberto Francescon

University of Padova

Academic Year
2023/2024

To my beloved parents,
my father Haydar and my mother Nermin,
whose unwavering support and endless love have been my constant source of strength,
and to my dearest brother Arman,
my precious Melis,
and friends, whose encouragement
and companionship have been my guiding light through every challenge
this is for you.

Abstract

This thesis explores the optimization and performance evaluation of YOLOv5 and YOLOv8 object detection algorithms within drone networks, particularly when implemented on edge computing devices such as the Raspberry Pi 4. The research is grounded in the practical application of these algorithms to two distinct datasets: the Stanford Drone Dataset and the VisDrone Dataset. These datasets were chosen due to their relevance in aerial surveillance scenarios and the unique challenges they present, such as varying object scales and densities. The findings reveal that both YOLOv5 and YOLOv8, when coupled with OpenVINO and NCNN optimizations, demonstrate substantial improvements in accuracy and processing speed, catering effectively to the constraints of edge computing environments. This study not only highlights the trade-offs between computational efficiency and detection performance but also contributes practical insights into the deployment of AI-driven surveillance systems in resource-limited settings.

Key contributions include a detailed comparative analysis of YOLOv5 and YOLOv8 performance across different datasets and optimization techniques, providing valuable guidelines for implementing these algorithms in real-world drone-based object detection scenarios. The research outcomes offer a foundation for developing more efficient and accurate object detection systems for autonomous drones and other edge-based applications in resource-constrained environments.

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

AP Average Precision

CNN Convolutional Neural Network

CPU Central Processing Unit

CSP Cross Stage Partial

CVGL Computer Vision and Geometry Laboratory

DPM Deformable Part-Base

FPN Feature Pyramid Network

FPS Frames Per Second

GPU Graphics Processing Unit

GPIO General Purpose Input/Output

IoU Intersection over Union

mAP mean Average Precision

ML Machine Learning

NCNN Neural Network Inference Framework

HOG Histogram of Oriented Gradients

OpenVINO Open Visual Inference and Neural Network Optimization

PAN Path Aggregation Network

ROI Regions of Interest

RAM Random Access Memory

RCNN Region-based Convolutional Neural Network

R-FCN Region-based Fully Convolutional Networks

SDD Stanford Drone Dataset

SSD Single Shot Detector

SIFT Scale Invariant Feature Transform

SPPF Spatial Pyramid Pooling Fast

SVM Support Vector Machine

UAV Unmanned Aerial Vehicle

YOLO You Only Look Once

1 Introduction

Network edge object identification has become increasingly crucial due to the rise of drone technologies and the need for real-time data processing [1] [2]. This is particularly vital in sectors such as drone-based monitoring and security surveillance, where accurate and swift object detection can save lives [3].
Drone-based surveillance and monitoring systems rely heavily on object detection. Real-time object identification in drones improves operating safety and effectiveness [4]. In urban areas, these systems can detect and track automobiles, pedestrians, and other items of interest, giving vital data for traffic management and public safety [3]. Similar to this, proactive surveillance systems that use object detection algorithms in security applications can quickly identify possible risks, allowing for prompt interventions and risk reduction [5].

Edge computing plays a fundamental role in reducing latency by processing data near the point of data acquisition, which is essential in mobile or remote work environments where processing capacity and energy availability are limited [6] [7] [8]. Small, power-efficient devices like the Raspberry Pi are ideally suited for these applications, which benefit from decentralized processing frameworks that minimize latency [9].

The rapid evolution of automotive networks and intelligent transportation systems has led to a rise in demand for accurate and efficient object detection algorithms in urban mobility and smart traffic management [10]. Especially when it comes to accident monitoring systems, these improvements have made cities much safer, more efficient, and more secure [11].

Infrastructure-based object detection and tracking systems have become a popular area of study because they might be able to solve the problems that onboard sensors have [12]. UAVs provide a new perspective for monitoring scenarios that contains automotives, complementing ground-based sensors [13]. Drones with advanced object detection capabilities can quickly survey broad regions, detect traffic anomalies, monitor congestion, and even assist in emergency response scenarios. This aerial perspective enhances situational awareness, allowing for more comprehensive and dynamic traffic management strategies [14].

The You Only Look Once (YOLO) family of object detection algorithms, particularly YOLOv5 and YOLOv8, have gained prominence for their speed and accuracy in real-time applications [15]. YOLOv8, in particular, has been developed to further enhance performance and robustness, incorporating innovations such as attention mechanisms and dynamic convolution [16]. These algorithms are well-suited for scenarios requiring low-latency object detection, such as automotive and drone-based monitoring systems. They have been shown to outperform other state-of-the-art benchmarks in terms of detection accuracy and computational efficiency [16]. Furthermore, YOLOv5 has been successfully implemented for real-time object detection and image detection tasks, demonstrating its practical applications in surveillance, automated driving, and robotics [15].

The implementation of object detection on Raspberry Pi platforms has demonstrated high-precision detection and real-time processing capabilities, crucial for edge computing applications [17, 18]. This approach addresses the challenges of wireless communication and real-time processing in distributed object detection architectures [1], particularly for UAV missions that demand low latency and high accuracy [19]. Raspberry Pi's versatility extends to various applications, including image processing, surveillance, and facial recognition [20]. Its potential as a wireless sensor node, capable of interfacing with diverse peripherals, further broadens its research applications [21]. In the context of drone-based systems, Raspberry Pi offers significant advantages. It enables autonomous on-board processing of visual data, reducing network latency and enhancing privacy [22]. The platform optimizes energy consumption, which is crucial for extending operational duration in surveillance missions [23, 24]. Moreover, Raspberry Pi provides a compact, lightweight solution ideal for integration into drone systems, effectively addressing payload and power constraints [25]. By focusing on energy-efficient computing and algorithm optimization for the Raspberry Pi, we can enhance the practicality of drone-based monitoring systems.

This approach not only extends mission durations but also addresses broader challenges in sustainable and long-endurance UAV operations across various applications, from urban planning to emergency response scenarios [26].

The research also involves adapting and benchmarking the YOLO algorithms for edge deployment by converting YOLOv5 to the Open Visual Inference and Neural Network Optimization (OpenVINO) format and YOLOv8 to the Neural Network Inference Framework (NCNN) format. This is to assess their compatibility with Raspberry Pi 4 hardware and optimize their performance for edge computing scenarios.

A comparative analysis will evaluate the performance trade-offs associated with each adaptation, providing insights into the balance between speed, power efficiency, and accuracy in object detection tasks.

This thesis explores the optimization and performance evaluation of YOLOv5 and YOLOv8 object detection algorithms, focusing on their implementation on edge computing devices such as the Raspberry Pi 4. By analyzing these algorithms' performance on two distinct datasets relevant to urban and aerial surveillance - the Stanford Drone Dataset and the VisDrone Dataset - we aim to provide insights into the practical application of advanced object detection in resource-constrained environments. Our comprehensive approach assesses the models' performance across critical metrics, including mean Average Precision (mAP), inference time, and energy consumption. Additionally, we explore the impact of model optimization techniques such as OpenVINO and NCNN, and investigate the effects of object grouping on detection performance.

This research not only addresses the technical challenges of deploying AI-driven surveillance systems in edge computing scenarios but also contributes to the broader goal of enhancing safety and efficiency in smart city and intelligent transportation applications.

We implement YOLOv5 and YOLOv8 algorithms using the PyTorch framework, leveraging their pre-trained weights as starting points. For our edge computing simulations, we use Python 3.9.2 along with PyTorch 1.11.0 and torchvision 0.12.0 for model deployment on Raspberry Pi 4 hardware. To optimize performance, we utilize OpenVINO for YOLOv5 and NCNN for YOLOv8, enabling efficient inference on the Raspberry Pi platform.

Our experiments yield several significant findings. In terms of detection accuracy, YOLOv8 models, particularly YOLOv8l, demonstrate superior performance on aerial imagery, achieving mAP scores of up to 56.1% at IoU 0.5. YOLOv5 models show strong performance in urban surveillance scenarios, with YOLOv5x reaching an mAP of 80.7% at IoU 0.5. Regarding inference time, smaller models like YOLOv8n exhibit lower latency, with times as low as 389.6 ms on grouped object datasets at 416x416 resolution. Energy consumption analysis reveals that optimized models significantly reduce power usage, with NCNN and OpenVINO configurations showing particular efficiency.

In conclusion, our research demonstrates that careful selection of model architecture, optimization technique, and dataset preprocessing can substantially enhance the performance of object detection systems in drone-based edge computing scenarios, balancing accuracy, speed, and energy efficiency.

This thesis is structured as follows:

- **Chapter 2:** Existing research is examined in depth. It analyses the evolution of object detection algorithms, focusing on the YOLO family, and edge computing in modern applications. The review lays the groundwork for this thesis by identifying gaps and opportunities.
- **Chapter 3:** The experimental design, datasets, YOLOv5 and YOLOv8 algorithms, and Raspberry Pi 4 hardware configuration are covered in this thesis. It describes adapting algorithms to OpenVINO and NCNN edge computing frameworks and training and testing methodologies.
- **Chapter 4:** Presentation of experimental results. This contains object detection method performance on multiple datasets, Raspberry Pi 4 inference time, and power consumption statistics. The algorithms' accuracy and optimizations are also covered.
- **Conclusions and Future Work:** The final chapter summarises the research's main findings, discusses field contributions, and offers further research. It concludes that edge device object detection methods must be optimized.

2

Related Works

Object detection is a computer technology used in Artificial Intelligence (AI) and Machine Learning (ML) that involves identifying and locating objects within digital images or videos. This technology plays a crucial role in various applications, from security surveillance systems to autonomous vehicles, by allowing machines to interpret and analyze visual data similarly to how humans do. The basic process involves several steps; image acquisition, pre-processing, feature extraction, object classification, and localization.

Unmanned Aerial Vehicle (UAV)s use object detection for a variety of reasons:

- Navigation and Collision Avoidance: To safely navigate their environment, avoid obstacles, and maintain safe distances from other objects or terrain features.
- Surveillance and Monitoring: For tasks that require consistent monitoring, like border security or traffic analysis, where drones need to identify specific objects or activities.
- Search and Rescue: To quickly locate people in distress in vast or challenging terrains, improving response times and outcomes in emergencies.
- Agricultural and Environmental Monitoring: To assess crop health, detect pests, and manage farms more efficiently, saving time and resources.
- Delivery and Logistics: For identifying delivery points, avoiding hazards, and ensuring that goods are transported safely and efficiently [5].

2.1 The Difference between UAV Object Detection and Common Object Detection

Object detection algorithms commonly rely on datasets obtained from handheld cameras or fixed settings, resulting in a predominance of side-view images in a typical view. Nevertheless, due to their elevated vantage point, aerial images captured by UAVs display noticeable disparities compared to conventional ground-level photographs. Hence, the object detection techniques utilized in standard visual perception cannot be readily adapted to UAV aerial imagery. Various challenges, such as equipment instability, can negatively impact the quality of UAV aerial photos. These challenges can result in issues such as jitter, blur, low resolution, light changes, and image distortion. To improve the effectiveness of detection systems, it is essential to preprocess these considerations for the video [27].

Furthermore, the aerial image displays an uneven distribution of objects and a significant prevalence of minuscule object dimensions. For example, when viewed from above, cars and people may cover many pixels, but when viewed normally, they only cover a few pixels. The uneven distribution of these objects causes them to appear distorted, making it difficult to recognize multiple objects at once. This requires the employment of specialist network modules to extract features.

There are differences in the occlusion seen from a normal perspective compared to that seen from an aerial perspective. The item can be obscured from ordinary vision by additional obstructions, such as an individual positioned in front of a vehicle. Nevertheless, when observed from an aerial perspective, the object can get obscured by the encompassing surroundings, such as foliage and structures [28] [29].

2.2 Challenges in UAV Object Detection

UAVs can operate at different heights and view angles, making vehicle detection and recognizing pedestrians, cars, skaters, bikers, and others in aerial imagery difficult. Scale variations, diverse orientations and types of objects, changes in illumination, high object density, similarity in appearance, partial occlusions, complex backgrounds, and varying image qualities make it difficult to detect these objects in UAV-based images. The problem gets further complicated by limited annotated training datasets and real-time detection.

- Small object: High-resolution aerial images show fewer items besides autos than typical scene photographs, resulting in less feature extraction information. This increases missed detections and localization difficulty.
- Scale Diversity: A UAV's varied altitudes can confuse and ambiguously size an object. The different scales of cars, pedestrians, and other objects in photos and videos make recognition harder.
- Object Orientations and Types: UAVs' mobility offers front, side, and top views. Pictures and videos show a range of elements in different shapes and orientations, making detection harder.
- Illumination Condition Variations: Light, which is crucial to categorization and detection, is one of the hardest problems to solve. Image preprocessing is needed to decrease lighting and illumination effects to construct a reliable and accurate system.
- Complex Background and Similar Appearance: Due to their large size and wide viewing angles, aerial photographs have complicated backgrounds with many items, making it difficult to discern between comparable object types. A pedestrian could be mistaken for a skater, bicyclist, or tree. Vehicles can also be mistaken for trash bins and air conditioners, making identification and classification more difficult.
- Limited Annotated Datasets: Low-availability annotated datasets make aerial image-based vehicle or pedestrian recognition harder. The lack of tagged object datasets in UAV photography reduces detector accuracy, especially for deep-learning-based detectors that need many annotated images/videos for training. The most accessible datasets are satellite imagery. Most of the study is based on their UAV-gathered datasets.
- Real-Time Issue: UAVs must detect and classify automobiles, pedestrians, skaters, and bicycles in crowded surroundings in real-time to manage road traffic or parking lots. However, limited onboard hardware resources on the UAV platform make sophisticated tasks with huge data loads difficult, making real-time detection and classification difficult. Aerial photos are larger than scene images, slowing detection. Thus, low-power and low-processing UAV platforms need accurate, rapid detection algorithms that can handle many object kinds to fulfill their missions quickly [30].

Figure 2.1: Images from the Visdrone dataset.
[31]

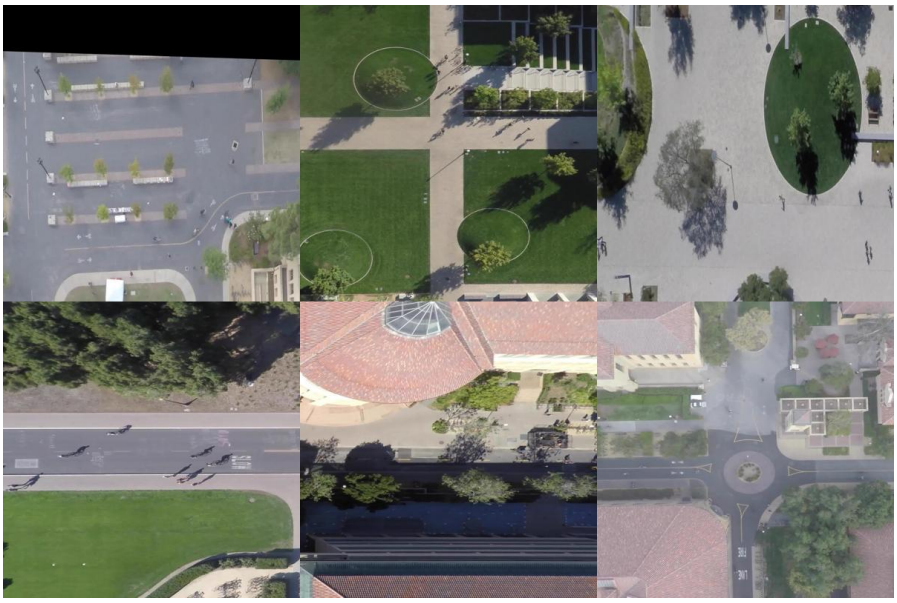


Figure 2.2: Images from Stanford Drone Dataset.
[32]

In the challenging frames of drone-based videos([32], [31]) defocus, motion blur, occlusion, and other variations (e.g., illumination, size, small object) may leave too few clues for successful detections (Figures 2.1 and 2.2).

2.3 Traditional Object Detection Approaches

Vision-based and machine classifier-based methods are part of classical categorization. Our main interest is deep learning methods, thus modern categorization emphasizes these. Figure 2.3 illustrates the detection models, both classical and deep learning-based. Classical object detection methods include improvements in aerial images using machine learning algorithms with handcrafted features. Traditional aerial image analysis methods include inertial optical flow, shape-based descriptors, online boosting with Histogram of Oriented Gradients (HOG) based features, Deformable Part-Base (DPM) descriptors, multiple trained cascaded Haar classifiers, and Markov random field descriptors.

Machine learning algorithms for aerial photos leverage automated classifiers on human-selected parameters to improve performance. Bayesian networks, graph cut methods, HOG with Support Vector Machine (SVM) classifier, Viola jones-SVM hybrid, multi-scale HOG, AdaBoost classifier, Scale Invariant Feature Transform (SIFT) descriptor with SVM classifier, and stochastic constraints-based detection systems are used. Both linear and rotational velocities are detected and distinguished using inertial optical flow. In high-visual-noise environments, shape-based characteristics were used to recognize fixed and moving objects. For fuzzy aerial data detection, the systems used a Bayesian network with numerous learning features and gradient mask filters as low-resolution features. The system displayed poor performance.

Classic object detection methods like inertial optical flow, shape context feature descriptors, and boosting frameworks with HOG features in aerial images had trouble learning object-specific features like contour size and hierarchical shape dynamics. These techniques were vulnerable to errors in real-world settings due to their inability to handle moving objects, dynamic backgrounds, or lighting variations, which are essential to aerial pictures. Due to image resolution and feature description issues, these models were unable to attain considerable accuracy. When feeding an algorithm a vector with millions of values, HOG, SIFT, and Markov Random field are inefficient. Its length and noisy data make it difficult [33].

2.4 Deep Learning Object Detection Approaches

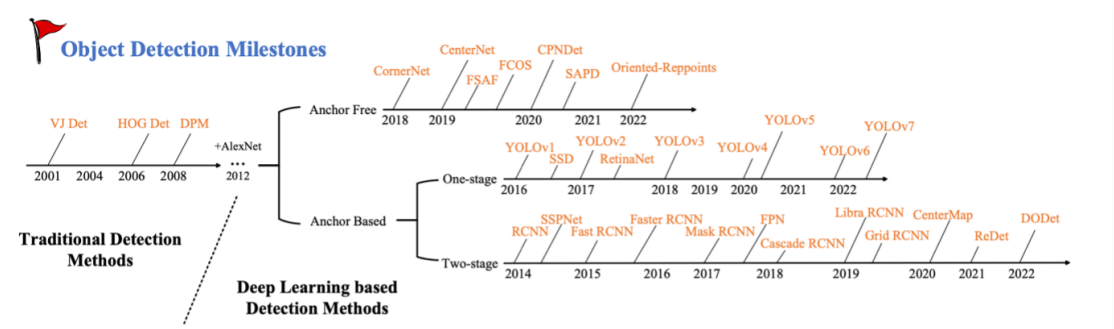


Figure 2.3: The development history of object detection. [34]

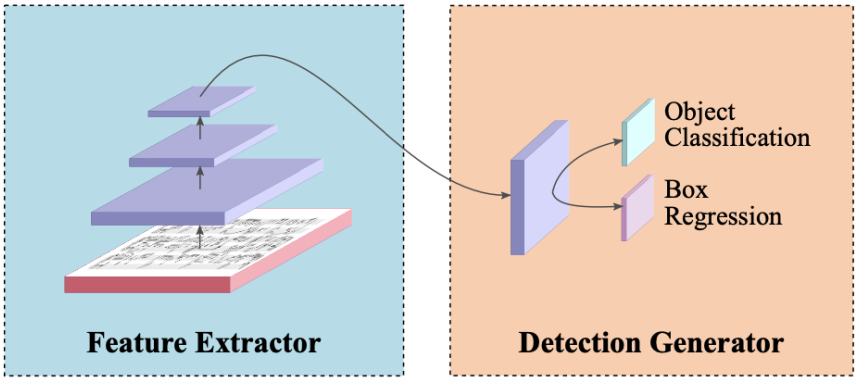
UAV aerial imaging is utilized for entertainment, detection and classification investigations, animal observation, and other exciting reasons. Recent UAVs are cheap to operate for end consumers who seeking aerial imaging equipment on a budget, unlike an aircraft. Advanced deep learning-based object detection methods have a promising future. Recent years have seen several improvements in deep learning-based object detection systems in low-altitude UAVs. Since the dataset distribution includes photos collected from top view angles and some from lower view angles, perspective variance is one of the main issues in drone-recorded photographs. Object traits learned from different viewpoints are not transportable. Thus, sophisticated detectors must detect aerial images. Fig. 2.3 shows two-stage, one-stage, and advanced deep learning-based aerial picture object detection techniques. Two-stage detectors include faster Region-based Convolutional Neural Network (RCNN), Mask RCNN, Cascade RCNN, Feature Pyramid Network (FPN), and Region-based Fully Convolutional Networks (R-FCN), whereas one-stage detectors include YOLO, Single Shot Detector (SSD), RefineDet, and RetinaNet [33].

2.4.1 Two stage-based object detection algorithms

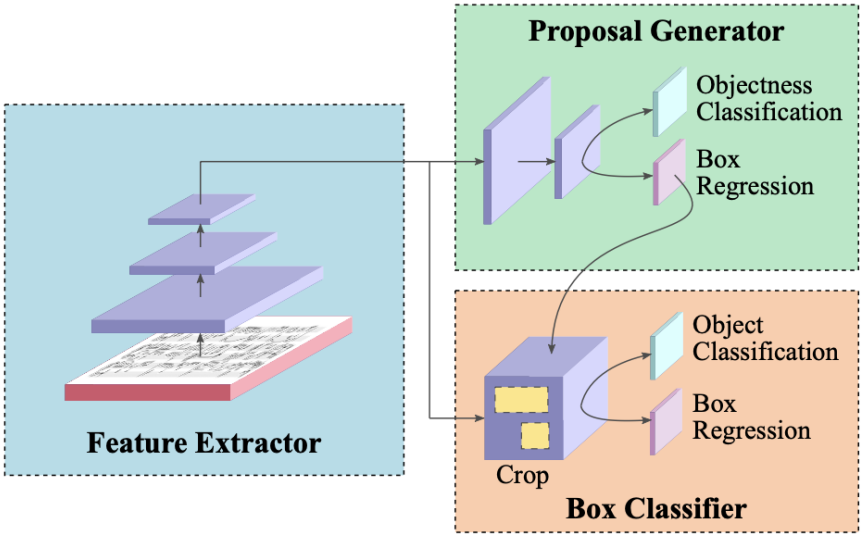
Two-stage object identification algorithms employ a two-step process to identify objects. Fig.2.4 shows the Basic architecture of two-stage and one-stage detectors.
Initially, they generate a limited number of Regions of Interest (ROI) and subsequently classify each of these areas using a network. The RCNN algorithm, which was one of the earliest advanced algorithms, greatly enhanced object recognition by utilizing deep learning to compute the locations of objects from a vast collection of region candidates. It achieved this by cropping and classifying each candidate individually. The Fast RCNN algorithm was developed to mitigate the significant computational and temporal expenses associated with RCNN. It achieves this by employing an end-to-end training process that classifies item suggestions and accurately determines their bounding box coordinates. Fast RCNN enhanced the performance of RCNN by enhancing detection accuracy and reducing the requirement for storing individual feature extraction.

Additional progress has been made with the introduction of Feature Pyramid Networks (FPN), which offer a multi-scale feature representation that is crucial for recognizing objects at various scales. This addresses a fundamental obstacle in the field of computer vision. FPN has been expanded to include mask proposals, which improves both recall and speed for object detection tasks. Mask RCNN enhances object localization accuracy by incorporating a mask predictor.

In the context of drone object detection, especially in low-altitude aerial photos where vehicles appear diminutive and are difficult to accurately locate, conventional approaches such as RCNN and Fast RCNN frequently prove inadequate. They are not designed to efficiently handle small objects or accurately extract specific attributes beyond the bounding box. These networks have significant slowdowns due to the substantial computing load and the nature of their architecture. Recent breakthroughs, such as Cascade RCNN and light-head RCNN, enhance object detection in high-resolution aerial photos by integrating attention mechanisms. Nevertheless, additional adjustments are necessary for these approaches to adequately fulfill the requirements of detecting low-altitude airborne objects [33].



(**a**) Basic architecture of a one-stage detector.



(**b**) Basic architecture of a two-stage detector.

Figure 2.4: Displays a schematic plot illustrating the configuration of a one-stage detector (a) and a two-stage detector (b) [35].

2.4.2 One stage-based object detection algorithms

Two-stage detectors enabled early deep learning-based object detection, but the inference speed was a problem. The efficiency of one-stage detectors over two-stage detectors makes them suitable for low-altitude object detection. Thus, researchers finally switched to one-stage detectors given their flexibility, high-speed and low-memory requirements. Single-stage algorithms pass the entire image into a grid-based Convolutional Neural Network (CNN) instead of in patches like two-stage detectors. Fig. 2.4 displays a one-stage object-detector model. Initially, one-stage detection algorithms recommended a real-time single-pass method called YOLO, which provided better findings mAP higher than two-stage detectors) in a short time. The aim was to forecast object number and location from an image. YOLO trained on full photos improved detection. This integrated object detection model had many advantages over previous methods such as:

- 45 Frames Per Second (FPS) base network speed on high-performance GPU.
- Generalizable object representations reduce breakdown risk in new domains.

SSD algorithm, a more advanced single-shot detector, outperformed two-stage detectors that proposed regions. SSD feature detection outperformed prior detectors that calculated features by executing a neural network on input once. Anchor boxes were used to learn bounding box coordinates. RefineDet's two-step cascade regression improves one-stage detector. These two interconnected modules mimic the two-stage structure for precise, efficient detection. RefineDet achieves state-of-the-art generic object identification (PASCAL VOC 2007, 2012, and MS COCO). RetinaNet, another FPN-based single-stage detector, uses focal loss to address class imbalance induced by excessive foreground-background ratio. A new dynamic loss function called focused loss was employed to change weights between positive and negative training data to solve RetinaNet's degenerative model for a large number of background cases. A simple dense detector was trained to test focus loss. The most accurate object detectors use region proposal methods like RCNN series to apply a classifier to a sparse set of possible object locations. One-pass detectors are faster and simpler but less accurate than two-pass detectors. Foreground background class imbalance during dense detector training is the main cause of accuracy issues [33].

2.5 Performance Metrics for Object Detection

Once object detection algorithms have been developed, they need to go through testing. The evaluation of object detection approaches includes evaluating many metrics such as mAP,Intersection over Union (IoU), recall rate, and precision. These are the key evaluation metrics [34].

True Positive, False Positive, True Negative, and False Negative

The true positive (TP) result indicates the detector's accurate automobile count. A bounding box is positive if its IoU value exceeds a threshold. If the same object has many bounding boxes, only the one with the largest overlap ratio is a positive detection. False positive (FP) is the number of autos the detector misidentifies. Bounding boxes are false negatives (FNs) when they fail to detect vehicles and true negatives (TNs) when they accurately identify misdetections. "IoU" stands for intersection over union, which measures how well the bounding box matches the item.

Intersection Over Union

A detection frame is formed when an object is detected. IoU is a mathematical measure of the ratio of the overlapping area between a priori and real frames to their total area. The threshold is usually 0.5, the same as the cross-union ratio. The object is detected if the value surpasses 0.5.

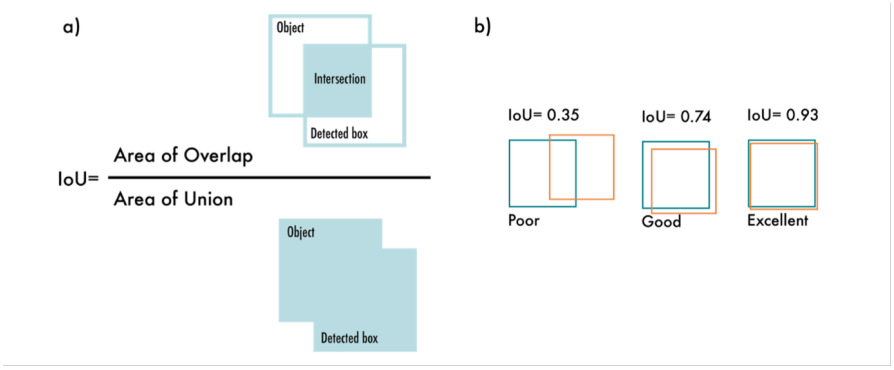


Figure 2.5: Intersection over Union. a) The IoU is calculated by dividing the intersection of the two boxes by the union of the boxes; b) examples of three different IoU values for different box locations [36].

Recall

An essential parameter for assessing the detector's performance is its recall rate. The following illustrates the proportion of predicted positive objects that are objects.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} = \frac{TruePositive}{AllObservations}$$

Precision

Precision is defined as the model's accuracy in identifying the correct sample divided by the total sample in the prediction result. When the intersection-union ratio is more than the threshold, the result is classified as True Positive (TP); otherwise, it is classified as False Positive (FP). If an object is not picked up by the detector in the detection frame that has the sample designated on it, it is classified as False Negative (FN). Accuracy is defined as follows:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} = \frac{TruePositive}{AllGroundTruth}$$

Average Precision

Average Precision (AP) is the precision averaging over a [0, 1] recall. The greater the AP value, the better the detector performs in detecting a particular type of object in the dataset. The following is the definition of average precision.

$$AP_u = \frac{1}{|\Omega_u|} \sum_{i \in \Omega_u} \frac{\sum_{j \in \Omega_u} h(p_{uj} < p_{ui}) + 1}{p_{ui}}$$

where the location of object j is indicated by p_{uj}, the Ground Truth result is indicated by Ω_u, and the recommendation list's prioritisation of object j over item i is indicated by $p_{uj} < p_{ui}$.

Mean Average Precision

The average accuracy of each type of item that the detector detects is averaged (mAP). For the whole dataset, higher mAP values signify improved detector performance. The definition of the mean average accuracy is

$$mAP = \frac{\sum_{u \in U} AP_u}{|U|}$$

2.6 Evolution of YOLO Algorithms

It introduced an end-to-end real-time object identification method. YOLO, which stands for "You Only Look Once," refers to the fact that it completed the detection task in a single network pass, as opposed to earlier techniques that either employed sliding windows and required hundreds or thousands of runs of a classifier for each image, or more sophisticated techniques that divided the task into two steps: the first step finds potential regions with objects or regions proposals, and the second step runs a classifier on the proposals.
Additionally, YOLO employed a simpler output that relied just on regression to forecast the detection outputs, in contrast to Fast RCNN, which employed two distinct outputs a regression for the box coordinates and a classification for the probabilities. YOLO's real-time object detection has helped autonomous driving systems identify and track vehicles, pedestrians, bicycles, and other obstacles. These skills have been used in action recognition, surveillance video analysis, sports analysis, and human-computer interaction. YOLO models have been used for license plate detection and traffic sign recognition, helping develop intelligent transportation systems and traffic management solutions. They locate and monitor endangered species for biodiversity conservation and ecological management. Finally, YOLO is applied in robotics and drone object identification. This thesis focuses on researching drone object detection using YOLO techniques.

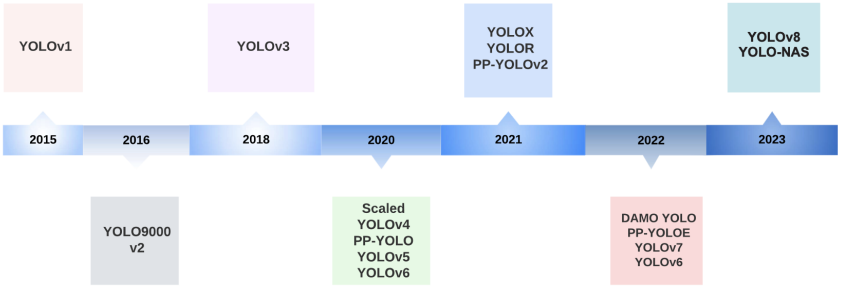


Figure 2.6: YOLO version timeline [36]

There have been several iterations of YOLO over the years, all updated with cutting-edge concepts this section will be explained. Figure 2.6 Figure shows how the YOLO family has evolved. The proposed models YOLOv5 and YOLOv8 will be explained in Chapter 3.

2.6.1 YOLO: You Only Look Once

The YOLO model predicts bounding boxes using P_c, which measures item presence and placement precision. The model rates its object location prediction accuracy with this confidence grade. Each bounding box has coordinates its center concerning the grid cell it lies in and dimensions, b_h and b_w, which indicate its height and width as proportions of the image. Non-maximum suppression eliminates multiple detections from the $S \times S \times (B \times 5 + C)$ tensor [37].

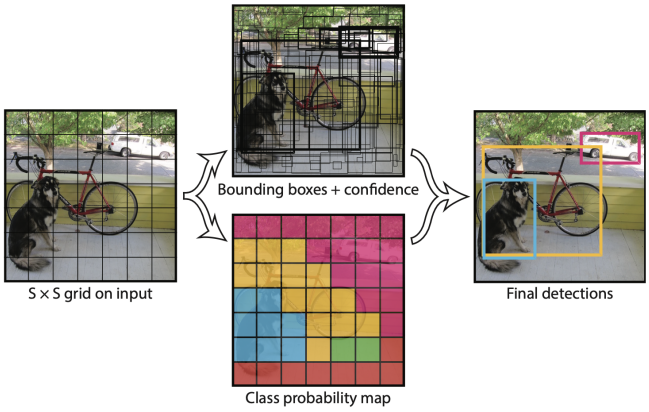


Figure 2.7: The system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B \times 5 + C)$ tensor [37].

Strengths and Limitations

YOLO has certain drawbacks while being a fast object detector. State-of-the-art techniques such as Fast RCNN have a less localization error than YOLO. There exist three primary factors for this limitation:

- This means that YOLO's predictive power is limited to detecting a maximum of two objects of the same type in a grid cell.
- Things with aspect ratios that weren't in the training set of data are hard for YOLO to forecast.
- The down-sampling approach allows YOLO to learn from coarse object features.

2.6.2 YOLOv2: Better, Faster, and Stronger

On the PASCAL VOC2007 dataset, the model known as YOLOv2 had an amazing average accuracy (AP) of 78.6%, surpassing the performance of its predecessor, YOLOv1, which only managed to obtain 63.4%. These results demonstrate the superior object identification and recognition capabilities of YOLOv2, opening new avenues for computer vision research in the future [38].

The system has undergone several enhancements. These enhancements involve implementing batch normalization for convolutional layers to enhance convergence and mitigate overfitting. An advanced classifier with high-resolution capabilities was incorporated, leading to improved performance while processing inputs with greater resolutions. The architecture was modified to incorporate fully convolutional layers, specifically utilizing a backbone known as DarkNet. DarkNet consists of 19 convolutional layers and 5 max-pooling layers. Additionally, anchor boxes are employed to forecast bounding boxes.

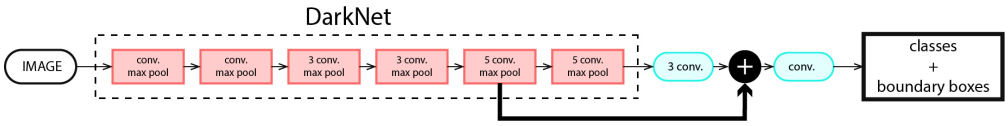


Figure 2.8: Darknet-19 simplified [39]

Ultimately, dimension clusters were incorporated to enhance the precision of predictions by utilizing appropriate priors identified by k-means clustering.

During the training process, the two datasets were merged in a way that allows the detection network to backpropagate when a detection training image is utilized. The utilization of a classification training picture results in the backpropagation of the classification component inside the architecture [38].

The outcome is a YOLO model with the ability to identify over 9000 categories, hence its designation as YOLO9000.

2.6.3 YOLOv3

YOLOv3 has significantly outperformed its predecessors in several parameters, most notably speed and accuracy. YOLOv3 has demonstrated rapid identification times on the COCO dataset, which is comparable to the accuracy of SSD models but around three times faster, indicating a significant improvement in real-time object detection capabilities. At a 0.5 IOU threshold, it achieves a mAP of 57.9%, demonstrating competitive performance versus other cutting-edge models such as RetinaNet [40].

Yolov3 brings several improvements that help it perform better. With 106 layers, it has a deeper and more intricate architecture that allows for more precise feature extraction. To enhance the detection of items with diverse sizes, this model also includes multiscale predictions, which employ three distinct scales. Furthermore, by employing three different kernel sizes for detection 13x13, 26x26, and 52x52 YOLOv3 increases its accuracy across a range of object scales and improves its adaptability in a variety of detection scenarios [41].

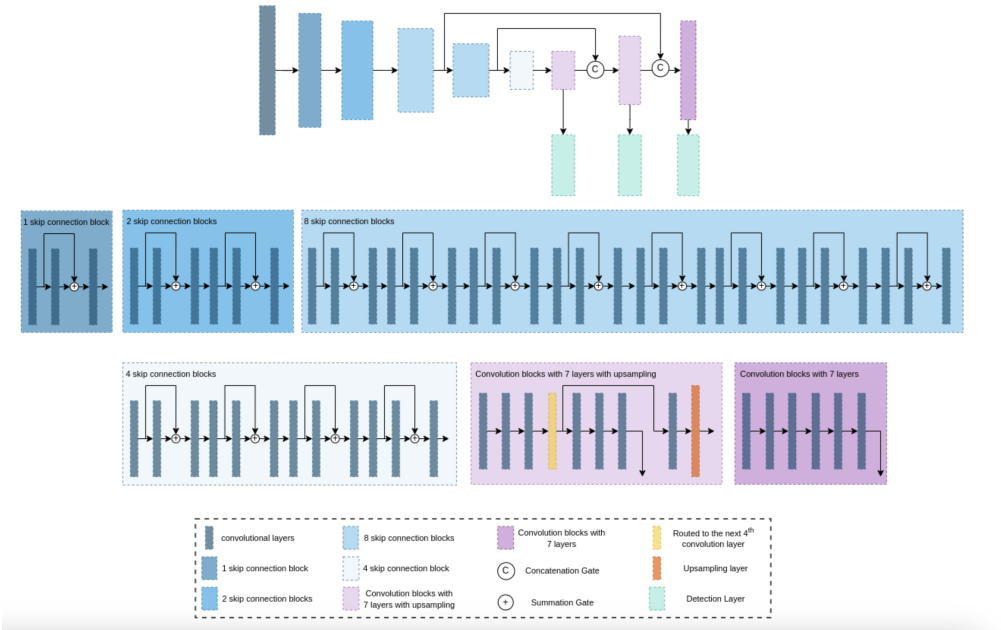


Figure 2.9: Shows the original YOLOv3 architecture [42].

All things considered, YOLOv3 stands out with its real-time object identification speed and precision, making it a strong option for applications that demand quick and accurate visual data interpretation.

2.6.4 YOLOv4 - High-Speed and Precise Object Detection

By enhancing the capabilities of its forerunners to achieve high speed and accuracy, YOLOv4 has significantly advanced the field of real-time object detection. Several additional characteristics are integrated into this model to optimize efficiency while avoiding an increase in processing loads during inference. YOLOv4, for example, makes use methods such as Mish-activation, Cross-Stage Partial connections, and Weighted-Residual Connections to improve the model's efficiency and learning capabilities.

YOLOv4 exhibits a robust performance while providing balance between speed and accuracy. It uses the CSPDarknet53 as its skeleton, PANet as its neck, and the YOLOv3 detection methods as its head. Its remarkable performance on object identification tasks makes it appropriate for real-time applications. These components all play a part in this. In addition to demonstrating exceptional speed in detection, YOLOv4 outperforms YOLOv3 in terms of accuracy and processing time, attaining cutting-edge outcomes in benchmarks.

The "bag of freebies," or tactics used in the training phase to improve accuracy without affecting the model's runtime performance, are what set YOLOv4 apart in practical applications. The robustness and scenario-generalizability of the model are greatly enhanced by data augmentation, which includes geometric and photometric aberrations. Through its GitHub repository, developers can access resources and instructions for the implementation or further development of YOLOv4. This comprehensive guide covers setup, training, and deployment on conventional GPUs, making it easier for YOLOv4 to be adopted in a variety of applications, ranging from personal projects to large-scale systems [43] [44].

2.6.5 YOLOv6

Impressively performing on multiple measures, YOLOv6 is a huge step forward in the evolution of the YOLO object identification models. With its elaborate speed and accuracy optimizations, YOLOv6 is especially well-suited for real-time industrial applications.

When it comes to performance, YOLOv6 models outperform their predecessors in terms of accuracy and efficiency. For example, when tested on the COCO dataset using FP16 precision on NVIDIA T4 GPUs, the YOLOv6-S model obtains a mAP of 43.5% at a processing speed of 495 frames per second (FPS). In comparison to previous iterations such as YOLOv5, this represents a new state-of-the-art for the series [45].

Many architectural improvements are also included in YOLOv6, such as the utilization of a single-path network with the increased model capacity to better balance detection accuracy and processing load. Using RepVGG blocks for effective feature processing and a Cross Stage Partial (CSP) link to increase efficiency without adding too much computational overhead are two noteworthy aspects [46].

YOLOv6 has variations for real-world applications, such as the YOLOv6-Nano and YOLOv6-Small, which are made for settings with constrained computing resources but still need good performance. These models maintain extremely low latency while achieving competitive mAP scores, which is important for applications like embedded or mobile devices that demand quick processing times [47] [48].

These improvements combine high accuracy with the efficiency required for deployment in a variety of operating scenarios, making YOLOv6 a strong option for developers and researchers working on cutting-edge object identification tasks.

2.6.6 YOLOv7

YOLOv7 is a powerful tool for real-time object identification in settings with constrained computational resources since it has been skillfully tailored for usage on edge devices. Specifically, the YOLOv7-tiny variation uses architectures and activation functions (such as leaky ReLU) appropriate for lightweight and mobile computing devices and is optimized to execute efficiently on edge GPUs. Because of this, it is very suitable for dispersed edge servers and devices, guaranteeing quick and precise object identification without requiring a lot of hardware [49].

Moreover, YOLOv7's usefulness on edge devices like the Nvidia Jetson Nano is demonstrated by its implementation in real-world applications like crop disease detection. The computational burden is greatly decreased by improving YOLOv7 with lightweight network architectures like GhostNetV1 and using channel pruning. As a result, the model's parameter size is significantly reduced while maintaining excellent accuracy (88.6%) and high detection speeds (up to 217 frames per second) [50].

These modifications highlight YOLOv7's ability to satisfy the demanding specifications of edge computing, offering a reliable answer for applications requiring real-time processing in settings with limited computational resources.

2.6.7 Summary

By using different frameworks, adopting different backbones, or integrating anchor boxes, each version aims to optimize a certain aspect of performance. Average precision (AP) and processing speed increase gradually as a result of advancements in network architecture and training methodologies. This is indicative of the continuous progress in object detection technology.

Version	Date	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	63.4
YOLOv2	2016	Yes	Darknet	Darknet24	78.6
YOLOv3	2018	Yes	Darknet	Darknet53	33.0
YOLOv4	2020	Yes	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Yes	Pytorch	YOLOv5CSPDarknet	55.8
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	56.8
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	53.9

Table 2.1: YOLO Versions and Their Performance [36]

From YOLO to YOLOv8, the YOLO object detection models have progressed greatly. Each iteration has addressed unique challenges and improved its predecessors.

The 2015 YOLO model was groundbreaking in speed but lacking in precision compared to existing detectors. This method treats object recognition as a single regression problem, using picture pixels to predict bounding box coordinates and class probabilities. This is innovative.

Later versions, YOLOv2 and YOLOv3, improved accuracy. YOLOv2 used anchor boxes to predict bounding box offsets, while YOLOv3 was detected at three scales to improve accuracy.

In YOLOv5, Darknet was replaced by PyTorch, improving speed and accuracy. Modern neural network features and optimization methods were added.

YOLOv8, the latest version, continues to improve. The algorithm uses compound scaling and enhanced data augmentation to handle images with different object sizes and shapes.

While PP-YOLO, YOLOX, and YOLO-NAS were produced alongside the core YOLO series, they prioritized factors like model efficiency for certain platforms or neural architecture search for backbone design. This thesis solely compares the primary versions from YOLO to YOLOv8 in consecutive order to ensure a thorough and consistent development comparison. YOLO has become a popular computer vision framework by improving object identification speed and accuracy with each iteration [36].

In Chapter 3, the focus is on the proposed models based on the YOLOv5 and YOLOv8 architectures.

3

Methodology

3.1 Proposed Models: YOLOv5 & YOLOv8 for Aerial Image Object Detection

YOLOv5 and YOLOv8 were chosen as the object detection algorithms for the Stanford and VisDrone datasets due to their advanced features and impressive performance.

YOLOv5 provides a good balance between speed and accuracy, is user-friendly, has a flexible training process [51], and includes unique features like Mosaic data augmentation and auto-learning bounding box anchors [52].

YOLOv8, being the latest in the YOLO series, uses cutting-edge technology, has an improved design, better training methods, and can adapt to various model sizes. These models can effectively address the unique challenges presented by the datasets, such as varying object sizes, complex backgrounds, and diverse environmental conditions [53] [40].

The application of YOLOv5 and YOLOv8 in this research aims to improve the accuracy and efficiency of object detection tasks, ultimately leading to enhanced overall performance of the detection systems utilized for these datasets.

3.1.1 YOLOv5

In June 2020, Ultralytics introduced YOLOv5, an updated object detection network framework modeled similarly to YOLOv4. This version features enhancements in both speed and accuracy of detection, along with a noticeably smaller model size. YOLOv5 has garnered significant attention across both industry and academic circles.

YOLOv5 is an extremely popular single-stage object detection model that comes in four different sizes: YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra large). As the size of the model increases from YOLOv5s to YOLOv5x, both the depth and width of the model also increase, leading to a higher number of parameters. Glen Jocher oversees the customization of the model by adjusting the `depth_multiple`, `width_multiple` parameters to accommodate various detection needs [54]. YOLOv5's architecture is divided into four main parts: the input, the backbone network model, the neck network model, and the output. The backbone consists of a convolutional neural network that processes detailed images to create feature maps. The neck network integrates these features from the backbone and passes them to the output, which handles the detection and classification tasks.

The layout of the YOLOv5s model is depicted in Figure 3.1.

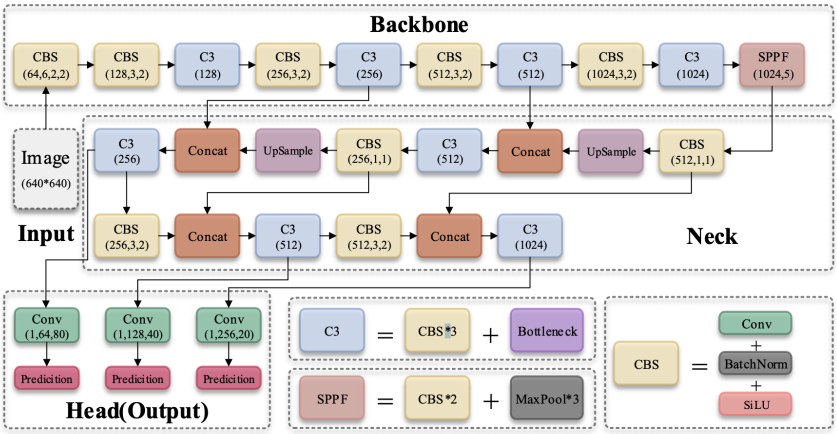


Figure 3.1: Network structure of YOLOv5 [55].

Input: YOLOv5 implements mosaic data augmentation on the input side and includes built-in adaptive anchoring and scaling features. This mosaic augmentation involves randomly selecting four images, and merging them into a single training image, which effectively reduces the pixel size of the detection targets and enhances YOLOv5's ability to detect smaller objects (Figure 3.2). After entering the network, the dataset images are resized to 640x640 pixels. YOLOv5 then applies the k-means clustering algorithm to determine anchor boxes that best match the annotations in the dataset. These are compared with preset anchor boxes. If the best recall rate exceeds 0.98, it indicates that the preset anchor sizes are adequate for the dataset. Otherwise, network parameters are adjusted accordingly [56].

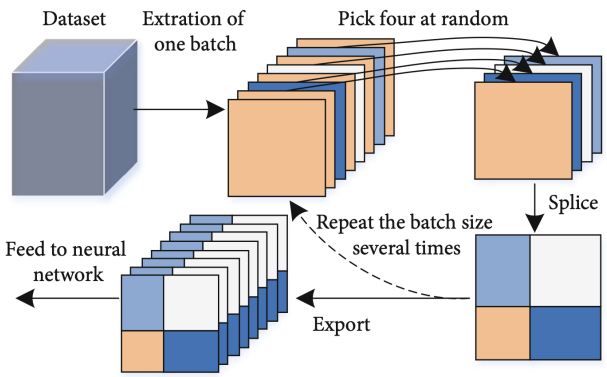


Figure 3.2: Mosaic data augmentation flow [57].

Backbone: In its backbone architecture, YOLOv5 incorporates FOCUS, SPP, and CSP techniques. The Focus method performs a slicing operation on each pixel in the image, similar to local downsampling, which quadruples the input channel without losing information. The modified image is then processed to produce a bipartite downsampled feature map that preserves data integrity. Additionally, YOLOv5 utilizes the CSPNet residual framework(Figure 3.3) in both the backbone and neck parts of the network. This framework splits the feature map of the basic layer into two segments and merges them across different stages, thereby minimizing computational load while maintaining the full integrity of the feature information [56].

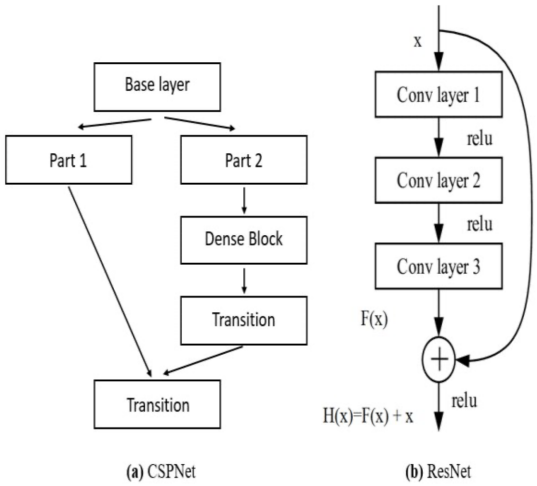


Figure 3.3: Structure of (a) YOLOv4 and YOLOv5 backbone draws from CSPNet (b) ResNet [58].

Neck: In its neck architecture, YOLOv5 integrates both FPN and Path Aggregation Network (PAN) technologies (Figure 3.4). The shallow feature maps in this part of the network contain richer location data but less semantic information. As layers in the neural network increase, deeper feature maps develop more semantic content while potentially losing small pixel detail and some positional data. Nonetheless, retaining both location and semantic information is crucial for effective target detection, necessitating a deeper network structure that balances both. The FPN technology transfers rich semantic details from the upper feature maps to the lower ones, whereas PAN enhances the transmission of localization details from lower to upper feature maps. Together, these frameworks significantly improve the capability of feature fusion in the neck part of the network model [56].

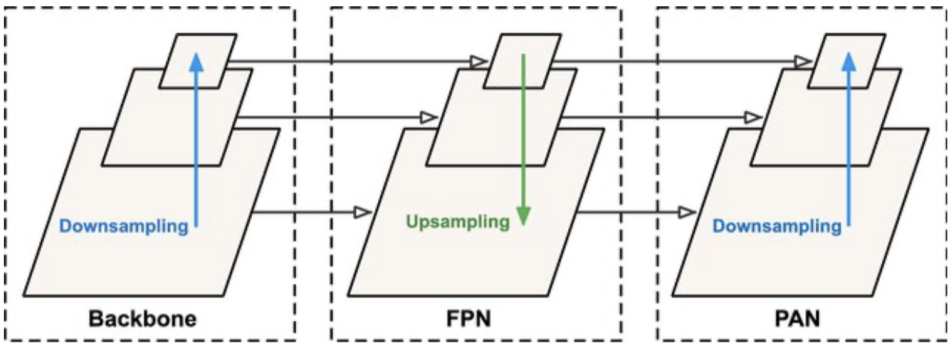


Figure 3.4: The structure of FPN and PAN in YOLOv5 [59].

3.1.2 YOLOv8

Launched by Ultralytics in January 2023, YOLOv8 is the most recent progression in the YOLO series, improving upon the capabilities of predecessors such as YOLOv5 and YOLOv7. This model is acclaimed for its exceptional detection speed and accuracy, achieved through a variety of architectural innovations. Its structure is composed of three main sections: the backbone, neck, and head, each specifically tailored to boost performance across different scales and use cases (Figure 3.5).

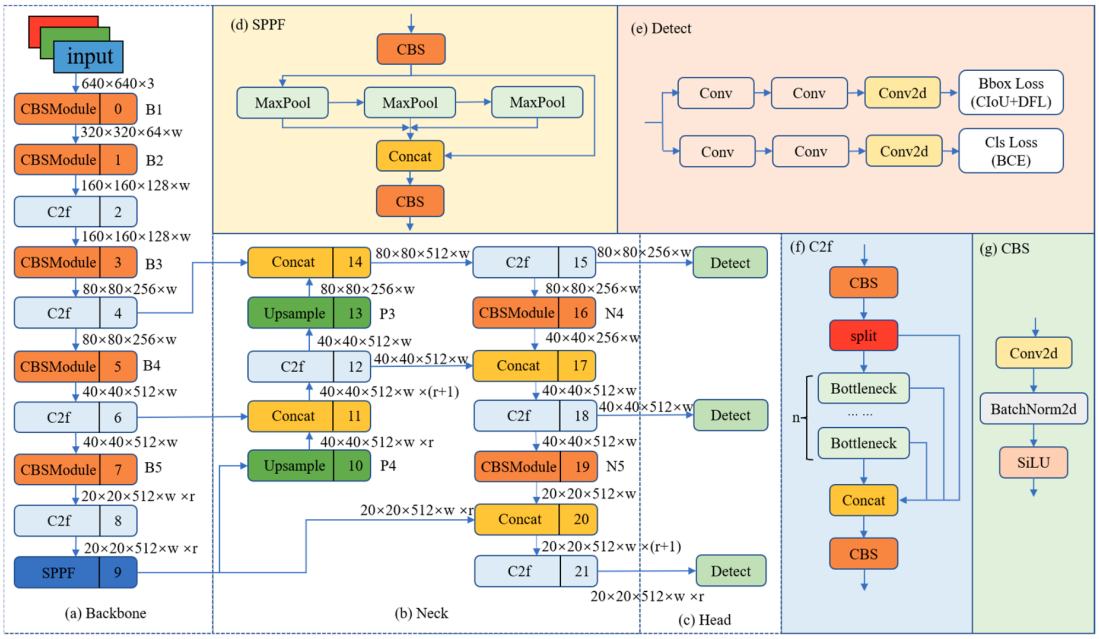


Figure 3.5: The architectural design of YOLOv8 [60].

The **backbone** of YOLOv8 incorporates an advanced CSPDarknet53 architecture enhanced by the C2f module from the ELAN design used in YOLOv7. This module introduces gradient shunt connections that support the efficient flow of information while preserving a compact structure. It enriches the gradient flow, significantly improving feature extraction capabilities. The backbone concludes with the Spatial Pyramid Pooling Fast (SPPF) module, a refined version of the traditional SPP that reduces latency with sequential maximum pooling layers [61].

In the **neck**, YOLOv8 combines a PAN-FPN structure, which refines traditional FPN architectures by integrating PAN to enhance the integration of top-down and bottom-up features. This configuration ensures a balanced integration of deep semantic and precise positional information, effectively mitigating object localization errors often seen in standard FPN setups [60].

The **head** of YOLOv8 adopts a decoupled approach, separating object classification and bounding box regression into distinct pathways, each optimized with specific loss functions binary cross-entropy for classification and a combination of distribution focal loss and CIoU for box regression. This specialized segmentation not only speeds up model convergence but also increases the accuracy of object detections [60].

YOLOv8 significantly advances object detection technology by integrating key features from previous versions with new structural enhancements. This model uses an anchor-free design and a dynamic task-aligned assigner that optimizes sample distribution during training, improving robustness and precision. Its adaptability ensures compatibility with various hardware setups, including Central Processing Unit (CPU) and Graphics Processing Unit (GPU) [61].

Optimized for real-time processing, YOLOv8 is suited for diverse computer vision and AI applications. It employs a multi-scale approach to handle varying object sizes effectively and is noted for its precise bounding box predictions, crucial for detailed tracking and localization tasks [61] [60].

Overall, YOLOv8's adaptable network and advanced modules demonstrate its efficiency and broad applicability, redefining performance standards in object detection.

3.2 Datasets Overview

3.2.1 Stanford Drone Dataset

The Stanford Drone Dataset (SDD) [32], created by the Computer Vision and Geometry Laboratory (CVGL) at Stanford University, encompasses approximately 60 high-resolution aerial videos (1400 × 1904 pixels), taken from eight diverse outdoor locations across the Stanford campus. Captured by drones operating at an altitude of 80 meters and equipped with 4K cameras, the dataset notably focuses on dynamic entities such as pedestrians, bicyclists, cars, skaters, carts, and buses.

Renowned for its extensive annotation set, the SDD contains over 20,000 detailed annotations for a variety of objects, including around 11,200 pedestrians, 6,400 bicyclists, 1,300 cars, 300 skateboarders, 200 golf carts, and 100 buses. These annotations are essential for tasks like human trajectory prediction and multi-object tracking in crowded settings. The bounding boxes provided for each object type such as "Pedestrian," "Biker," "Skateboarder," "Cart," "Car," and "Bus," particularly emphasize the prevalence of pedestrians and bikers, who make up approximately 85%-95% of these annotations [32].

Scenes	Videos	Bicyclist %	Pedestrian %	Skater %	Cart %	Car %	Bus %
gates	9	51.94	51.94	2.55	0.29	1.08	0.78
little	4	56.04	42.46	0.67	0	0.17	0.67
nexus	12	4.22	64.02	0.60	0.40	29.51	1.25
coupa	4	18.89	80.61	0.17	0.17	0.17	0
bookstore	7	32.89	63.94	1.63	0.34	0.83	0.37
deathCircle	5	56.30	33.13	2.33	3.10	4.71	0.42
quad	4	12.50	87.50	0	0	0	0
hyang	15	27.68	70.01	1.29	0.43	0.50	0.09

Table 3.1: Distribution of Categories Across Scenes from the Stanford Drone Dataset with All Object Categories

(a) (b)

Figure 3.6: Two sample images from the Stanford dataset [32].

The dataset is organized into eight zones based on geographical campus locations, enhancing its utility for studying object interactions and social behaviors, which are key areas in object detection, tracking, and traffic surveillance. This structured organization not only supports academic and research initiatives but also extends to practical applications such as traffic monitoring and security measures. The dataset offers rich action trajectories and detailed identifiers for each object type, making it a valuable resource for advancing computer vision and artificial intelligence technologies.

The SDD is publicly available through Stanford University [32] and features visuals from significant campus locations like the Gates and Death Circle scenes, demonstrating its practical utility and broad applicability (see Figure 3.6). After downloading, the dataset is organized into two primary folders: annotations and videos. Each contains eight subfolders corresponding to different scenes, where the videos or annotations are categorized based on their parent directory [62].

```
/
├── videos
│   ├── bookstore ..... 7 videos .mov
│   ├── coupa ..... 4 videos .mov
│   ├── deathCircle ..... 5 videos .mov
│   ├── gates ..... 9 videos .mov
│   ├── hyang ..... 15 videos .mov
│   ├── little ..... 4 videos .mov
│   ├── nexus ..... 12 videos .mov
│   └── quad ..... 4 videos .mov
└── annotations
    ├── bookstore ..... 7 .txt annotations + reference.jpg
    ├── coupa ..... 4 .txt annotations + reference.jpg
    ├── deathCircle ..... 5 .txt annotations + reference.jpg
    ├── gates ..... 9 .txt annotations + reference.jpg
    ├── hyang ..... 15 .txt annotations + reference.jpg
    ├── little ..... 4 .txt annotations + reference.jpg
    ├── nexus ..... 12 .txt annotations + reference.jpg
    └── quad ..... 4 .txt annotations + reference.jpg
```

Each video within the `videos` directory corresponds to a specific scene and is paired with an annotation file (`annotation.txt`) and an example frame (`reference.jpg`) located in the `annotations` directory. The following details outline the structure and contents of each entry in the `annotations.txt` file, where each line represents a unique annotation and consists of ten or more columns separated by spaces:

- **Track ID**: All rows sharing the same ID correspond to the same trajectory.
- **xmin**: The x-coordinate of the top left corner of the bounding box.
- **ymin**: The y-coordinate of the top left corner of the bounding box.
- **xmax**: The x-coordinate of the bottom right corner of the bounding box.
- **ymax**: The y-coordinate of the bottom right corner of the bounding box.
- **frame**: Indicates the frame number that this annotation belongs to.
- **lost**: If set to 1, indicates that the annotation is outside the view screen.
- **occluded**: If set to 1, indicates that the annotation is obstructed from view.
- **generated**: If set to 1, indicates that the annotation was interpolated automatically.
- **label**: The label of the object being annotated, enclosed in quotation marks.

3.2.2 Stanford Drone Dataset with Grouped Object

The modified version of the Stanford Drone Dataset(SDD) has reorganized object categories to improve object detection and tracking in aerial imagery. The updated structure groups similar objects, reducing the complexity of detection tasks and potentially speeding up computational algorithms. Key groupings include:

- **Pedestrians** with **Skaters** due to similar sizes and motion patterns.
- **Cars**, **Carts**, and **Buses** into one category, reflecting their common features from an aerial view.
- **Bicyclists** remain a separate category due to unique characteristics.

This restructuring aims to cut down inference or testing time by simplifying object classification, especially in dynamic settings like university campuses where the dataset was originally gathered. This should enhance processing efficiency and accelerate practical application response times.

Scenes	Videos	Bicyclist %	Pedestrian %	Car %
gates	9	51.94	54.49	2.15
little	4	56.04	43.13	0.84
nexus	12	4.22	64.62	31.16
coupa	4	18.89	80.78	0.34
bookstore	7	32.89	65.57	1.54
deathCircle	5	56.30	35.46	8.23
quad	4	12.50	87.50	0.00
hyang	15	27.68	71.30	1.02

Table 3.2: Distribution of Grouped Object Categories in Different Scenes from the Stanford Drone Dataset

Table 3.2 displays the distribution of revised categories across scenes post-reorganization, streamlining classification for faster processing and broader applicability in fields like autonomous navigation. This simplification boosts data handling and enhances trajectory prediction and object tracking. Section 4 evaluates the effectiveness of this method.

3.2.3 VisDrone Dataset

The VisDrone Dataset, developed by the AISKYEYE [53] team from the Lab of Machine Learning and Data Mining at Tianjin University, China, is a comprehensive benchmark designed for drone-based image and video analysis. This extensive dataset includes 288 video clips that sum up to 261,908 frames, alongside 10,209 static images captured using diverse drone-mounted cameras.

Class	All	Train	Val
Pedestrian	79,337	8,844	88,181
People	27,059	5,125	32,184
Bicycle	10,480	1,287	11,767
Car	144,867	14,064	158,931
Van	24,956	1,975	26,931
Truck	12,875	750	13,625
Tricycle	4,812	1,045	5,857
Awning-tricycle	3,246	532	3,778
Bus	5,926	251	6,177
Motor	29,647	4,886	34,533

Table 3.3: VisDrone2019 class-wise distribution [63].

The VisDrone dataset spans 14 cities in China, covering urban to rural settings and featuring a diverse range of objects including pedestrians, vehicles, and bicycles in scenes from sparse to densely populated. Compiled from multiple drone platforms under different weather and lighting conditions, it comprises over 2.6 million manually annotated bounding boxes. The dataset includes attributes like scene visibility, object classification, and occlusion, making it valuable for advanced computer vision applications [31].

(a) (b)

Figure 3.7: Two sample images from the VisDrone dataset [31].

The VisDrone2019-DET dataset comprises three main directories, each tailored for different phases of model development: training, validation, and testing. The structure ensures organized access to:

```
VisDrone2019-DET
├── VisDrone2019-DET-train
│   ├── images ..................................... 6471 images.jpg
│   └── annotations ................................ 6471 texts.txt
├── VisDrone2019-DET-val
│   ├── images ..................................... 548 images.jpg
│   └── annotations ................................ 548 texts.txt
└── VisDrone2019-DET-test
    ├── images ..................................... 1610 images.jpg
    └── annotations ................................ 1610 texts.txt
```

The Visdrone dataset uses a specific format for labels in its annotation files. Each label includes the following:

- **bbox_left**: X-coordinate of the bounding box's upper-left corner.
- **bbox_top**: Y-coordinate of the bounding box's upper-left corner.
- **bbox_width**: Width of the bounding box in pixels.
- **bbox_height**: Height of the bounding box in pixels.
- **score**: Confidence level of the bounding box; 1 in ground truth indicates evaluation, 0 indicates ignored.
- **object_category**: Type of object, ranging from 0 (ignored regions) to 11 (various categories including pedestrians).
- **truncation**: Indicates the extent of truncation; 0 for none, 1 for 1% to 50% truncation in ground truth.
- **occlusion**: Degree of occlusion; 0 for none, 1 for 1% to 50%, and 2 for over 50% in ground truth.

This format helps ensure clarity and consistency in how objects are annotated within the Visdrone dataset, facilitating ease of use and understanding for researchers and practitioners.

3.2.4 VisDrone Dataset with Grouped Object

The VisDrone2019 dataset has implemented a grouping strategy akin to that used in the Stanford Drone Dataset, merging its ten original categories into three broad groups to facilitate tasks like object detection and tracking in aerial images.

The revised category groups are as follows:

- **Pedestrian**: Merges "Pedestrian" and "people," reflecting their similarities in urban environments.
- **CycleVariants**: Groups "bicycle," "tricycle," "awning-tricycle," and "motor," consolidating all two- and three-wheeled vehicles based on size and mobility.
- **AutoMobiles**: Combines "car," "van," "truck," and "bus," unifying these vehicles by common usage and visual characteristics from an aerial perspective.

Class	All	Train	Val
Pedestrian	106,396	13,969	120,365
CycleVariants	48,185	7,750	55,935
AutoMobiles	188,624	17,040	205,664

Table 3.4: VisDrone dataset class-wise distribution after grouping

This reorganization streamlines classification reduces computational demands, and facilitates the training process for machine learning models. By generalizing object categories, the updated VisDrone2019 dataset aims to enhance model performance in varied operational tasks involving different vehicle types and pedestrian movements, optimizing it for aerial surveillance and autonomous navigation applications.

3.3 Preprocessing and Data Organization

3.3.1 Dataset Organization

Stanford Drone Dataset

The dataset presents three primary issues:

1. The directory structure is not properly organized for implementing object detection algorithms.
2. The dataset consists solely of videos; object detection models require individual frames for training.
3. The existing annotation format does not meet the specifications needed for use with Yolov5 and Yolov8.

To enhance accessibility, videos, and annotations were reorganized into dedicated folders and renamed following the format: `[sceneName]_video[number].mov/txt`. This naming strategy speeds up the retrieval of both videos and annotations since they have matching names but are located in distinct folders [62].

VisDrone2019-DET Dataset

The dataset presents three primary issues:

1. The annotations are complex and require conversion to be used with popular detection frameworks like Yolov5 and Yolov8.
2. The dataset has an unbalanced class distribution, leading to potential biases in model training.
3. High levels of occlusion and object crowding in urban scenes complicate accurate object detection.

3.3.2 Frames Extraction

Stanford Drone Dataset

To facilitate training and detection with object detection algorithms, which require images, frames were initially extracted from each video. Using a Python library called `CV2`, we extracted and organized these frames into specific folders labeled for their intended use: train, validation, or test. Each frame was saved under a unique naming format:

```
[sceneName]_video[Number]_[frameNumber].jpg
```

VisDrone2019-DET Dataset

For the VisDrone2019 dataset, frame extraction is not required as the dataset directly provides still images ready for analysis.

3.3.3 Annotation Format

YOLOv5 and YOLOv8 necessitate that annotations adhere to the Darknet format. The specific requirements include:

- Each image has a separate .txt file containing labels.
- Each object in an image has its own row in the corresponding .txt file.
- The row format is: class_index, bbox_x_center, bbox_y_center, bbox_width, bbox_height.
- Bounding box coordinates are normalized between 0 and 1 [62].

To transform the original annotations into this format, specific algorithms have been developed. These algorithms process a single line from the original annotation file along with the frame's height and width to normalize the bounding box coordinates.

```python
# Calculate the normalized center of the BB
def centerBB(row, heightIm, widthIm):
    x = ((row['xmax'] + row['xmin']) / 2) / widthIm
    y = ((row['ymax'] + row['ymin']) / 2) / heightIm
    return str(x), str(y)

# Calculate the normalized width and height of the BB
def dimBB(row, heightIm, widthIm):
    widthBB = (row['xmax'] - row['xmin']) / widthIm
    heightBB = (row['ymax'] - row['ymin']) / heightIm
    return str(widthBB), str(heightBB)
```

Stanford Drone Dataset

Each annotation in the Stanford Drone Dataset is stored within a dedicated folder according to its designated purpose: train, validation, or test. Frames are uniquely named and saved in a structured format, following the convention: [sceneName]_video[Number]_[frameNumber].txt.

After completing the preprocessing, the directory structure appears as follows [62]:

```
├── videos_only
├── annotations_only
└── video_frames
    ├── images
    │   ├── train ..............................all frames for training
    │   ├── validation .........................all frames for validation
    │   └── test ...............................all frames for test
    └── labels
        ├── train ..........................all annotations for training
        ├── validation .................all annotations for validation
        └── test ..............................all annotations for test
```

VisDrone2019-DET Dataset

In the VisDrone dataset, annotations are organized within specific folders corresponding to their intended usage: training, validation, or testing. The annotations have been uniformly transformed into the YOLO format, ensuring consistency across the dataset. Each image's annotations retain their original filenames, maintaining alignment with the corresponding image files. This systematic approach facilitates efficient data management and compatibility with YOLO-based object detection frameworks.

Stanford Drone Dataset annotation:

ID	xmin	ymin	xmax	ymax	frame	lost	occluded	generated	label
0	1378	804	1418	858	9500	1	0	0	"Biker"

VisDrone2019-DET Dataset annotation:

bbox_left	bbox_top	bbox_width	bbox_height	score	object_category	truncation	occlusion
303	0	130	81	1	10	0	1

YOLOv5 and YOLOv8 annotation:

class_index	bbox_x_center	bbox_y_center	bbox_width	bbox_height
0	0.8731	0.6781	0.0345	0.0545

Figure 3.8: Comparison between different annotation formats

3.4 Training YOLOv5 and YOLOv8 Algorithms

The YOLOv5 and YOLOv8 algorithms underwent the same training procedures with identical datasets to guarantee performance comparability.

3.4.1 Dataset Utilization

The **Stanford Drone Dataset** and its grouped objects variant were used, with:

- 4,080 images for training,
- 1,020 images for validation,
- 2,000 images for testing.

Images were evenly extracted from videos to ensure uniformity across all datasets.

The **VisDrone Dataset** and its grouped version included:

- 6,471 images for training,
- 548 images for validation,
- 1,610 images for testing,

forming a comprehensive base for assessing the object detection algorithms.

3.4.2 Hardware Configuration

Training was supported by Google Colab Pro's high-performance hardware:

- **GPU:**The Nvidia Tesla P100 GPU, which has a significant computational capacity, is crucial for the demanding calculations required in deep learning.
- **CPU and RAM:** High-performance CPUs and 25 GB of Random Access Memory (RAM) were essential for managing large datasets and complex computational tasks.

3.4.3 Training with YOLOv5

For the training of each YOLO model, the parameters were largely kept to the defaults recommended by the official repository. Initially, a selection was made from various pre-trained models, each of which had been initially trained on the COCO dataset. These models were differentiated by their unique architectural logic, which is illustrated through the varied performances depicted in Figure 3.9.

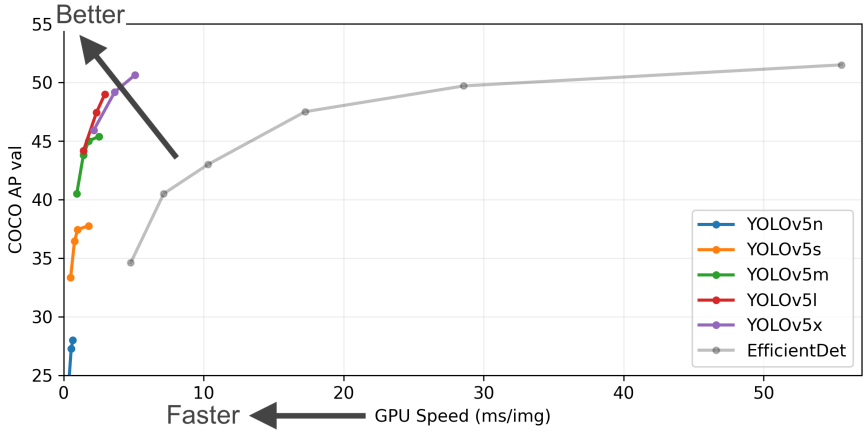


Figure 3.9: Performance of YOLOv5 pre-trained models [51] .

For this project, three variants of the YOLOv5 algorithm were selected based on their complexity and number of parameters: **YOLOv5s** with 7.2 million parameters, **YOLOv5m** with 21.2 million parameters, and **YOLOv5x** with 89 million parameters. To ensure uniformity in experimental conditions, the same training parameters were applied across all datasets. For the **VisDrone Dataset**, the training was initiated using the `train.py` script.

```
!python train.py --epochs 100 --data ../VisDrone_Dataset/VisDrone.yaml \
--cfg ./models/yolov5s.yaml --weights yolov5s.pt \
--project ../drive/MyDrive/all_result/VisDrone_Dataset_Result \
--name VisDrone_Dataset_yolov5s --cache
```

The following parameters were passed to the training script:

- **epochs**: The model was trained for 100 epochs.
- **data**: Path to the dataset configuration file.
- **cfg**: Path to the model configuration file.
- **weights**: Use of pre-trained weights from the YOLOv5s model.
- **project**: Destination directory for saving training checkpoints.
- **name**: Designation for the model training session.
- **cache**: Enables caching of dataset images to expedite training.

Default parameters used include:

- **img**: Images were resized to 640x640 pixels.
- **batch**: The training process utilized batches of 16 images.

In the next section are summarized all the results.

3.4.4 Training with YOLOv8

Initially, a variety of pre-trained models were chosen, each previously trained on the COCO dataset. These models varied in their architectural designs, leading to distinct performances, as demonstrated in Figure 3.10.

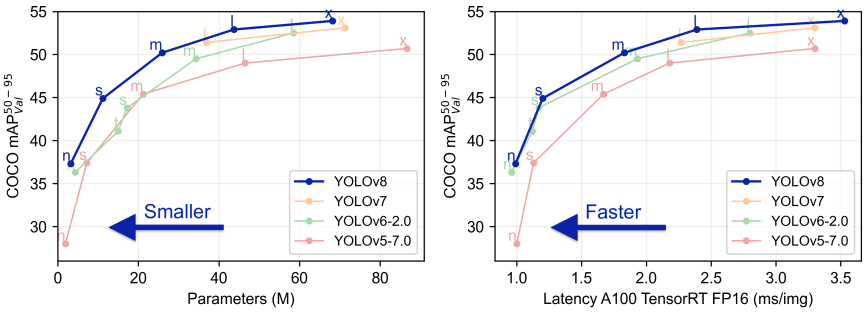


Figure 3.10: Performance of YOLO models [52] .

For this investigation, four versions of the YOLOv8 model were employed, differentiated by their complexity and parameter count:

- **YOLOv8n** with 3.2 million parameters,
- **YOLOv8s** with 11.2 million parameters,
- **YOLOv8m** with 25.9 million parameters,
- **YOLOv8x** with 68.2 million parameters.

Consistent training parameters were maintained across all datasets to standardize experimental conditions. Training for the **VisDrone Dataset** commenced with the execution of the `train.py` script.

```
!yolo train epochs=100
data=../VisDrone_Dataset/VisDrone.yaml
model=yolov8n.pt
name=VisDrone_Dataset_yolov8n
project=../drive/MyDrive/all_result/VisDrone_Dataset_Result
```

The training of the YOLOv8 model was initiated with the following command, specifying the desired parameters for the training session:

- **epochs**: Specifies that the model should be trained for 100 epochs.
- **data**: Provides the path to the dataset configuration file.
- **model**: The pre-trained model file used, `yolov8n.pt`, indicating use of the YOLOv8n variant.
- **name**: The designation for the training session, was initially mislabeled.
- **project**: The directory where training checkpoints will be saved.

Additionally, some default parameters are used implicitly unless overridden:

- **img**: Images are resized to 640x640 pixels for processing.
- **batch**: The batch size is set to 16 images per batch, which optimizes GPU usage during training.

3.5 Training Results

3.5.1 Comparison of YOLO models on different datasets

The evaluation of the YOLO models (YOLOv5 and YOLOv8) on different datasets VisDrone-2019, VisDrone-2019 with Grouped Objects, Stanford Drone, and Stanford Drone with Grouped Objects provides a comprehensive analysis of their performance in terms of precision (P), recall (R), mAP at IoU threshold 0.5 (mAP@0.5), and mAP at IoU threshold range 0.5 to 0.95 (mAP@0.5-0.95). The results from these evaluations are summarized in the following table (Table 3.5), with the highest values in each category bolded to highlight the best-performing configurations.

Datasets	**Result**	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
VisDrone-2019	P(%)	30.3	38.4	54.8	44.9	49.9	55.4	57.7
	R(%)	23.4	37.1	40.2	33.5	39.3	43.0	43.7
	mAP@0.5 (%)	20.7	37.2	41.9	34.0	40.4	44.4	45.8
	mAP@0.5-0.95 (%)	9.9	21.6	25.1	19.7	24.3	27.0	28.2
VisDrone-2019 Grouped	P(%)	**47.6**	**65.0**	**69.0**	**72.7**	**63.8**	**70.0**	**72.1**
	R(%)	**33.4**	**48.7**	**53.2**	**55.8**	**47.2**	**53.0**	**56.0**
	mAP@0.5 (%)	**32.1**	**53.3**	**58.5**	**61.5**	**52.3**	**59.3**	**62.7**
	mAP@0.5-0.95 (%)	**15.4**	**27.7**	**31.9**	**35.0**	**28.4**	**33.2**	**37.9**
Stanford Drone	P(%)	62.1	68.7	78.4	70.0	75.9	**82.8**	76.6
	R(%)	56.3	59.7	66.7	60.9	65.2	65.1	70.6
	mAP@0.5 (%)	62.1	69.8	77.0	64.9	71.0	72.9	73.3
	mAP@0.5-0.95 (%)	28.9	36.7	**45.3**	**32.0**	**39.8**	**43.1**	**44.3**
Stanford Drone Grouped	P(%)	**77.5**	**80.1**	**82.4**	**72.7**	**77.6**	78.8	**77.2**
	R(%)	**69.1**	**73.5**	**76.8**	**66.2**	**70.6**	**72.1**	**75.0**
	mAP@0.5 (%)	**73.6**	**78.3**	**80.7**	**74.9**	**76.4**	**75.8**	**75.5**
	mAP@0.5-0.95 (%)	**32.4**	**38.5**	43.7	31.9	37.9	40.7	39.7

Table 3.5: Comparison of YOLO models on different datasets

3.5.2 Performance Analysis

1. VisDrone-2019 vs. VisDrone-2019 Grouped:

- **Precision (P):** Precision improved significantly in the VisDrone-2019 Grouped dataset, with YOLOv8n reaching the highest precision of 72.7%.
- **Recall (R):** Recall also saw improvements, with YOLOv8l achieving the top score of 56.0%.
- **mAP@0.5:** The YOLOv8l model exhibited the highest mAP at an IoU threshold of 0.5, registering 62.7%.
- **mAP@0.5-0.95:** Consistent enhancements were noted across the IoU spectrum, with YOLOv8l achieving the peak at 37.9%.

2. Stanford Drone vs. Stanford Drone Grouped:

- **Precision (P):** Precision markedly increased in the Stanford Drone Grouped dataset for nearly all models, with YOLOv8m recording the highest precision of 82.8%.
- **Recall (R):** Recall improved in the Grouped dataset as well, with YOLOv5x showing the highest recall of 76.8%.
- **mAP@0.5:** There was a notable rise in mAP, with YOLOv5x achieving the top mark of 80.7%.
- **mAP@0.5-0.95:** Significant enhancements were observed across the IoU range in the Stanford Drone dataset, with YOLOv5x attaining the highest mAP of 45.3%.

This comparative analysis indicates that grouping objects within the datasets leads to substantial improvements in precision, recall, and mAP across all evaluated YOLO models. YOLOv8n was particularly effective in the VisDrone-2019 Grouped dataset, while YOLOv5x demonstrated superior performance in the Stanford Drone Grouped dataset. These findings underscore the efficacy of YOLO models in handling grouped objects and provide critical insights for optimizing object detection tasks in diverse operational contexts.

Models	Dataset	Yolov5s	Yolov5m	Yolov5x	Yolov8n	Yolov8s	Yolov8m	Yolov8l
VisDrone-2019	Pedestrian	28	44.5	49.5	34.8	44.5	48.2	51
	People	23.5	33.6	39.3	28.1	33.7	37.2	39.6
	Bicycle	2.9	15.8	19.1	8	13.1	19.5	20.5
	Car	61.2	75.7	78.9	75.8	79.8	81.7	82.5
	Van	15.2	40.3	43.7	38.8	45.9	47.7	48.4
	Truck	14.5	36	39.1	32	40.2	44	42.5
	Tricycle	9.6	22.9	28.8	22.7	28.9	33.8	36.2
	Awning-Tricycle	5.4	12.6	16.2	12.4	16.3	17.3	18.8
	Bus	19.6	49.2	56.4	50.2	58.8	65.5	66.1
	Motor	27.2	40.9	47.6	37.1	44.3	49.7	52.7
VisDrone-2019 Grouped	Pedestrian	**43.3**	**49.3**	**53.5**	**40.1**	**48.2**	**54.2**	**56.7**
	CycleVariants	**40.1**	**46.4**	**49.7**	**37.6**	**43.3**	**49.3**	**51.4**
	Automobiles	**76.5**	**79.9**	**82**	**78.3**	**82.9**	**84.5**	**86.2**
Stanford Drone	Pedestrian	65.8	67.8	74.2	62.1	69	85.3	76.6
	Biker	64.3	67.4	66.5	61.5	65.7	65.2	70.6
	Skater	23.8	26.4	28.9	26.7	27.4	28.1	29.4
	Cart	25.4	28.8	26.7	26.1	26.8	27.5	29
	Car	76.2	79.8	87.4	82.3	85.3	85.5	83.2
	Bus	76.9	79.9	87.2	80.2	84.3	85.1	83.5
Stanford Drone Grouped	Pedestrian	**67**	**72.6**	**76.4**	**62.6**	**69.7**	**71.9**	**70.8**
	Biker	**66.3**	**71.8**	**74.1**	**62.6**	**67.2**	**68.5**	**69.4**
	Car	**87.4**	**90.6**	**91.7**	**84.8**	**87.9**	**88.7**	**89.1**

Table 3.6: Comparison of YOLO models on class-wise different datasets

3.5.3 Class-Wise Analysis

1. VisDrone-2019 vs. VisDrone-2019 Grouped:

- **Pedestrian and People (Grouped as Pedestrian):** In the VisDrone-2019 dataset, YOLOv8l achieved the highest mAPs of 51% for Pedestrians and 39.6% for People. Grouping these classes in the VisDrone-2019 Grouped dataset increased the mAP to 56.7% for Pedestrians, showing enhanced detection efficacy.
- **CycleVariants:** Individual mAPs for Bicycle, Tricycle, Awning-Tricycle, and Motor were 20.5%, 36.2%, 18.8%, and 52.7% respectively, with YOLOv8l. Grouping these into 'CycleVariants' raised the mAP to 51.4%, indicating improved detection performance.
- **Automobiles:** The highest mAPs for Car, Van, Truck, and Bus were 82.5%, 48.4%, 44%, and 66.1% respectively. Grouping these as 'Automobiles' in the VisDrone-2019 Grouped dataset increased the mAP to 86.2%, demonstrating an overall enhancement in detection metrics.

2. Stanford Drone vs. Stanford Drone Grouped:

- **Pedestrian and Skaters (Grouped as Pedestrian):** The highest mAPs in the Stanford Drone dataset were 76.6% for Pedestrian and 29.4% for Skater, both achieved by YOLOv8l. Grouping improved the mAP for Pedestrians to 70.8% in the grouped dataset.
- **Bicyclists (Separate Category):** The highest mAP for Bikers was 70.6% (YOLOv8l). In the grouped dataset, it increased slightly to 71.8% (YOLOv8m).
- **Cars, Carts, and Buses (Grouped as Car):** Car, Cart, and Bus had highest mAPs of 87.4%, 29%, and 87.2% respectively. Grouping them increased the mAP for 'Car' to 91.7%, achieved by YOLOv5x, indicating superior detection capabilities.

The analysis highlights that YOLOv8 models, especially YOLOv8l and YOLOv8m, show superior mAP performance when objects are grouped, making them particularly effective for real-time detection in complex and dynamic environments. This suggests that object grouping can significantly enhance the performance of detection models, leading to more accurate and comprehensive detection outcomes, beneficial for applications requiring high precision and adaptability.

In Chapter 4, the experiments and results section will assess the performance of these models, including measuring the inference time on a Raspberry Pi 4. This practical evaluation will offer insights into the real-world applicability and efficiency of the YOLOv8 models in resource-constrained environments, further validating their effectiveness in real-time object detection tasks.

4

Experiments and Results

4.1 Testing Process

The objective was to assess the YOLOv5 and YOLOv8 models on the Stanford Drone and VisDrone Datasets, focusing on both standard and grouped object forms. The evaluation primarily involved precision, recall, and mAP metrics at various IoU thresholds. This analysis underscored the models' proficiency in detecting and classifying objects from elevated urban vantage points, shedding light on their utility for urban surveillance and monitoring applications. The results provide valuable insights into the effectiveness of these models in aerial surveillance scenarios.

4.2 Hardware and Software Setup

The Raspberry Pi, a low-cost single-board computer, has emerged as a versatile tool in various sectors, from industrial automation to personal projects. The platform's General Purpose Input/Output (GPIO) ports facilitate easy integration with sensors, actuators, and other hardware components, making it ideal for a diverse range of applications. Raspberry Pi boards support multiple operating systems and are available in several models, each with unique specifications. Despite their differences, all models share core design principles: affordability, portability, and versatility.

4.2.1 Raspberry Pi 4 Specifications

This section outlines the hardware and software setup for YOLO object detection algorithms on the Raspberry Pi 4. Selected for its balance of performance, affordability, and accessibility, this device excels in edge computing applications like real-time object detection.

Hardware Overview

The Raspberry Pi 4 Model B used in this study offers the following specifications [64]. Below is a table comparing all three models of Raspberry Pi:

	Raspberry Pi 3	**Raspberry Pi 4**	**Raspberry Pi 5**
CPU	Broadcom BCM2837, Cortex-A53 64Bit SoC	Broadcom BCM2711, Cortex-A72 64Bit SoC	Broadcom BCM2712, Cortex-A76 64Bit SoC
CPU Max Frequency	1.4GHz	1.8GHz	2.4GHz
GPU	Videocore IV	Videocore VI	VideoCore VII
GPU Max Frequency	400Mhz	500Mhz	800Mhz
Memory	1GB LPDDR2 SDRAM	1GB, 2GB, 4GB, 8GB LPDDR4-3200 SDRAM	4GB, 8GB LPDDR4X-4267 SDRAM
PCIe	N/A	N/A	1xPCIe 2.0 Interface
Max Power Draw	2.5A@5V	3A@5V	5A@5V (PD enabled)

Table 4.1: Raspberry Pi Series Comparison

Software Setup

The Raspberry Pi 4 was configured to optimize the deployment of YOLO object detection models:

Operating System: Raspberry Pi OS (64-bit) Bullseye was selected for its enhanced stability and compatibility.

Python Version: Python 3.9.2 was installed to support essential machine learning libraries.

Deep Learning Libraries: PyTorch 1.11.0 and torchvision 0.12.0 were chosen for their efficiency on resource-limited devices like the Raspberry Pi.

Environment Upgrade: Due to performance limitations on the Raspberry Pi 3 with YOLOv5m, an upgrade to Raspberry Pi 4 was necessary, including reinstallation of the OS for improved remote access functionality.

Network Configuration: SSH and VNC setups were implemented for reliable remote access, facilitating easy management and deployment of models.

System Optimization: Adjustments to Python packages and system configurations were made to enhance performance, ensuring optimal operation of the deep learning models.

Performance Considerations

Initial experiments on the Raspberry Pi 4 demonstrated marked enhancements in stability and computational speed over the Raspberry Pi 3 configuration. The YOLOv5 and YOLOv8 models were fine-tuned to accommodate the computational boundaries of the Raspberry Pi 4, ensuring effective performance even in resource-constrained environments.

Benchmark data from the Ultralytics website reveal that the inference time for NCNN on the Raspberry Pi 4 averages 414.73 ms per image (Table 4.2) [65]. Additional results from my experiments will be detailed later, providing deeper insights into the efficacy and operational efficiency of these configurations.

Format	Size on disk (MB)	mAP50-95(B)	Inference time (ms/im)
PyTorch	6.2	0.6381	1068.42
TorchScript	12.4	0.6092	1248.01
ONNX	12.2	0.6092	560.04
OpenVINO	12.3	0.6092	534.93
TF SavedModel	30.6	0.6092	816.50
TF GraphDef	12.3	0.6092	1007.57
TF Lite	12.3	0.6092	950.29
PaddlePaddle	24.4	0.6092	1507.75
NCNN	12.2	0.6092	414.73

Table 4.2: YOLOv8n on RPi4

4.3 Testing on Raspberry Pi 4

4.3.1 Exporting YOLOv5 to OpenVINO

To export YOLOv5 to OpenVINO for testing on the Raspberry Pi 4, use the export.py script provided by Ultralytics:

```
python export.py --weights yolov5s.pt --include openvino
```

This command converts the YOLOv5 model (`yolov5s.pt`) into the OpenVINO format. The exported model can then be used for inference with OpenVINO, which can enhance performance on CPU-based systems like the Raspberry Pi 4 [66].

4.3.2 Exporting YOLOv8 to NCNN

To export YOLOv8 to NCNN for testing on the Raspberry Pi 4, use the export.py script from Ultralytics. Run the following command:

```
python export.py --weights yolov8s.pt --include ncnn
```

This converts the YOLOv8 model (`yolov8s.pt`) into the NCNN format. The exported model can then be used for inference with NCNN, optimizing performance on the Raspberry Pi 4.

Stanford Drone								
	Metric / Model	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
Image size 640	P(%)	35.4	34.6	33.7	35.9	**37.3**	34	32.4
	R(%)	24.9	25.1	**25.3**	14.9	18.6	19	20
	mAP@0.5	**26**	25.4	24.9	22.8	24.8	23.6	22.9
	mAP@0.5-0.95	9.2	9	9.2	8.5	**9.5**	8.8	8.7
	Metric / Model	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
Image size 416	P(%)	35.5	34.9	33.2	**36.7**	33.9	35.7	33.3
	R(%)	16.8	19.2	**23**	11.3	15.7	15.4	18.8
	mAP@0.5	24.1	24	**24.4**	22.7	22.2	23.5	24
	mAP@0.5-0.95	9.3	9.3	**9.4**	7.5	8.8	8.7	8.4
	Metric / Model Format	**Yolov5s openvino**	**Yolov5m openvino**	**Yolov5x openvino**	**Yolov8n ncnn**	**Yolov8s ncnn**	**Yolov8m ncnn**	**Yolov8l ncnn**
Different Formats	P(%)	**35.2**	34.2	33.5	30.3	31.9	28.7	29.6
	R(%)	25	25.1	**25.2**	13.5	18.5	19	19.9
	mAP@0.5	**25.6**	24.3	24.8	19.7	21.5	20.4	21.4
	mAP@0.5-0.95	9.2	9.1	**9.5**	7.5	7.6	7.5	7.5
Stanford Drone Grouped								
	Metric / Model	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
Image size 640	P(%)	53.8	51.7	49	53.1	48.2	52.5	**54.3**
	R(%)	34.9	33.5	**39.8**	22.1	28.3	30.5	33
	mAP@0.5	37.8	36.5	36.6	32.9	32.3	35	**38**
	mAP@0.5-0.95	12.5	12.5	12.3	10.2	10	12.1	**13.5**
	Metric / Model	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
Image size 416	P(%)	50.6	48.9	47.3	50.1	44.5	**54**	52.6
	R(%)	29.5	32.2	**35.1**	14.1	22.1	27.6	28.9
	mAP@0.5	**35.7**	34.1	34.8	28.9	27.9	35.5	35.1
	mAP@0.5-0.95	12.5	11.5	12.2	9.1	7.9	**13.8**	13.3
	Metric / Model Format	**Yolov5s openvino**	**Yolov5m openvino**	**Yolov5x openvino**	**Yolov8n ncnn**	**Yolov8s ncnn**	**Yolov8m ncnn**	**Yolov8l ncnn**
Different Formats	P(%)	**53.3**	50.9	49.6	51.1	43.3	47.8	47.8
	R(%)	33	35.2	**40.1**	22.7	28.5	30.5	33.4
	mAP@0.5	**37.1**	36.2	**37.1**	32	29.1	32.2	34.7
	mAP@0.5-0.95	12.2	12	**12.6**	10.1	8.6	11	11.7

Table 4.3: Comparison of different models in Stanford Drone test Dataset

VisDrone								
	Metric / Model	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
Image Size 640	P(%)	**56.9**	51	50.1	50.8	56.1	56	54.9
	R(%)	12.7	17.2	20.2	13	16.7	19.4	**21.2**
	mAP@0.5	34.9	34.1	35.1	31.8	36.4	37.6	**37.8**
	mAP@0.5-0.95	21.6	20.7	21.9	20	23.5	24.5	**24.8**
	Metric / Model	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
Image Size 416	P(%)	46	47.3	49.7	51	**55.3**	55.1	54.8
	R(%)	7.3	11.6	**14.4**	7.2	10.6	12.3	13.8
	mAP@0.5	26.7	29.5	32	29.1	32.9	33.7	**34.3**
	mAP@0.5-0.95	16	16.8	18.6	17.7	19.9	21	**21.3**
	Metric / Model Format	**Yolov5s openvino**	**Yolov5m openvino**	**Yolov5x openvino**	**Yolov8n ncnn**	**Yolov8s ncnn**	**Yolov8m ncnn**	**Yolov8l ncnn**
Different Formats	P(%)	**58.1**	51.5	50.9	44.5	50.4	50.4	49.6
	R(%)	13.7	18.6	**21.8**	13	16.7	19.3	21.3
	mAP@0.5	36	35.1	**36.3**	27.8	32.4	33	33.5
	mAP@0.5-0.95	22.3	21.5	**22.8**	16.5	19.3	19.7	20.2
VisDrone Grouped								
	Metric	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
Image Size 640	P(%)	73.3	73.1	73.2	**79.9**	78.8	79.1	78.6
	R(%)	27.4	32.3	35.9	24.6	30.1	34	**36.4**
	mAP@0.5	49.8	51.6	53.4	51.6	53.4	55.3	**56.1**
	mAP@0.5-0.95	28.3	29.9	31.7	31.2	32.4	33.8	**34.3**
	Metric	**Yolov5s**	**Yolov5m**	**Yolov5x**	**Yolov8n**	**Yolov8s**	**Yolov8m**	**Yolov8l**
Image Size 416	P(%)	71.2	71.8	72	76.1	**77.4**	76.5	77
	R(%)	19.1	22.7	**26.3**	15	19.6	22.6	24.5
	mAP@0.5	44.9	46.6	48.3	45.2	47.8	48.7	**49.7**
	mAP@0.5-0.95	23.8	25.3	26.7	25.5	27.6	28.3	**28.9**
	Metric / Model Format	**Yolov5s openvino**	**Yolov5m openvino**	**Yolov5x openvino**	**Yolov8n ncnn**	**Yolov8s ncnn**	**Yolov8m ncnn**	**Yolov8l ncnn**
Different Formats	P(%)	73.5	73	**73.6**	72.8	72.2	72.2	72.3
	R(%)	28.7	33.3	36.2	24.7	30.3	34	**36.5**
	mAP@0.5	50.5	52.1	**53.7**	46.7	48.5	49.7	50.8
	mAP@0.5-0.95	28.8	30.3	**31.8**	26.6	27.9	28.6	29.3

Table 4.4: Comparison of different models in VisDrone-2019 test Dataset

4.4 Performance Analysis

4.4.1 Accuracy of Object Detection

This section assesses the object detection accuracy of YOLOv5 and YOLOv8 models on the Raspberry Pi 4, using both the Stanford Drone and VisDrone test datasets. The evaluation focuses on key metrics including precision, recall, and mAP across various IoU thresholds, highlighting the models' effectiveness in urban object detection scenarios. Results are presented in Tables 4.3 and 4.4, which showcase performance variations across different model versions and image resolutions for the Stanford Drone and VisDrone datasets, respectively. Significant findings are emphasized with bold and underscored values in these tables. The comprehensive analysis of model performance under different configurations and datasets is further elaborated in Chapters 4.4.2 and 4.4.3, providing insights into the models' capabilities and limitations..

4.4.2 Comparative Analysis of YOLO Model Performance on Stanford Drone Dataset

This subsection evaluates YOLO models on the Stanford Drone test Dataset across different configurations. YOLOv8 models, specifically YOLOv8s and YOLOv8l, demonstrate significant strengths in precision and mAP across both standard and grouped configurations. YOLOv8s stands out at higher resolutions (640 * 640 pixels), while YOLOv5x exhibits a high recall, particularly at a resolution of 416 by 416 pixels. The OpenVINO optimization framework proves to be highly effective, consistently enhancing model performance in terms of precision and mAP. This effectiveness can be attributed to several factors:

Model Optimization: OpenVINO applies techniques like quantization, reducing weight and activation precision from 32-bit floating point to 8-bit integers, enhancing speed without significant accuracy loss [67] [68].

Hardware Acceleration: OpenVINO leverages Intel hardware, utilizing advanced vector extensions (AVX) instructions for faster inference times [69] [70] [68].

Graph Optimization: The framework optimizes the model's computational graph, fusing operations and eliminating redundant computations, leading to improved precision and overall performance [71] [72].

In grouped configurations, YOLOv8l excels, indicating its suitability for complex detection tasks involving multiple object classes. This superior performance can be explained by:

Model Capacity: YOLOv8l's larger network architecture allows it to learn more complex features and relationships between different object classes [16].

Multi-scale Feature Extraction: Its advanced feature pyramid network extracts features at multiple scales, beneficial for identifying objects of various sizes and complexities simultaneously [73].

Improved Loss Function: YOLOv8 refines the loss function, including sophisticated objectness prediction, helping handle scenarios with multiple, potentially overlapping objects [74] [16].

Advanced Data Augmentation: Techniques like mosaic augmentation improve performance on diverse and complex scenes, often encountered in grouped object scenarios [16] [75].

These findings suggest that while YOLOv8 models offer advanced detection capabilities, leveraging optimization frameworks like OpenVINO can further enhance performance, making these models highly adaptable and effective for various object detection applications.

4.4.3 Comparative Analysis of YOLO Model Performance on VisDrone Dataset

This section evaluates YOLO model performance on the VisDrone test dataset under various conditions. YOLOv8 models, particularly YOLOv8l, demonstrate strong precision and mAP across 640 and 416 pixel resolutions. YOLOv8l's consistent high mAP scores at both 0.5 and 0.5-0.95 IoU thresholds indicate reliable detection capabilities across scenarios. Conversely, YOLOv5x excels in recall, especially at 416 pixels, suggesting effective object detection within images. OpenVINO and NCNN optimizations significantly enhance precision and mAP metrics for YOLO models. YOLOv5x notably benefits from OpenVINO optimization, improving precision for accurate detection.

OpenVINO's effectiveness stems from its model optimization techniques, hardware acceleration capabilities, and graph optimization, as previously discussed in the Stanford Drone Dataset analysis. In grouped analyses, YOLOv8l demonstrates exceptional versatility in complex object detection scenarios.

These results underscore the efficacy of YOLOv8 models and optimization techniques in achieving advanced object detection on the VisDrone dataset, particularly in complex scenarios involving multiple object classes and varied object sizes typical in aerial imagery.

4.4.4 Effects of Class Variations on Performance

This section evaluates how class variations affect the performance on Stanford test datasets (Table 4.5) and VisDrone test datasets (Table 4.6).

Stanford Drone Dataset Analysis

Resolution Impact:

- Higher resolution (640) generally results in better AP scores across most object classes and configurations.
- Some models perform relatively better at lower resolution (416), highlighting a trade-off between model complexity and input size.

Model Performance:

- YOLOv5 models, particularly YOLOv5s and YOLOv5x openvino, consistently perform well across different classes and configurations.
- YOLOv8 models show competitive performance but tend to lag slightly behind YOLOv5 in certain classes, especially at higher resolutions.

Class-Specific Performance:

- For "Pedestrian" and "Biker" classes, YOLOv5 models show a notable performance advantage over YOLOv8.
- In the "Car" and "Bus" classes, YOLOv5 models often achieve higher AP scores, particularly at higher resolutions.

Grouped Class Performance:

- Grouping classes impacts performance, but YOLOv5 models generally maintain a slight edge over YOLOv8.
- YOLOv8 models handle grouped scenarios well, indicating potential for multi-class object detection.

For the Stanford Drone dataset, YOLOv5 models, particularly YOLOv5s and YOLOv5x openvino, consistently outperform YOLOv8 models across most classes and configurations. YOLOv5 models demonstrate a notable performance advantage for "Pedestrian" and "Biker" classes, and they often achieve higher AP scores in the "Car" and "Bus" classes, especially at higher resolutions. Although grouping classes affects performance, YOLOv5 models generally maintain a slight edge over YOLOv8 models. However, YOLOv8 models handle grouped scenarios well, indicating their potential for effective multi-class object detection.

The apparent difference in performance between YOLOv8 and YOLOv5 in the general analysis versus the class-wise analysis for the Stanford dataset is interesting and requires additional examination. Although straightforward answers are not accessible without further research, various possible causes to this observation can be considered:

1. Overall vs. Specific Performance: In the general performance analysis, YOLOv8 shows better overall metrics (mAP, precision) across all classes combined. However, in the class-wise analysis, YOLOv5 appears to demonstrate superior performance for specific individual classes [76] [77] [78] [79].

2. Class Imbalance: The Stanford dataset have imbalanced class distributions [80]. YOLOv8 could be handling this imbalance better overall, but YOLOv5 might excel at detecting certain less represented classes [78] [81].

3. Detection Strategies: YOLOv5 and YOLOv8 might employ different strategies for object detection. YOLOv5's approach might be more effective for certain object types or sizes common in the Stanford dataset [78] [76].

To justify these results: The general performance metrics (overall mAP, precision) favor YOLOv8, indicating its strength in handling the dataset as a whole. The class-wise analysis reveals that YOLOv5 retains advantages for specific object categories, suggesting it might be better optimized for certain types of objects or scene characteristics present in the Stanford dataset.

This discrepancy highlights the importance of considering both overall performance and class-specific metrics when evaluating object detection models. It also underscores the complexity of model selection, where the best overall performer may not always be the optimal choice for every specific task or object class. In practical applications, the choice between YOLOv5 and YOLOv8 for the Stanford dataset would depend on the specific requirements of the task at hand whether overall performance or excellence in detecting particular object classes is more critical.

Model	Stanford Dataset				Stanford Grouped Dataset		
	Pedestrian	Biker	Car	Bus	Pedestrian	Biker	Car
Yolov5s (640)	**30.8**	37.2	**62.3**	24.9	**30.7**	36.2	46.4
Yolov5m (640)	30.2	36.3	61.3	23.8	29.4	37	43
Yolov5x (640)	29	37.2	61.8	20	30.2	**37.8**	41.9
Yolov8n (640)	28.4	33.9	59.6	13.5	28.8	33.8	36.2
Yolov8s (640)	28.6	35.5	55.6	**27.8**	30.1	34.3	32.5
Yolov8m (640)	28.6	35.8	59.5	17.7	29.9	34.6	40.4
Yolov8l (640)	29.2	**37.9**	58.7	9.5	29.2	36.2	**48.5**
Yolov5s (416)	22	32.5	60.8	28.3	24.7	31.4	**51**
Yolov5m (416)	22.7	33.4	63.8	26.2	24	32.3	45.8
Yolov5x (416)	23.9	33.5	**65.4**	22.6	23.7	**33.3**	47.4
Yolov8n (416)	23.7	27.7	52.3	**32.7**	22.1	29.9	34.7
Yolov8s (416)	22.7	31.5	54.6	18.3	23.5	31.6	28.6
Yolov8m (416)	**25.5**	31.4	59.4	24.1	**25**	31.8	49.8
Yolov8l (416)	24.9	**33.6**	55.6	29.1	24.6	32.3	48.4
Yolov5s openvino	**30.8**	36.4	61.8	**23.3**	**30.5**	36.9	**43.9**
Yolov5m openvino	29	35.2	60.6	22.7	29.1	36.4	43
Yolov5x openvino	28.8	**36.8**	**63.7**	18.1	30	**37.7**	43.6
Yolov8n ncnn	28.4	31.9	56.9	19.7	27.9	32	36.1
Yolov8s ncnn	27.3	32.1	48.7	20.2	28.5	31.6	27.2
Yolov8m ncnn	27.3	32.5	51.3	10.8	28.8	31.9	35.8
Yolov8l ncnn	29.2	34.7	50.5	12.9	28.2	33.4	42.6

Table 4.5: AP values for different models and categories from the Stanford and Stanford Grouped test datasets

VisDrone Dataset Analysis

Resolution Impact:

- Higher resolution (640) generally yields better AP scores across most classes. Some models also perform well at 416 resolution, suggesting a balance between input size and model complexity.

Model Performance:

- YOLOv8 models, particularly YOLOv8l, often outperform YOLOv5 models in the VisDrone dataset, showing superior performance across several classes.

Class-Specific Performance:

- For "Pedestrian" and "People" classes, YOLOv8l (640) achieves the highest AP scores, indicating its effectiveness in detecting smaller objects.
- In the "Car" and "Van" classes, YOLOv8m and YOLOv8l models show excellent performance, with YOLOv8l (640) achieving the highest AP scores.

Grouped Class Performance:

- In grouped classes, YOLOv8 models generally maintain an advantage, particularly in "Pedestrian" and "CycleVariants" categories.
- YOLOv5x openvino achieves high AP scores in "AutoMobiles," demonstrating robustness in detecting larger vehicle classes.

In the VisDrone dataset, YOLOv8 models, particularly YOLOv8l, frequently outperform YOLOv5 models, demonstrating superior performance across several classes. YOLOv8l (640) achieves the highest AP scores for "Pedestrian" and "People" classes, indicating its effectiveness in detecting smaller objects. Furthermore, YOLOv8m and YOLOv8l models exhibit excellent performance in the "Car" and "Van" classes, with YOLOv8l (640) achieving the highest AP scores. In grouped classes, YOLOv8 models generally maintain an advantage, particularly in "Pedestrian" and "CycleVariants" categories, while YOLOv5x openvino achieves high AP scores in "AutoMobiles," demonstrating its robustness in detecting larger vehicle classes. Overall, the findings underscore the importance of selecting appropriate model configurations based on specific class characteristics and application re- quirements. YOLOv5 models show strong performance in the Stanford Drone dataset, while YOLOv8 models excel in the VisDrone dataset, suggesting that model selection should be tailored to the particular dataset and object classes being analyzed.

Model	**VisDrone Dataset**						**VisDrone Grouped Dataset**		
	Pedestrian	**People**	**Bicycle**	**Car**	**Van**	**Truck**	**Pedestrian**	**CycleVariants**	**AutoMobiles**
Yolov5s (640)	47.3	42.6	**40.1**	67.2	38.6	43.8	46.9	30.5	72
Yolov5m (640)	47.9	42.3	27.9	72.2	39.1	43.4	48.4	31.9	74.7
Yolov5x (640)	49.4	39.9	28.5	74.6	40.5	48.3	50.9	33.1	**76.6**
Yolov8n (640)	48.3	39	16	68.5	38.5	44.5	48.6	35.3	70.8
Yolov8s (640)	50.2	42.8	32	72.8	43	50.1	50.6	35.5	74
Yolov8m (640)	51.7	41.9	34.2	**74.7**	44.9	52.6	52	37.8	76
Yolov8l (640)	**52.5**	**42.9**	30.7	75.9	**46.6**	**53.7**	**53.1**	**38.4**	54.2
Yolov5s (416)	45.1	**41.8**	**39.8**	58.8	31.1	36.8	44.1	26.6	63.9
Yolov5m (416)	44.6	37.1	22.5	64.4	31.2	36.1	45	27.9	66.9
Yolov5x (416)	44.9	36.5	31.1	**67.3**	34.3	41.2	46.1	28.8	**69.8**
Yolov8n (416)	45.4	41.6	22.3	58.1	32.9	35.9	46.8	28	60.7
Yolov8s (416)	47	**41.8**	28.2	63.1	37	46.1	48.1	30.6	64.8
Yolov8m (416)	48.2	39.8	31	65.6	**39.4**	45.9	48.6	30.1	67.3
Yolov8l (416)	**48.6**	41.2	31.6	67.1	39.3	**46.4**	**48.9**	**31.7**	68.5
Yolov5s openvino	47.8	**44.7**	**41.1**	69.7	40	45	47.7	30.5	73.2
Yolov5m openvino	48.9	43.8	28.1	74.8	40.5	44.7	48.7	32.5	75.2
Yolov5x openvino	**50.1**	42.3	30.6	**76.2**	**42.1**	**49**	**50.9**	33.8	**76.3**
Yolov8n ncnn	43.4	38	22.5	60	32.9	35.9	44.7	33.8	61.7
Yolov8s ncnn	46.3	42.3	29.9	63.7	37.1	39.3	47.3	34	64.3
Yolov8m ncnn	47.5	41.1	34.2	63.7	37.1	40.3	48.7	35.8	64.7
Yolov8l ncnn	49.2	41.6	29.2	65.5	39.8	41.9	50	**36.5**	65.8

Table 4.6: AP values for different models and categories from the VisDrone and VisDrone Grouped test datasets

4.4.5 Inference Time Measurement

4.4.6 Inference Time Analysis Using Stanford Drone and Grouped Dataset

This analysis examines the inference times of YOLO models on a Raspberry Pi 4 using the Stanford Drone and Stanford Drone Grouped datasets. Models tested include YOLOv5 and YOLOv8 variants with default, NNCN, and OpenVINO weights at image sizes 640 and 416 pixels (Figure 4.1).

Key Findings

YOLOv5s:

- **Stanford Drone (640)**: Default - 1427.5 ms, OpenVINO - 761.3 ms
- **Stanford Drone Grouped (640)**: Default - 1430.6 ms, OpenVINO - 775.0 ms

YOLOv8n:

- **Stanford Drone (640)**: Default - 882.9 ms, NNCN - 401.6 ms; (416): Default - 415.7 ms
- **Stanford Drone Grouped (640)**: Default - 861.7 ms, NNCN - 422.0 ms; (416): Default - 414.7 ms

Conclusions

The analysis shows numerous major findings. First, smaller models like YOLOv8n have faster inference times, making them ideal for real-time applications that require fast processing. Second, OpenVINO and NNCN weights improve performance, making them a good optimization technique for resource-constrained devices. Third, while lowering image sizes to 416 pixels reduces inference durations, it may affect accuracy, therefore a careful balance between speed and precision is needed. Finally, grouped datasets improve object detection but increase inference durations due to the complexity of processing many object categories simultaneously. These findings emphasize the necessity of carefully evaluating model size, optimization strategies, input image dimensions, and dataset complexity when deploying object identification systems in real-world applications, especially on edge devices like the Raspberry Pi 4.

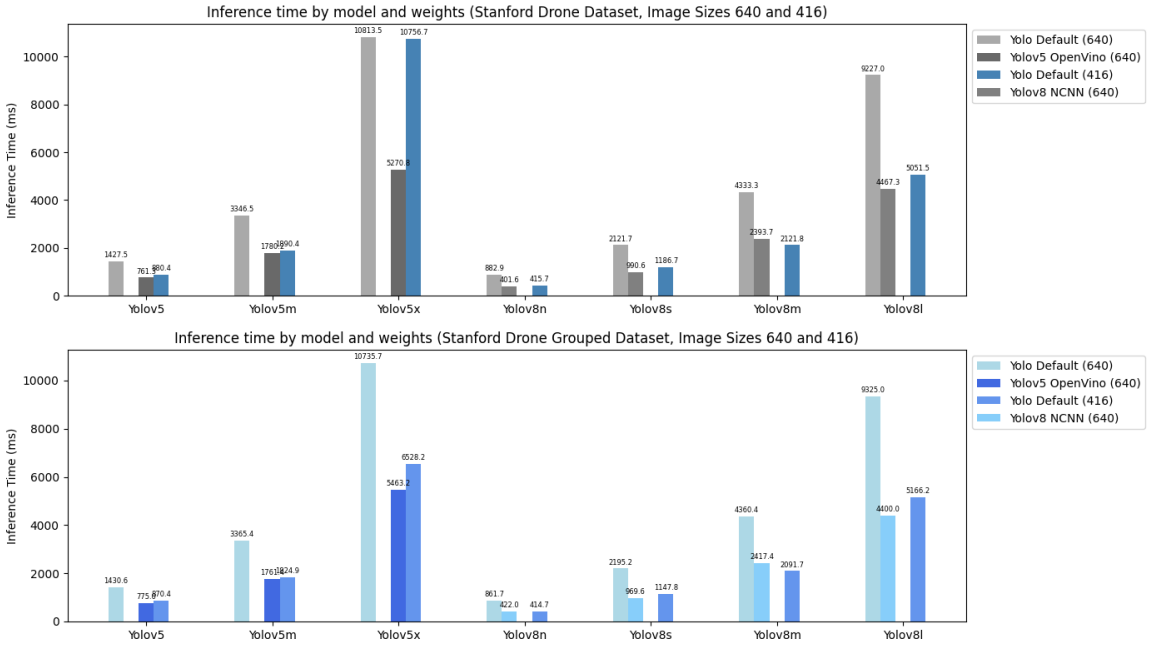


Figure 4.1: Inference time by model and weights (Stanford Drone and Stanford Drone Grouped Dataset with Grouped Dataset)

4.4.7 Inference Time Analysis Using Visdrone-2019 DET and Grouped Datasets

Models tested include YOLOv5 and YOLOv8 variants with default, NNCN, and OpenVINO weights at image sizes 640 and 416 pixels (Figure 4.2).

Key Findings

YOLOv5s:

- **Visdrone (640)**: Default - 1280.5 ms, OpenVINO - 761.3 ms; (416): Default - 766.8 ms
- **Visdrone Grouped (640)**: Default - 1316.1 ms, OpenVINO - 772.5 ms; (416): Default - 733.9 ms

YOLOv8n:

- **Visdrone (640)**: Default - 787.4 ms, NNCN - 443.9 ms; (416): Default - 393.7 ms
- **Visdrone Grouped (640)**: Default - 811.6 ms, NNCN - 432.0 ms; (416): Default - 389.6 ms

Conclusions

The Raspberry Pi 4 investigation shows that smaller YOLO models, such as YOLOv8n, are better for real-time applications on resource-constrained devices due to their lower inference times. Performance improvement methods like OpenVINO for YOLOv5 and NNCN weights for YOLOv8 boost efficiency. Reducing image sizes to 416 pixels speeds inference but may reduce accuracy, requiring a balance between speed and precision. Scene complexity increases inference times for grouped datasets. For NNCN and OpenVINO models, decreasing picture size increases post-process inference time unexpectedly. These findings show that model architecture, optimization techniques, input dimensions, and dataset complexity are crucial when implementing object detection systems on edge devices to optimise performance in resource-constrained environments while maintaining detection accuracy.

These findings support the use of optimized smaller YOLO models for real-time object detection on resource-constrained devices like the Raspberry Pi 4.

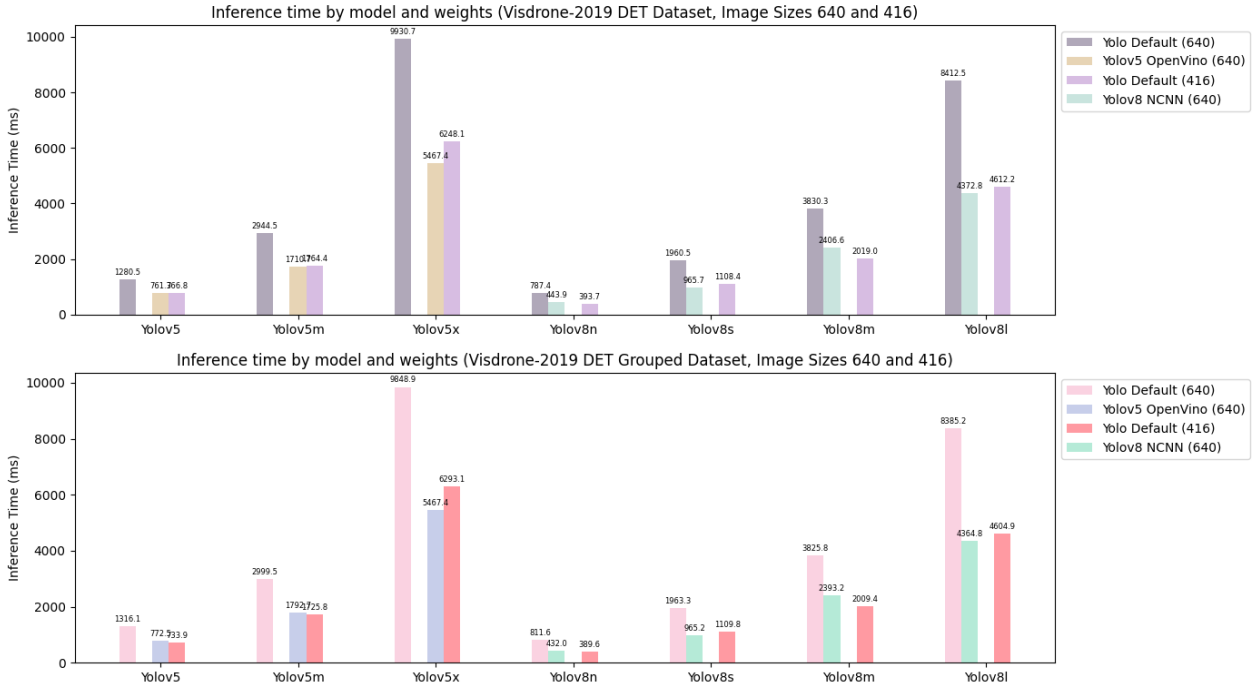


Figure 4.2: Inference time by model and weights (Visdrone-2019 and Visdrone-2019 with Grouped Dataset)

4.4.8 Power Consumption Measurement

Measuring power consumption is critical for evaluating the efficiency of edge devices, such as the Raspberry Pi 4, particularly when running sophisticated YOLO models in drone-based applications. This analysis examines the energy demands associated with the operation of YOLOv5 and YOLOv8 under various configurations, highlighting the paramount importance of balancing performance and power usage for drone operations in remote or power-constrained environments.

In the context of drone-based object detection, energy efficiency becomes a crucial factor that directly impacts mission duration, operational range, and overall system reliability. Drones typically have limited battery capacity, and every milliwatt-hour of energy saved can translate to extended flight time or increased payload capacity. For instance, in applications such as search and rescue operations, environmental monitoring, or urban surveillance using drones, the ability to operate for longer periods without recharging is invaluable [82].

Moreover, the power consumption of onboard computing systems, including object detection models, significantly affects the drone's endurance. Optimizing the energy efficiency of these models can lead to substantial improvements in the drone's operational capabilities. This is particularly relevant in scenarios where drones need to process real-time video feeds for extended periods [83].

By analyzing the power consumption of different YOLO model configurations on the Raspberry Pi 4, we aim to provide insights that can directly inform the design and deployment of energy-efficient drone-based object detection systems. This analysis is crucial for developing sustainable and long-endurance drone operations, especially in remote areas where power sources for recharging may be limited or unavailable.

According to the data provided, the Raspberry Pi 4 consumes 0.340 A at idle. The average power usage of the Raspberry Pi 4 under different configurations can be calculated by averaging the additional current drawn across all tested configurations. This will enable the estimation of the total power consumption when the Raspberry Pi is active.

To determine the average current draw under different configurations, the current values listed above were measured with an oscilloscope during inference on a Raspberry Pi.

The given current values in amperes are:

- Group 1: 0.84, 0.93, 1.1, 0.9, 1.1
- Group 2: 0.83, 0.95, 0.97, 0.98
- Group 3: 0.87, 0.97, 1.1, 0.85, 0.9

First, the average current for each group was calculated:

$$\text{Average of Group 1} = \frac{0.84 + 0.93 + 1.1 + 0.9 + 1.1}{5} = 0.974\,\text{A}$$

$$\text{Average of Group 2} = \frac{0.83 + 0.95 + 0.97 + 0.98}{4} = 0.9325\,\text{A}$$

$$\text{Average of Group 3} = \frac{0.87 + 0.97 + 1.1 + 0.85 + 0.9}{5} = 0.938\,\text{A}$$

Next, the overall average current was computed by taking the mean of these group averages. When examining the average current values, it was observed that there were not significant differences between them, so an overall average current was calculated:

$$\text{Overall average current} = \frac{0.974 + 0.9325 + 0.938}{3} = 0.9482\,\text{A}$$

The energy consumption of a device running specific tasks, such as object detection models on a Raspberry Pi, can be calculated using the formula:

$$\text{Energy (Wh)} = \text{Current (A)} \times \text{Voltage (V)} \times \text{Time (hours)}$$

Here's a breakdown of each component:

1. **Current (A)**: This is the amount of electric current, measured in amperes (A), that the device consumes while performing the task. It is crucial to subtract the idle current from the total current drawn during the task to isolate the current solely attributable to the task.

2. **Voltage (V)**: This is the potential difference across the device, measured in volts (V). For the Raspberry Pi, this is typically 5 volts when powered by a standard USB power supply.

3. **Time (hours)**: This is the duration for which the task runs, measured in hours. Since inference times are often provided in milliseconds for quick tasks like object detection, they need to be converted into hours by dividing by 3600000 (the number of milliseconds in an hour).

The formula used in our scenario calculates the energy consumed during the inference task in milli-watt hours (mWh), accounting for the fact that inference tasks are usually short:

$$\text{Energy (mWh)} = (\text{Current (A)} - \text{Idle Current (A)}) \times \text{Voltage (V)} \times \frac{\text{Time (ms)}}{3600000} \times 1000$$

Here:

- **Idle Current (A)** is the current the Raspberry Pi draws when it is turned on but not performing any computational tasks.
- **Time (ms)** is the inference time in milliseconds, and the result is multiplied by 1000 to convert watt-hours (Wh) to milli-watt hours (mWh) for more precise measurement at a smaller scale.

This formula gives a precise measurement of the energy specifically used for the task, excluding the base energy consumption of the device when idle.

Detailed Energy Consumption Analysis

Energy consumption was meticulously calculated based on the inference times and the current draw for various configurations of the YOLO models applied to both the VisDrone and Stanford Drone datasets, including grouped object scenarios. These calculations adjust for the idle current draw, multiplied by the voltage (5V) and normalized for the duration of inference, providing a measure in milli-watt hours (mWh).

- **VisDrone-2019 DET Dataset:** For both image sizes 640 and 416, configurations including default, NCNN, and OpenVINO were analyzed. The results show that optimizations such as NCNN and OpenVINO effectively reduce energy consumption while maintaining high detection performance.
- **VisDrone-2019 DET Grouped Object Dataset:** Grouped configurations exhibited a decrease in energy usage, suggesting that model optimizations not only enhance detection accuracy but also improve power efficiency.

- **Stanford Drone Dataset:** Analyzing this dataset revealed significant insights into how each model configuration consumes energy, with a notable reduction in energy usage demonstrated by optimized models over standard configurations.

- **Stanford Drone Grouped Dataset:** Similar to the VisDrone Grouped configurations, the Stanford grouped data also showed improved energy efficiency. This underscores the benefits of object grouping and advanced model optimizations in reducing power consumption.

These results are illustrated in detailed bar charts that compare energy consumption across different models and configurations, underscoring the impact of various optimizations on power efficiency.
Figures detailing these results (refer to Figures 4.3 for the Stanford dataset and for the VisDrone dataset Figures 4.4 in the appendix) provide a visual summary of the power efficiency gains achievable through strategic model optimizations.

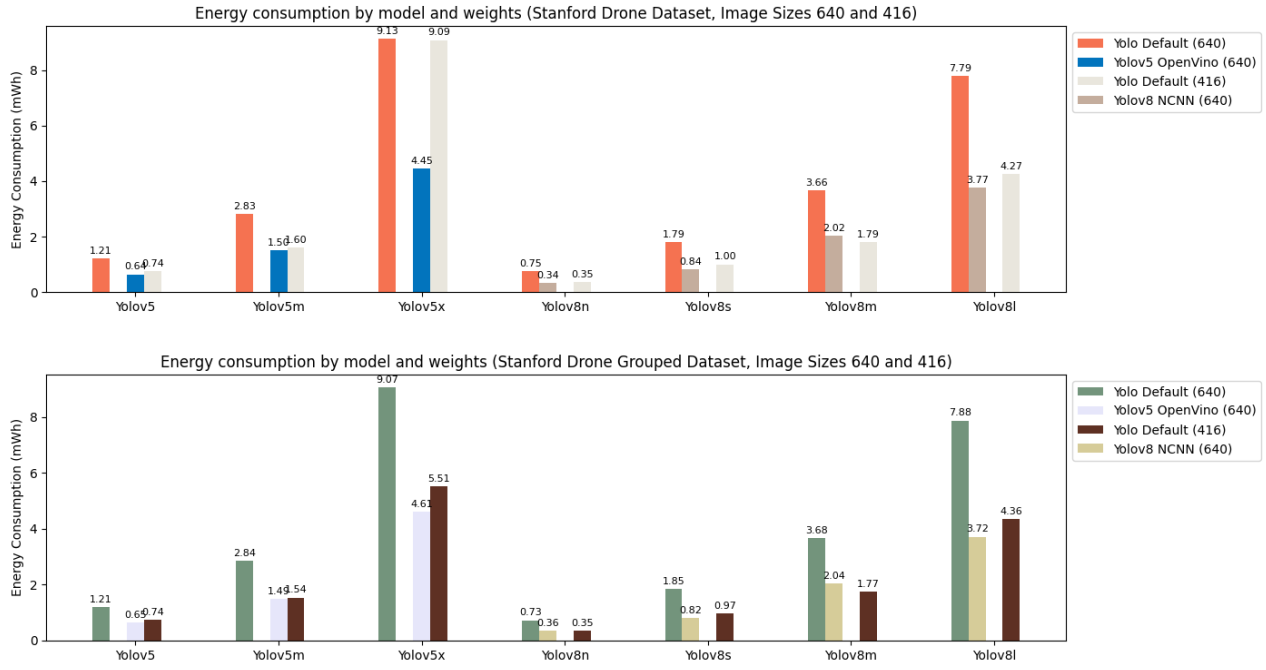


Figure 4.3: Energy consumption of Stanford Drone Datasets for each YOLO model configuration based on inference times and current consumption

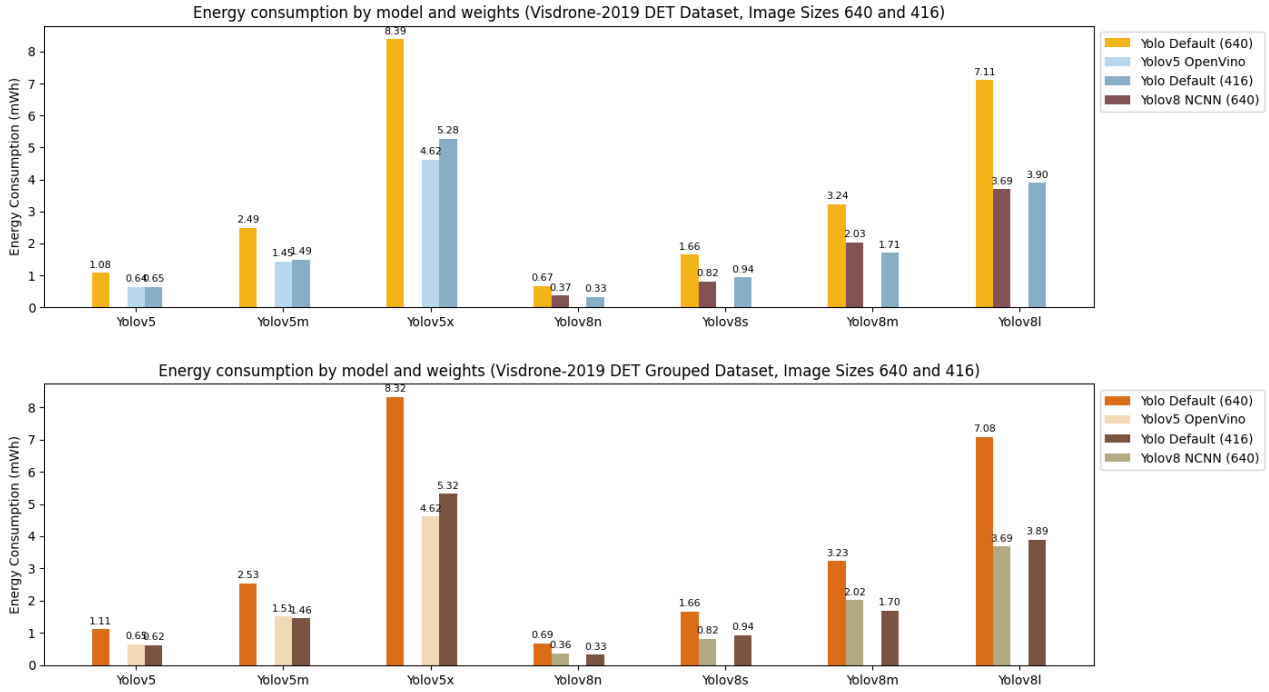


Figure 4.4: Energy consumption of VisDrone Datasets for each YOLO model configuration based on inference times and current consumption

4.5 Summary of Key Findings

This research has provided comprehensive insights into the performance of YOLOv5 and YOLOv8 algorithms for object detection in drone networks, particularly when implemented on edge computing devices like the Raspberry Pi 4. Our extensive analysis across different configurations, optimization techniques, datasets, and hardware platforms has yielded several key findings:

- **YOLO Version:** Overall, YOLOv8 demonstrated superior performance, particularly in terms of accuracy and adaptability to complex scenarios. YOLOv8l consistently achieved higher mAP scores, especially on the VisDrone dataset, indicating its effectiveness in detecting small objects and handling diverse urban environments.

- **Optimization Technique:** The NCNN optimization for YOLOv8 and OpenVINO for YOLOv5 proved highly effective in enhancing both inference speed and energy efficiency. These optimizations were crucial for deploying these models on resource-constrained devices, with NCNN showing particularly impressive results for YOLOv8 models.

- **Dataset Performance:** The VisDrone dataset, especially in its grouped configuration, yielded the best results in terms of detection accuracy and model adaptability. The grouped approach in both Stanford Drone and VisDrone datasets significantly improved detection performance, suggesting its utility in real-world applications.
- **Raspberry Pi Platform:** Our analysis focused on the Raspberry Pi 4, which proved to be a capable platform for edge computing applications in drone-based object detection. The Raspberry Pi 4 demonstrated a good balance between computational power and energy efficiency, making it suitable for real-world deployment. For instance, the inference time for YOLOv8n using NCNN was 414.73 ms, which is acceptable for many real-time applications considering the device's compact size and low power consumption.
- **Optimal Configuration:** Considering the balance between accuracy, speed, and energy efficiency, the YOLOv8l model, optimized with NCNN, running on a Raspberry Pi 4, and trained on the grouped VisDrone dataset emerged as the most promising configuration for drone-based object detection in edge computing scenarios.

These findings underscore the potential of optimized YOLO models in drone-based edge computing applications. The combination of advanced model architectures like YOLOv8, efficient optimization techniques such as NCNN, and thoughtfully structured datasets offers a powerful solution for real-time object detection in resource-constrained environments. As edge computing and drone technologies continue to evolve, these insights provide a solid foundation for future developments in this field.

5

Conclusions and Future Work

5.1 Conclusions

This thesis extensively analyzed the performance of YOLOv5 and YOLOv8 algorithms on the Stanford Drone and VisDrone datasets using a Raspberry Pi 4 platform, focusing on scenarios involving both individual and grouped objects. The experimental results highlighted the effectiveness of these models in handling real-time object detection tasks with varying degrees of computational constraints and power efficiencies.

1. **Performance Insights:** The YOLO models demonstrated robust detection capabilities across both datasets, with significant improvements in detection accuracy when optimized with OpenVINO and NCNN frameworks. The research underscored the trade-offs between speed, accuracy, and power consumption, essential for edge computing applications in drone networks.

2. **Technological Contributions:** The adaptation of YOLO models to edge devices like Raspberry Pi 4 involved significant technical enhancements that have broad implications for the deployment of intelligent systems in resource-limited environments. The findings contribute to the growing body of knowledge in aerial surveillance, providing actionable insights for improving object detection frameworks in edge computing scenarios.

5.2 Future Work

Looking forward, the rapidly evolving field of object detection presents several avenues for further research:

1. **Algorithmic Enhancement:** Future studies could explore the integration of more recent advancements in neural network architectures and learning paradigms. The development of YOLOv10 and subsequent models could be investigated for their potential to enhance detection performance further, particularly in environments with even more stringent power and processing limitations.

2. **Broader Application Scenarios:** Expanding the application domains to include underwater and nighttime environments could diversify the utility of the detection systems. These environments pose unique challenges such as varying light conditions and obscured views, which could drive innovations in sensor technology and algorithmic adaptability.

3. **Cross-Platform Optimization:** Further research could also look into the cross-platform optimization of these models, assessing their performance not only on different hardware configurations but also across various operating systems. This could enhance the models' adaptability and scalability, crucial for their deployment in diverse operational settings.

4. **Energy Efficiency Improvements:** Given the constraints observed in power consumption, future research should also focus on developing more energy-efficient algorithms that do not sacrifice performance. This could involve exploring new methods of data processing and transmission that reduce the energy footprint of drone-based monitoring systems.

5. **Real-Time Data Processing:** Enhancing the capability for real-time data processing and decision-making at the edge would be critical for applications requiring immediate responses, such as active surveillance and emergency response scenarios.

In summary, this thesis enhances the use of YOLO algorithms in edge computing, contributing to the field of drone-based monitoring. Future work should build on these results, focusing on how these solutions can be scaled and applied in real-world situations.

References

[1] Ju Ren et al. "Distributed and Efficient Object Detection in Edge Computing: Challenges and Solutions". In: *IEEE Network* 32 (2018), pp. 137–143. URL: https://api.semanticscholar.org/CorpusID:54213309.

[2] Joel Dick et al. "High speed object tracking using edge computing: poster abstract". In: *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (2017). URL: https://api.semanticscholar.org/CorpusID:22237310.

[3] Razvan-Alexandru Bratulescu et al. "Object Detection in Autonomous Vehicles". In: *2022 25th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. 2022, pp. 375–380. DOI: 10.1109/WPMC55625.2022.10014804.

[4] Rohit Jadhav et al. "Drone Based Object Detection using AI". In: *2022 International Conference on Signal and Information Processing (IConSIP)*. 2022, pp. 1–5. DOI: 10.1109/ICoNSIP49665.2022.10007476.

[5] Panagiotis Aposporis. "Object Detection Methods for Improving UAV Autonomy and Remote Sensing Applications". In: *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 2020, pp. 845–853. DOI: 10.1109/ASONAM49781.2020.9381377.

[6] Abhishek Kumar Saxena, Rajiv Pandey, and Neeraj Kumar Singh. "Latency Analysis and Reduction Methods for Edge Computing". In: *2023 IEEE World Conference on Applied Intelligence and Computing (AIC)* (2023), pp. 480–484. URL: https://api.semanticscholar.org/CorpusID:263627183.

[7] Minh Le et al. "Reliable and efficient mobile edge computing in highly dynamic and volatile environments". In: *2017 Second International Confer-*

ence on Fog and Mobile Edge Computing (FMEC) (2017), pp. 113–120. URL: https://api.semanticscholar.org/CorpusID:10820855.

[8] Reinaldo Padilha França et al. "An Overview of the Edge Computing in the Modern Digital Age". In: 2021. URL: https://api.semanticscholar.org/CorpusID:234257350.

[9] Redowan Mahmud and Adel Nadjaran Toosi. "Con-Pi: A Distributed Container-Based Edge and Fog Computing Framework". In: *IEEE Internet of Things Journal* 9 (2021), pp. 4125–4138. URL: https://api.semanticscholar.org/CorpusID:231572930.

[10] Shivani Mistry and S. Degadwala. "A Comprehensive Review on Object Detectors for Urban Mobility on Smart Traffic Management". In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* (2023). URL: https://api.semanticscholar.org/CorpusID:265548940.

[11] Akash Babu. "Reviewing Innovations: Advances in Transportation, Security, and Accident Detection". In: *INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT* (2024). URL: https://api.semanticscholar.org/CorpusID:268814157.

[12] Zhengwei Bai et al. "Infrastructure-Based Object Detection and Tracking for Cooperative Driving Automation: A Survey". In: *2022 IEEE Intelligent Vehicles Symposium (IV)* (2022), pp. 1366–1373. URL: https://api.semanticscholar.org/CorpusID:246411447.

[13] Anuj Puri. "A Survey of Unmanned Aerial Vehicles (UAV) for Traffic Surveillance". In: 2005. URL: https://api.semanticscholar.org/CorpusID:27195648.

[14] Nikolai Vladimirovich Kim and Mikhail Chervonenkis. "Situation Control of Unmanned Aerial Vehicles for Road Traffic Monitoring". In: *Mathematical Models and Methods in Applied Sciences* 9 (2015), p. 1. URL: https://api.semanticscholar.org/CorpusID:54066803.

[15] Rushikesh Lakhotiya et al. "Image Detection and Real Time Object Detection". In: *International Journal for Research in Applied Science and Engineering Technology* (2023). URL: https://api.semanticscholar.org/CorpusID:258775010.

[16] Rejin Varghese and Sambath. M. "YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness". In: *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)* (2024), pp. 1–6. URL: https://api.semanticscholar.org/CorpusID:269988598.

[17] Tran Quang Khoi, Nguyen Anh Quang, and Ngô Khánh Hiu. "Object detection for drones on Raspberry Pi potentials and challenges". In: *IOP Conference Series: Materials Science and Engineering* 1109 (2021). URL: https://api.semanticscholar.org/CorpusID:233835205.

[18] Tyler Gizinski and Xiang Cao. "Design, Implementation and Performance of an Edge Computing Prototype Using Raspberry Pis". In: *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)* (2022), pp. 0592–0601. URL: https://api.semanticscholar.org/CorpusID:247230791.

[19] Mengxiao Wu and Chi Li. "Edge-based Realtime Image Object Detection for UAV Missions". In: *2021 30th Wireless and Optical Communications Conference (WOCC)* (2021), pp. 293–294. URL: https://api.semanticscholar.org/CorpusID:244137089.

[20] Hari Kishan Kondaveeti et al. "A Review of Image Processing Applications based on Raspberry-Pi". In: *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)* 1 (2022), pp. 22–28. URL: https://api.semanticscholar.org/CorpusID:249475451.

[21] Vladimir Vujovic and Mirjana Maksimovic. "Raspberry Pi as a Wireless Sensor node: Performances and constraints". In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (2014), pp. 1013–1018. URL: https://api.semanticscholar.org/CorpusID:17139187.

[22] Hyun-Kyun Choi et al. "Open source computer-vision based guidance system for UAVs on-board decision making". In: *2016 IEEE Aerospace Conference* (2016), pp. 1–5. URL: https://api.semanticscholar.org/CorpusID:38727127.

[23] Jiashun Suo et al. "E3-UAV: An Edge-Based Energy-Efficient Object Detection System for Unmanned Aerial Vehicles". In: *IEEE Internet of Things*

Journal 11 (2023), pp. 4398–4413. URL: https://api.semanticscholar.org/CorpusID:260646545.

[24] Jianing Deng, Zhiguo Shi, and Cheng Zhuo. "Energy-Efficient Real-Time UAV Object Detection on Embedded Platforms". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39 (2020), pp. 3123–3127. URL: https://api.semanticscholar.org/CorpusID:213569666.

[25] C. Kyrkou et al. "DroNet: Efficient convolutional neural network detector for real-time UAV applications". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2018), pp. 967–972. URL: https://api.semanticscholar.org/CorpusID:5040275.

[26] Igor Bisio et al. "Accuracy-Versus-Energy Evaluation In Drone-Based Video Processing For Object Detection". In: *GLOBECOM 2022 - 2022 IEEE Global Communications Conference* (2022), pp. 5886–5891. URL: https://api.semanticscholar.org/CorpusID:255597404.

[27] Khizer Mehmood et al. "Efficient Online Object Tracking Scheme for Challenging Scenarios". In: *Sensors* 21.24 (2021). ISSN: 1424-8220. DOI: 10.3390/s21248481. URL: https://www.mdpi.com/1424-8220/21/24/8481.

[28] Akshatha K.R. et al. "Manipal-UAV person detection dataset: A step towards benchmarking dataset and algorithms for small object detection". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 195 (2023), pp. 77–89. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2022.11.008. URL: https://www.sciencedirect.com/science/article/pii/S0924271622003008.

[29] Xuexue Li et al. "OGMN: Occlusion-guided multi-task network for object detection in UAV images". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 199 (May 2023), pp. 242–257. ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2023.04.009. URL: http://dx.doi.org/10.1016/j.isprsjprs.2023.04.009.

[30] Abdelmalek Bouguettaya et al. "Vehicle Detection From UAV Imagery With Deep Learning: A Review". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.11 (2022), pp. 6047–6067. DOI: 10.1109/TNNLS.2021.3080276.

[31] Pengfei Zhu et al. "Detection and tracking meet drones challenge". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.11 (2021), pp. 7380–7399.

[32] Alexandre Robicquet et al. "Learning Social Etiquette: Human Trajectory Prediction In Crowded Scenes". In: *European Conference on Computer Vision (ECCV)*. 2016.

[33] Payal Mittal, Raman Singh, and Akashdeep Sharma. "Deep learning-based object detection in low-altitude UAV datasets: A survey". In: *Image and Vision Computing* 104 (2020), p. 104046. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2020.104046. URL: https://www.sciencedirect.com/science/article/pii/S0262885620301785.

[34] Xuan Wang et al. "Small Object Detection Based on Deep Learning for Remote Sensing: A Comprehensive Review". In: *Remote Sensing* 15.13 (2023). ISSN: 2072-4292. DOI: 10.3390/rs15133265. URL: https://www.mdpi.com/2072-4292/15/13/3265.

[35] Alexander Pacha, Jan Hajič, and Jorge Calvo-Zaragoza. "A Baseline for General Music Object Detection with Deep Learning". In: *Applied Sciences* 8.9 (2018). ISSN: 2076-3417. DOI: 10.3390/app8091488. URL: https://www.mdpi.com/2076-3417/8/9/1488.

[36] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS". In: *Machine Learning and Knowledge Extraction* 5.4 (2023), pp. 1680–1716. ISSN: 2504-4990. DOI: 10.3390/make5040083. URL: https://www.mdpi.com/2504-4990/5/4/83.

[37] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[38] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[39] Alan Henry et al. "Lane Detection and Distance Estimation Using Computer Vision Techniques". In: *Machine Learning, Image Processing, Network Security and Data Sciences*. Ed. by Nilay Khare et al. 2022. ISBN: 978-3-031-24367-7.

[40] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV].

[41] Ultralytics. *YOLOv3 Documentation*. https://docs.ultralytics.com/models/yolov3/. Accessed: 2024-04-25. 2022.

[42] Oyku Sahin. "Improving the Performance of YOLO-based Detection Algorithms for Small Object Detection in UAV-Taken Images". Master's thesis. Ankara, Turkey: Bilkent University, Jan. 2023.

[43] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[44] Ultralytics. *YOLOv4 Documentation*. https://docs.ultralytics.com/models/yolov4/. Accessed: 2024-04-25. 2022.

[45] K. Wong, X. Chen, and L. Zhao. "YOLOv6 v3.0: A Full-Scale Reloading". In: *Papers With Code* (2022). URL: https://paperswithcode.com/paper/yolov6-v3-0-a-full-scale-reloading.

[46] Chuyi Li et al. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022. arXiv: 2209.02976 [cs.CV].

[47] Meituan. *YOLOv6 Releases*. https://github.com/meituan/YOLOv6/releases. Accessed: 2024-04-26. 2022.

[48] Deci AI. *How YOLOv6 Differs From YOLOv5 or YOLOX?* https://deci.ai/blog/yolov6-vs-yolov5-vs-yolox/. Accessed: 2024-04-26. 2022.

[49] Viso.ai. *YOLOv7 Complete Guide: Understanding YOLOv7 Inside-Out*. https://viso.ai/deep-learning/yolov7-guide/. Accessed: 2024-04-26. 2023.

[50] Jiayi Xiao et al. "Real-Time Lightweight Detection of Lychee Diseases with Enhanced YOLOv7 and Edge Computing". In: *Agronomy* 13.12 (2023). ISSN: 2073-4395. DOI: 10.3390/agronomy13122866. URL: https://www.mdpi.com/2073-4395/13/12/2866.

[51] Glenn Jocher and Ultralytics. *YOLOv5: YOLOv5 Repository*. https://github.com/ultralytics/yolov5. Accessed: 2023-05-12. 2023.

[52] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics.

[53] Pengfei Zhu et al. "Vision meets drones: A challenge". In: *arXiv preprint arXiv:1804.07437* (2018). URL: https://github.com/VisDrone/VisDrone-Dataset.

[54] Glenn Jocher and the Ultralytics Team. *Customization of YOLO Model*. https://github.com/ultralytics/ultralytics/issues/7304. Accessed: 2024-06-16. 2023.

[55] Haiying Liu et al. "SF-YOLOv5: A Lightweight Small Object Detection Algorithm Based on Improved Feature Fusion Mode". In: *Sensors* 22.15 (2022). ISSN: 1424-8220. DOI: 10.3390/s22155817. URL: https://www.mdpi.com/1424-8220/22/15/5817.

[56] Bailin Liu and Huan Luo. "An Improved Yolov5 for Multi-Rotor UAV Detection". In: *Electronics* 11.15 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11152330. URL: https://www.mdpi.com/2079-9292/11/15/2330.

[57] Boya Zhao et al. "An Improved Aggregated-Mosaic Method for the Sparse Object Detection of Remote Sensing Imagery". In: *Remote Sensing* 13.13 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13132602. URL: https://www.mdpi.com/2072-4292/13/13/2602.

[58] Olarewaju Lawal, Huamin Zhao, and Z Fan. "Ablation studies on YOLOFruit detection algorithm for fruit harvesting robot using deep learning". In: *IOP Conference Series: Earth and Environmental Science* 922 (Nov. 2021), p. 012001. DOI: 10.1088/1755-1315/922/1/012001.

[59] Shun Li et al. "TC-YOLOv5: rapid detection of floating debris on raspberry Pi 4B". In: *Journal of Real-Time Image Processing* 20 (Feb. 2023). DOI: 10.1007/s11554-023-01265-z.

[60] Gang Wang et al. "UAV-YOLOv8: A Small-Object-Detection Model Based on Improved YOLOv8 for UAV Aerial Photography Scenarios". In: *Sensors* 23.16 (2023). ISSN: 1424-8220. DOI: 10.3390/s23167190. URL: https://www.mdpi.com/1424-8220/23/16/7190.

[61] Haitong Lou et al. "DC-YOLOv8: Small-Size Object Detection Algorithm Based on Camera Sensor". In: *Electronics* 12.10 (2023). ISSN: 2079-9292. DOI: 10.3390/electronics12102323. URL: https://www.mdpi.com/2079-9292/12/10/2323.

[62] Matteo Bordin. "Autonomous driving from the sky: study, design and evaluation of communication techniques between UAVs and autonomous cars". Academic Year 2020-2021. Master's Thesis. Padova, Italy: Università degli Studi di Padova, 2021.

[63] Dawei Du et al. "VisDrone-DET2019: The Vision Meets Drone Object Detection in Image Challenge Results". In: Oct. 2019. DOI: 10.1109/ICCVW.2019.00030.

[64] *Raspberry Pi Documentation*. Accessed: date-of-access. 2024. URL: https://www.raspberrypi.com/documentation/.

[65] Ultralytics. *Raspberry Pi Benchmark Results*. Accessed: 2024-05-19. 2024. URL: https://docs.ultralytics.com/guides/raspberry-pi/#__tabbed_3_3.

[66] Ultralytics. *Ultralytics YOLOv5 Model Export Documentation*. Accessed: 2024-05-19. 2024. URL: https://docs.ultralytics.com/yolov5/tutorials/model_export/#colab-pro-cpu.

[67] Raghuraman Krishnamoorthi. "Quantizing deep convolutional networks for efficient inference: A whitepaper". In: *ArXiv* abs/1806.08342 (2018). URL: https://api.semanticscholar.org/CorpusID:49356451.

[68] Zheming Jin and Hal Finkel. "Analyzing Deep Learning Model Inferences for Image Classification using OpenVINO". In: *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2020), pp. 908–911. URL: https://api.semanticscholar.org/CorpusID:220891693.

[69] V. V. Zunin. "Intel OpenVINO Toolkit for Computer Vision: Object Detection and Semantic Segmentation". In: *2021 International Russian Automation Conference (RusAutoCon)* (2021), pp. 847–851. URL: https://api.semanticscholar.org/CorpusID:237550161.

[70] Nikita A. Andriyanov. "Analysis of the Acceleration of Neural Networks Inference on Intel Processors Based on OpenVINO Toolkit". In: *2020 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)* (2020), pp. 1–5. URL: https://api.semanticscholar.org/CorpusID:221160380.

[71] Yizhi Liu et al. "Optimizing CNN Model Inference on CPUs". In: *ArXiv* abs/1809.02697 (2018). URL: https://api.semanticscholar.org/CorpusID:52183221.

[72] Alexander V. Demidovskij et al. "OpenVINO Deep Learning Workbench: A Platform for Model Optimization, Analysis and Deployment". In: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)* (2020), pp. 661–668. URL: https://api.semanticscholar.org/CorpusID:229703495.

[73] Siwen Wang, Ying Li, and Sihai Qiao. "ALF-YOLO: Enhanced YOLOv8 based on multiscale attention feature fusion for ship detection". In: *Ocean Engineering* (2024). URL: https://api.semanticscholar.org/CorpusID:270053680.

[74] Tahreer Abdul Ridha Shyaa and Ahmed A. Hashim. "Enhancing real human detection and people counting using YOLOv8". In: *BIO Web of Conferences* (2024). URL: https://api.semanticscholar.org/CorpusID:268984394.

[75] Eben Panja, Hendry Hendry, and Christine Dewi. "YOLOv8 Analysis for Vehicle Classification Under Various Image Conditions". In: *Scientific Journal of Informatics* (2024). URL: https://api.semanticscholar.org/CorpusID:268586114.

[76] Edmundo Casas et al. "YOLOv5 vs. YOLOv8: Performance Benchmarking in Wildfire and Smoke Detection Scenarios". In: *Journal of Image and Graphics* (2024). URL: https://api.semanticscholar.org/CorpusID:269056481.

[77] Mini Han Wang et al. "Optimizing Real-Time Trichiasis Object Detection: A Comparative Analysis of YOLOv5 and YOLOv8 Performance Metrics". In: *2023 9th International Conference on Systems and Informatics (ICSAI)* (2023), pp. 1–5. URL: https://api.semanticscholar.org/CorpusID:267576314.

[78] Mahmudul Islam Masum et al. "YOLOv5 vs. YOLOv8 in Marine Fisheries: Balancing Class Detection and Instance Count". In: *ArXiv* abs/2405.02312 (2024). URL: https://api.semanticscholar.org/CorpusID:269605788.

[79] Mini Han Wang et al. "Comparative Analysis of YOLOv5 and YOLOv8 for Tear Film Lipid Layer Detection: Architectural Disparities, Performance Metrics, and Future Implications". In: *2023 International Conference on Computer Science and Automation Technology (CSAT)* (2023), pp. 147–150. URL: https://api.semanticscholar.org/CorpusID:268713424.

[80] Joshua Andle et al. “The Stanford Drone Dataset Is More Complex Than We Think: An Analysis of Key Characteristics”. In: *IEEE Transactions on Intelligent Vehicles* 8 (2022), pp. 1863–1873. URL: https://api.semanticscholar.org/CorpusID:247596716.

[81] Pablo Ruiz-Ponce et al. “POSEIDON: A Data Augmentation Tool for Small Object Detection Datasets in Maritime Environments”. In: *Sensors (Basel, Switzerland)* 23 (2023). URL: https://api.semanticscholar.org/CorpusID:257945589.

[82] Hazim Shakhatreh et al. “Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges”. English (US). In: *IEEE Access* 7 (2019). Publisher Copyright: © 2013 IEEE., pp. 48572–48634. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2909530.

[83] Ludovic Apvrille, Tullio Tanzi, and Jean-Luc Dugelay. “Autonomous Drones for Assisting Rescue Services within the context of Natural Disasters”. In: Jan. 2014.

Acknowledgments