University of the Aegean

School of Engineering

Department of Financial Management and Engineering

IMPROVING THE TRAINING PERFORMANCE OF THE YOLO ALGORITHM FOR DETECTING OBJECTS FROM UAV IMAGES RECORDED DURING MONITORING OF LOGISTIC FACILITIES

Lapsanis Panagiotis

Supervisor:	Prof. Georgios Dounias
Committee Members:	Associate Prof. Vasileios Zeimpekis
	Associate Prof. Vasileios Koutras

Chios, December 2024

To my family…

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Professor Emeritus Ioannis Minis for his continuous and invaluable feedback. His insights and guidance have not only improved the quality of this thesis but also contributed significantly to my personal growth. He consistently challenged me to push my boundaries and achieve more than I thought possible. Furthermore, I would like to thank Professor Georgios Dounias for kindly agreeing to become my supervisor following Mr. Minis' retirement, after many years of distinguished service.

I am deeply grateful to George Tepteris for guiding me through my first steps in the field of artificial intelligence. His boundless patience and willingness to answer my numerous questions were truly indispensable. Despite the volume of my questions, he was always kind and helpful.

In addition, I extend my gratitude to everyone in the DeOPSys lab of the Department of Financial and Management Engineering and to my colleague Anna Tsiflitzi for the strong teamwork and collaboration we developed throughout our projects.

Finally, I am thankful to my family and friends. My heartfelt appreciation goes especially to my mother for her unwavering presence, encouragement and support of my personal choices. The support I have received from everyone mentioned here has been vital to my progress.

Abstract

This thesis focuses on optimizing the training process of YOLOv4-p6 to detect and classify persons, small vehicles, large vehicles, and ships for surveillance applications in warehouses and ports. The optimization process involved fine-tuning the training hyperparameters of YOLOv4-p6 to maximize mean Average Precision (mAP). Several configurations achieved higher mAP results compared to the default settings. Consequently, this thesis examines the impact of each hyperparameter on YOLOv4-p6's training performance.

Specifically, we trained and tested YOLOv4-p6 using publicly available UAV annotated image datasets, including Aerial vehicle, DOTA, VisDrone-DET, Stanford drone, and DAC-SDC. These datasets were modified to contain only the selected classes, and then combined into a single dataset consisting of 76,872 images with 876,388 annotated objects. Subsequently, we divided the combined UAV dataset into training (80% of the combined dataset), validation (10%) and testing (10%) subsets.

The training hyperparameters of the YOLOv4-p6 algorithm were divided in two subsets. The first subset comprised those hyperparameters, the values of which depend on the characteristics of the training set. The values/levels of these hyperparameters were kept invariant throughout the tuning experiments. The second set comprised the hyperparameters to be tuned to optimize the training performance of YOLOv4-p6. Specifically, the second set included five (two-levels) hyperparameters, which led to the generation of thirty-two experiments using the Full-Factorial method $(2^5 = 32)$. For each of the 32 combinations we repeated the training and testing sessions to support the analysis of the results using Analysis of Variance (ANOVA). This resulted in 64 trained models.

The analysis of the mAP results by ANOVA revealed three statistically significant hyperparameters: image resolution, activation function, and anchor dimensions; furthermore, a three-way interaction has been identified as significant: among Non-Maximum Suppression, data augmentation, and anchor dimensions.

The best trained models (25^{th} and 29^{th}) achieved an average mAP value of 52% in validation and 53.3% in testing. This is in contrast with the lowest performing models, of which the mAP values were 39.8% in validation and 44% in testing. This supports our thesis that careful tuning of the hyperparameters during training may yield to major improvements in model effectiveness.

We also tested the best performing models on a new UAV dataset developed by the DeOPSys lab. They performed exceptionally well, achieving a value of average mAP up to 77.6% and 76.3%, respectively. This independent testing validates the quality of the trained models. More importantly it validates that the proposed hyperparameter tuning method enables effective training of high-performance YOLO models.

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στη βελτιστοποίηση της διαδικασίας εκπαίδευσης του συστήματος YOLOv4-p6 για την ανίχνευση και ταξινόμηση αντικειμένων από φωτογραφίες ή video και συγκεκριμένα ανθρώπων, μικρών οχημάτων, μεγάλων οχημάτων και πλοίων, με εφαρμογές επιτήρησης αποθηκευτικών χώρων και λιμένων. Η βελτιστοποίηση επιτεύχθηκε μέσω της κατάλληλης ρύθμισης των υπερπαραμέτρων εκπαίδευσης του YOLOv4-p6 με στόχο τη μεγιστοποίηση της μέσης τιμής της μέσης ακρίβειας (mean Average Precision ή εν συντομία mAP). Ορισμένοι συνδυασμοί υπερπαραμέτρων επέφεραν υψηλότερα αποτελέσματα mAP σε σύγκριση με τις προεπιλεγμένες στο σύστημα ρυθμίσεις. Η διπλωματική εργασία εξετάζει επίσης την επίδραση κάθε υπερπαραμέτρου στην απόδοση εκπαίδευσης του YOLOv4-p6.

Στο πλαίσιο της παρούσας έρευνας, εκπαιδεύσαμε το YOLOv4-p6 χρησιμοποιώντας δημόσια διαθέσιμα σύνολα δεδομένων εικόνων από UAV, όπως τα Aerial Vehicle, DOTA, VisDrone-DET, Stanford Drone και DAC-SDC. Τα εν λόγω σύνολα δεδομένων τροποποιήθηκαν ώστε να περιλαμβάνουν αποκλειστικά τις επιλεγμένες κατηγορίες αντικειμένων και στη συνέχεια συνδυάστηκαν σε ένα ενιαίο σύνολο δεδομένων που αποτελείται από 76.872 εικόνες με 876.388 επισημασμένα αντικείμενα. Στη συνέχεια, το ενιαίο σύνολο δεδομένων UAV διαιρέθηκε σε υποσύνολα εκπαίδευσης (80%), επικύρωσης (10%) και δοκιμής (10%).

Οι υπερπαράμετροι εκπαίδευσης του αλγορίθμου YOLOv4-p6 χωρίστηκαν σε δύο ομάδες. Η πρώτη ομάδα περιλάμβανε υπερπαραμέτρους των οποίων οι τιμές εξαρτώνται από τα χαρακτηριστικά του συνόλου δεδομένων εκπαίδευσης. Οι τιμές/επίπεδα αυτών των υπερπαραμέτρων διατηρήθηκαν αμετάβλητα κατά τη διάρκεια των πειραμάτων ρύθμισης. Η δεύτερη ομάδα περιλάμβανε τις υπερπαραμέτρους που ρυθμίστηκαν για τη βελτιστοποίηση της απόδοσης εκπαίδευσης του YOLOv4-p6. Συγκεκριμένα, η εν λόγω ομάδα περιλάμβανε πέντε υπερπαραμέτρους δύο επιπέδων, γεγονός που οδήγησε στη δημιουργία τριάντα δύο (32) πειραμάτων χρησιμοποιώντας τη μέθοδο Πλήρους Παραγοντικού σχεδιασμού (Full-Factorial design, $2^5 = 32$). Για κάθε ένα από τους 32 συνδυασμούς, επαναλάβαμε τους κύκλους εκπαίδευσης και δοκιμής ώστε να υποστηριχθεί η ανάλυση των αποτελεσμάτων μέσω της Ανάλυσης Διακύμανσης (ANOVA). Αυτό οδήγησε σε 64 εκπαιδευμένα μοντέλα.

Η ανάλυση των τιμών του mAP που προέκυψαν από την διαδικασία εκπαίδευσης και δοκιμών μέσω ANOVA αποκάλυψε τρεις στατιστικά σημαντικές υπερπαραμέτρους: την ανάλυση εικόνας, τη συνάρτηση ενεργοποίησης και τις διαστάσεις των περιγραμμάτων anchors των αντικειμένων. Επιπλέον, εντοπίστηκε ως σημαντική μια τριπλή αλληλεπίδραση μεταξύ της καταστολής μη μέγιστων τιμών (Non-Maximum Suppression), της αύξησης δεδομένων (data augmentation) και των διαστάσεων των περιγραμμάτων anchors.

Τα καλύτερα εκπαιδευμένα μοντέλα (25^o και 29^o) πέτυχαν μέση τιμή mAP 52% στην διαδικασία εκπαίδευσης/επικύρωσης και 53.3% στη διαδικασία δοκιμών. Επισημαίνεται ότι τα μοντέλα με τη χαμηλότερη απόδοση είχαν τιμές mAP 39.8% στην επικύρωση και 44% στη δοκιμή. Αυτό υποστηρίζει την υπόθεσή μας ότι η προσεκτική ρύθμιση των υπερπαραμέτρων κατά την εκπαίδευση μπορεί να οδηγήσει σε σημαντικές βελτιώσεις στην αποτελεσματικότητα του μοντέλου.

Επιπλέον, τα καλύτερα εκπαιδευμένα μοντέλα αξιολογήθηκαν σε ένα νέο σύνολο δεδομένων UAV που αναπτύχθηκε από το εργαστήριο DeOPSys. Τα μοντέλα κατέγραψαν εξαιρετική απόδοση, επιτυγχάνοντας μέση τιμή mAP έως 77.6% και 76.3%, αντίστοιχα. Η ανεξάρτητη αυτή αξιολόγηση επιβεβαιώνει την ποιότητα των εκπαιδευμένων μοντέλων. Πιο σημαντικό αποτέλεσμα της ανεξάρτητης αυτής δοκιμής είναι η επιβεβαίωση του γεγονότος ότι η προτεινόμενη μέθοδος ρύθμισης υπερπαραμέτρων επιτρέπει την αποτελεσματική εκπαίδευση μοντέλων YOLO υψηλής απόδοσης.

Table of Contents

Table of Figures

List of Tables

Chapter 1 Introduction

Artificial Intelligence (AI) is replicating human intelligence within computational systems or machinery furnished with appropriate software, demonstrating proficiency in executing intricate tasks within the domain of human expertise. Leveraging the capabilities of AI, several logistics-related applications have been developed, some of which focus on the process safeguarding warehouse facilities or even ports. For such environments, AI powered systems are sought for effective area surveillance in order to prevent unauthorized invasion, mitigate losses, organize routes for personnel, machinery, etc.

Object detection is a key prerequisite for such surveillance applications. Through object detection and image classification, once the information has been extracted, the systems are able to perform actions or make recommendations based on the extracted information. Note that there is a notable disparity between image classification and object detection. Image classification identifies the category of the most prominent entity within an image, while object detection encompasses the identification, position and dimensions of every discernible object within the image.

The aim of the present thesis is to enhance the performance of such applications for object detection in surveillance of logistics facilities. For this reason, we use the computer vision framework YOLO (You Only Look Once), using an advanced version of it, the YOLOv4-p6. After we analyze the structure and functionality of YOLOv4-p6, our study focuses on improving the training process of YOLOv4-p6. To do so, we systematically explore the space of training hyperparameters to identify the hyperparameter combination that results in training a YOLOv4-p6 model that achieves the highest mean Average Precision (mAP). Furthermore, by systematically experimenting with the training hyperparameters we seek to identify their effects and the effects of their interaction on the performance of training.

The structure of the remainder of the thesis is as follows: Chapter 2 overviews the first three YOLO algorithms and provides information about the functional framework, architectural structure, and novel features of YOLOv4 and Scaled YOLOv4 algorithms. Chapter 3 outlines the methodological steps for tuning the model's training hyperparameters. These encompass data preparation, the selection of the training hyperparameters, and the experimental design that leads to the optimal hyperparameter combination. Chapter 4 presents the analysis of the experimental outcomes. This involves the evaluation of mean Average Precision (mAP) performance metric, as well as the utilization of Analysis of Variance (ANOVA). Chapter 5 presents the average mAP results from testing the optimal YOLOv4-p6 models with our lab's UAV dataset. Finally, the conclusions of this thesis are presented in Chapter 6.

Chapter 2 Background of YOLO models

This Chapter begins with an overview of YOLO, YOLOv2, and YOLOv3 algorithms. It is followed by a detailed analysis of the YOLOv4 and Scaled-YOLOv4 models, since our work has been performed using the YOLOv4 variants. Specifically, attention is directed towards the functionality of YOLOv4 and YOLOv4-p6, along with their neural network architectures. At the end of this Chapter, there is a detailed presentation of the performance metrics, which will be used in Chapter 4 to evaluate the results of our experiments.

2.1 Introduction to the various YOLO versions

Figure 2.1 presents the evolution of the YOLO algorithm. Redmon et al., introduced YOLO (You Only Look Once) through a seminal scientific paper (Redmon et al., 2015). This seminal paper is entitled "*You Only Look Once: Unified, Real-Time Object Detection*" and the related research proved to be a revolutionary algorithm that led to the creation of various YOLO versions. Subsequently in 2016, Joseph Redmon and Ali Farhadi presented their new findings in a paper entitled as *"YOLO9000: Enhanced, Accelerated, Empowered"* and introduced YOLO9000, which is also recognized as the YOLOv2 algorithm (Redmon and Farhadi, 2016a). Following on from YOLOv2, in 2018 the YOLOv3 algorithm was introduced and is detailed in "*YOLOv3: An Incremental Advancement*", demonstrating significant improvements with three detection scales and an optimized backbone network (Redmon and Farhadi, 2018).

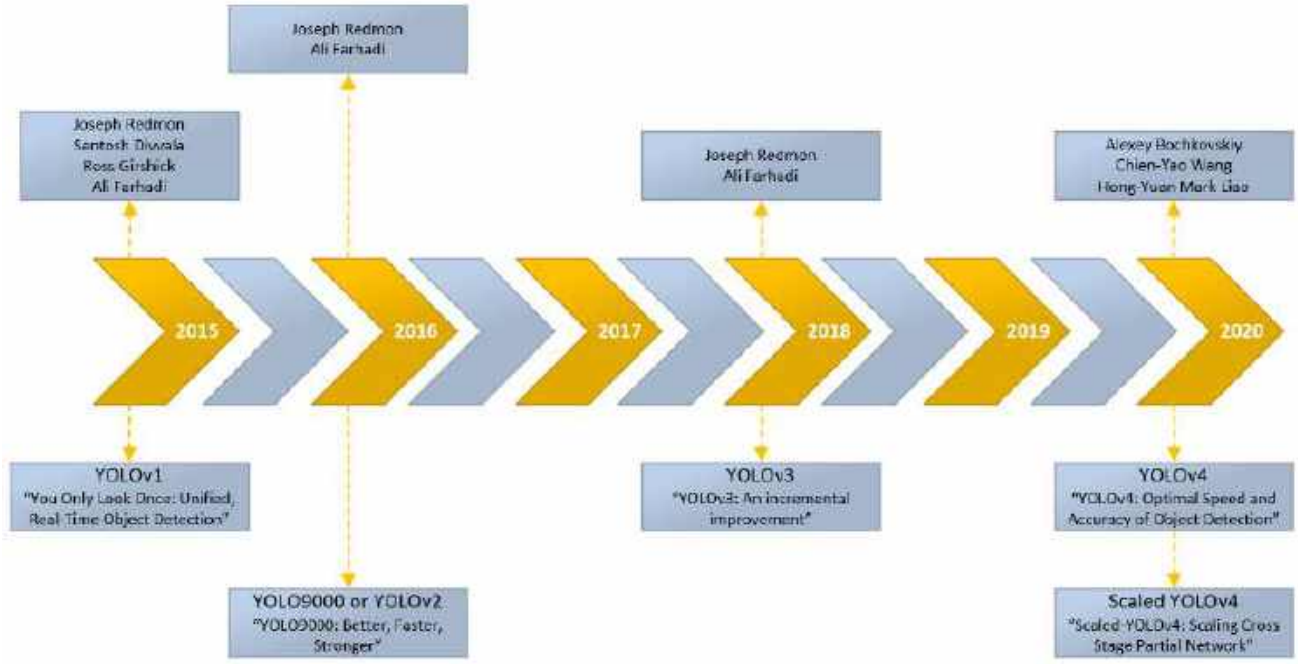

Figure 2.1 *The development of YOLO models over the years (Keita, 2022)*

The subsequent edition of YOLOv4 prompted Joseph Redmon to withdraw from further YOLO algorithm development, citing ethical reasons. Specifically, he was concerned that object detection algorithms would be used for military applications (Synced, 2020). Consequently, under the mentorship of Alexey Bochkovskiy, a new scientific team led by Chien-Yao Wang and Hong-Yuan Mark Liao undertook the responsibility for the continuous enhancement and development of YOLOv4, as detailed in *"YOLOv4:*

Optimal Speed and Accuracy of Object Detection" (Bochkovskiy et al., 2020). In 2020, the same research team revealed modified versions based on YOLOv4, namely the Scaled-YOLOv4 algorithms, which were designed to support larger input sizes to improve the overall accuracy. The related paper for the Scaled-YOLOv4 algorithms, entitled "*Scaled-YOLOv4: Scaling Cross Stage Partial Network*", details the advancements and adjustments implemented in these version of YOLO (Wang et al., 2020).

2.2 Earlier YOLO versions

2.2.1 You Only Look Once version 1

The YOLO (You Only Look Once) algorithm represents the first entry in the YOLO series of object detection models. Unlike traditional object detectors that treat detection as a classification problem, YOLO approaches it as a regression problem, accelerating the process to predict bounding boxes and class probabilities in a single network pass (Redmon et al., 2016). Specifically, the developers were seeking to create a single pass neural network capable of determining the class identity of objects and defining their spatial coordinates within the image through a single iteration.

At the beginning of the training process, the YOLO algorithm resizes its input images based on selected predetermined dimensions, which must be divisible by 32. For instance, YOLO applies dimensions of 448x448 to its input images for height and width, respectively. Then, it divides the input images into a grid of dimensions S x S, where S is typically 7 or 13. It is assumed that each grid cell contains a centered object and each cell predicts the various bounding box sizes of the (presumed) objects, along with their corresponding confidence scores and probabilities for the object to belong to the different available classes (El Aidouni, 2019). For each of the individual grid cells i, YOLO performs the following predictions:

- The total of two bounding boxes (B), also known as anchor boxes, as discussed in Appendices A.1 and A.2.
- Each of the B anchor boxes has a confidence score (C_i) and spatial coordinates (x, y, w, h), along with class probabilities $\left(p_i(c)\right)$ for each class ($c = 1, 2, 3, 4, \ldots, C$).

These forecasts are structured into a $S\ x\ S * (b\ *\ 5 + C)$ tensor, where "5" represents the dimensions of the vector (C_i, x, y, w, h). Here, (x, y) is the center of the bounding box relative to the grid cell, (w, h) is the width and height of the bounding box relative to the entire image, and C_i represents the confidence score (see Figure 2.2).

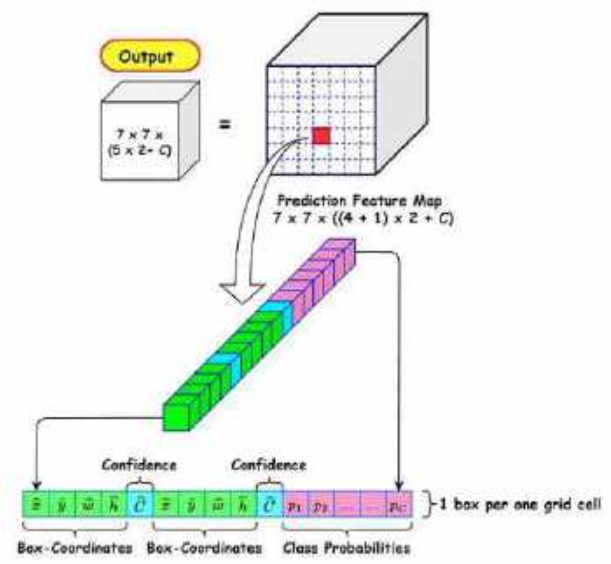

Figure 2.2 *The output vector for each anchor box within every grid cell is generated by the YOLO algorithm (WikiDocs, 2023)*

The YOLO model is designed based on a convolutional neural network architecture, capable of directly anticipating objects through the application of the two bounding boxes and identifying class probabilities for the object in each box. Compared to other object detection algorithms of the time, the design is quite simple. It consists of 4 Max Pooling layers, 2 fully connected layers, and 24 convolutional layers, as illustrated in Figure 2.3 (Kamal, 2021a).

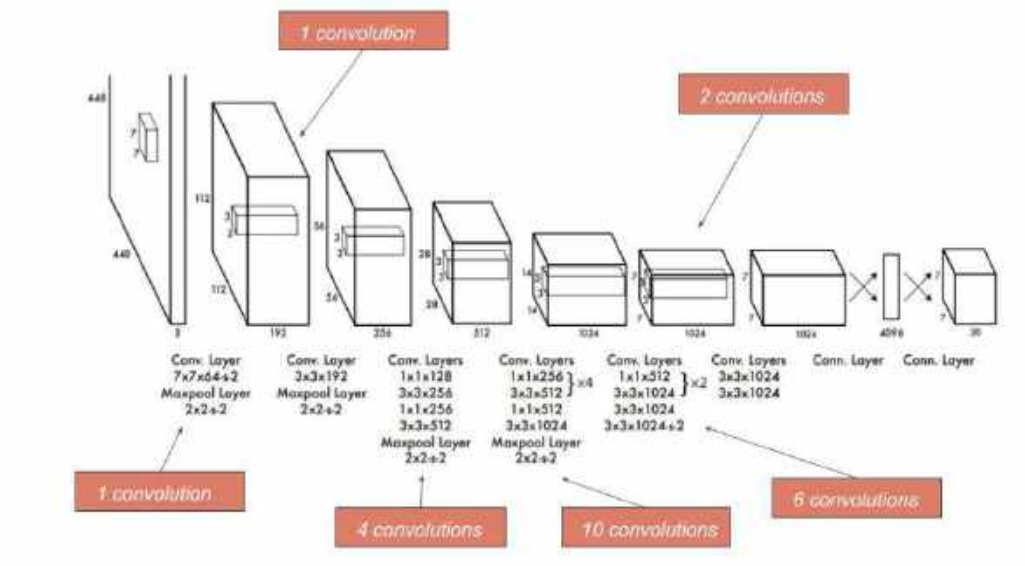

Figure 2.3 *YOLO neural network architecture (El Aidouni, 2019)*

2.2.2 You Only Look Once version 2

Following the groundbreaking innovation of YOLOv1, YOLOv2 (known also as YOLO9000) includes several enhancements that improve the model training process and increase object detection accuracy (Redmon and Farhadi, 2016a). The changes that are included in YOLOv2 are:

- **Batch Normalization**: As explained in Appendix C.5, the incorporation of batch normalization into the structure improves the convergence of the model, speeding up the training process. It eliminates the requirement to use alternative normalization techniques such as Dropout, while concurrently decreasing the risk of overfitting (pawangfg, 2020).
- **High Resolution Classifier**: The YOLOv1 algorithm trains its classifier network with an image of size 224×224 and then increases the resolution to 448x448 pixels for the purpose of improves detection. When moving to detection, the network must learn to identify objects and adjust to changes in image resolution. This causes a reduction in mean Average Precision (mAP). On the other hand, the initial training of the YOLOv2 algorithm uses images with dimensions of 224×224. Consequently, a refinement process was performed, where the classification network is further trained at the resolution of 448×448 for a duration of 10 epochs on the ImageNet dataset before starting the training for object detection (Kamal, 2021b).
- **Evolutionary utilization of anchor boxes**: In YOLOv1, the objective is to locate an object in the grid cell that includes the spatial midpoint of the object. Now see Figure 2.4, where the red cell is tasked with detecting multiple objects (not one as in YOLOv1)To address this challenge, the creators of YOLOv2 tried to enable the grid cell to detect multiple objects by introducing k bounding boxes (Kamal, 2021a).

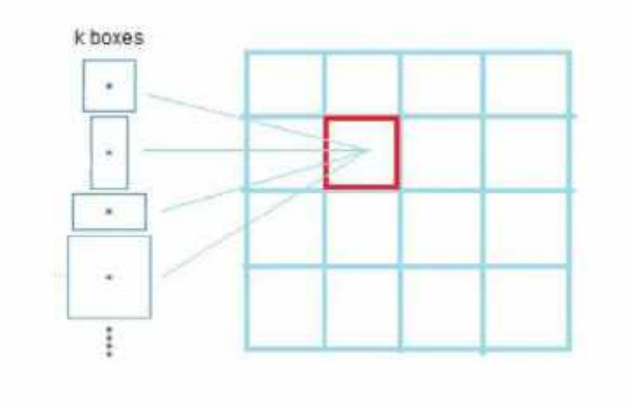

Figure 2.4 *The application of k bounding boxes (Kamal, 2021a)*

Consequently, YOLOv2 improves object detection by introducing anchor boxes, which replace the fully connected layers that YOLOv1 used towards the end of its network. As explained in Appendix B.2, this method enhances the accuracy of object detection by using predefined boxes of various sizes and aspect ratios. In contrast to YOLOv1, which directly predicts bounding box coordinates for objects without the application of anchor boxes (see Figure 2.5).

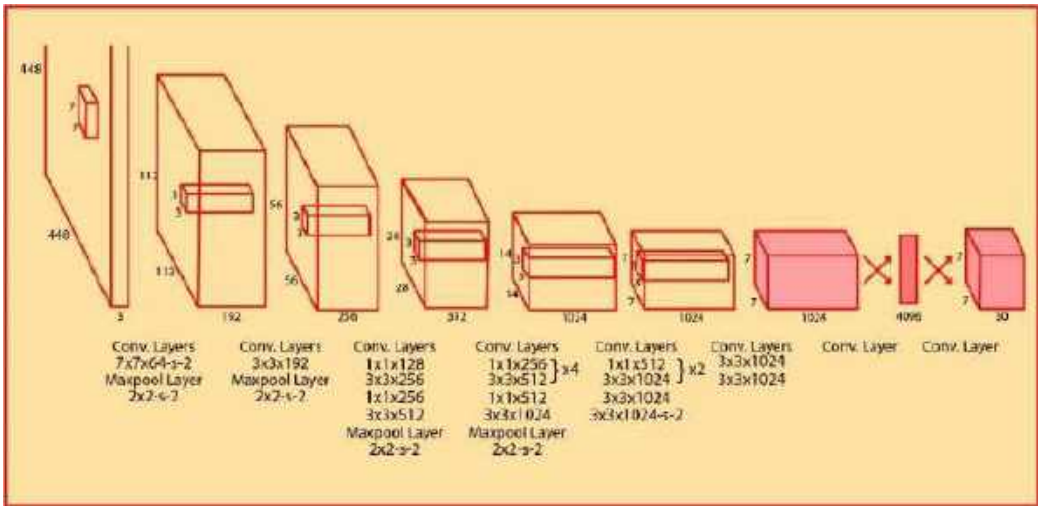

Figure 2.5 *Depicts the removal of fully connected layers highlighted in red colour (pawangfg, 2020)*

The architecture of YOLOv2 consists of 19 convolutional layers and 5 Max Pooling layers (Kamal, 2021b). As shown in Figure 2.6, the trainable architecture used as the backbone is Darknet-19, chosen for its comparatively lower computational requirements compared to alternative architectures.

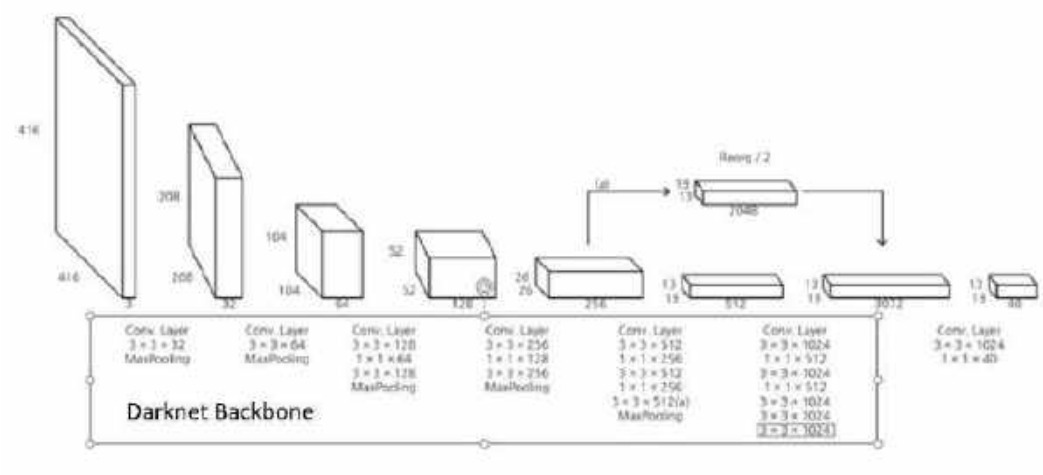

Figure 2.6 *YOLOv2 neural network architecture (Zhang, 2020)*

2.2.3 You Only Look Once version 3

YOLOv3 (Redmon and Farhadi, 2018) is the upgraded version of YOLOv2. It is an advancement from its predecessors, with capacity to swiftly recognize objects in both images and video streams.

At its beginning, the YOLOv3 architecture was created as a 53-layer network, initially pretrained on the ImageNet dataset, and designed with focus on the detection task. Following each layer within the 53-layer architecture, it includes a batch normalization layer along with the incorporation of the Leaky ReLU activation function. These 53 layers are contained in an architecture known as Darknet-53, which is an updated version of Darknet-19. Following the Darknet-53, there are 53 additional layers to its structure, resulting in an overall 106 layers (see Figure 2.7) that make up the YOLOv3 model (Nagpal, 2023).

The enhancements of YOLOv3 include the following:

- **Objectness Score** functions as a statistical metric, which estimates the probability of the existence of an object within a specified region of an image. This assists in reducing false positive detections (Kamal, 2021a)
- **Multi-label prediction** involves the existence of many overlapping labels within a dataset. The application of SoftMax function for class prediction introduces the condition that each bounding box corresponds exclusively to a single class. In comparison, YOLOv3 avoids using SoftMax function, and instead uses independent logistic classifiers assigned to each class. Durning the training process of the model, the utilization of binary cross-entropy loss is adopted to optimize the class prediction process (Kamal, 2021a)
- **Multi-scale predictions**: The YOLOv3 model generates predictions at three scales, enabling it to recognize small objects with greater accuracy (Kamal, 2021a)
- **Small objects detection**: YOLOv3 demonstrates improved efficiency in small object detection, due to the incorporation of shortcut connections. The application of shortcut connections allows the retrieval of more complicated details from the initial feature map. However, in contrast to YOLOv2, YOLOv3 displays reduced performance in the detection of medium and large objects (Kamal, 2021a).

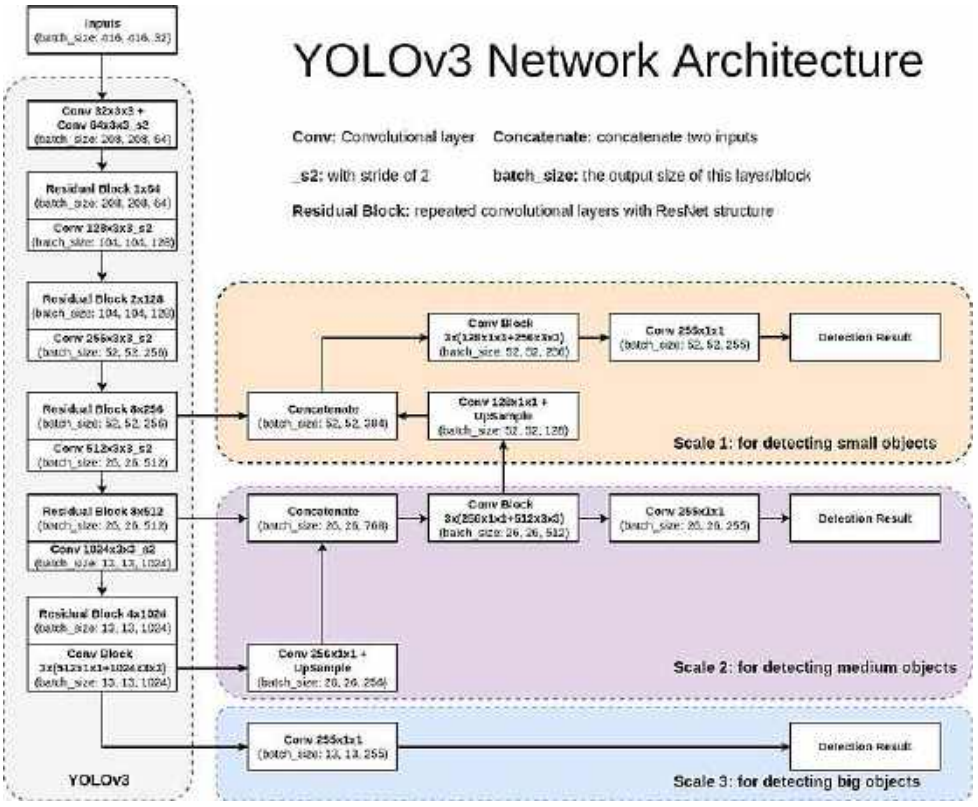

Figure 2.7 *YOLOv3 neural network architecture (Afif et al., 2020)*

2.3 Drill down to You Only Look Once version 4 algorithm

2.3.1 You Only Look Once version 4

YOLOv4 is designed for high-speed object detection and classification in real-time scenarios using general purpose GPUs. It exhibits superior efficiency, heightened precision, and increased consistency compared to previous versions (Bochkovskiy et al., 2020). The original network design, which consists of 161 layers, is made up of three components: the backbone, neck, and head. Additionally, YOLOv4 incorporates new methodologies referred to as "Bag of Freebies" (BoF) and "Bag of Specials" (BoS) to increase its performance.

As backbone in this network CSPDarknet-53 is used (Bochkovskiy et al., 2020), which is an updated version of Darknet-53 (Redmon and Farhadi, 2018) that integrates cross-stage partial connections (CSP) , as further explained in Appendix C.10. It combines the features of Darknet-53 with CSP connections to increase both precision and efficiency within the network.

The neck component of the model incorporates of Spatial Pyramid Pooling (SPP) (He et al., 2014) -see Appendix C.11 for more details, and Path Aggregation Network (PANet) (Liu et al., 2018), see Appendix C.12. SPP uses a technique known as pooling, which addresses the challenge of handling objects with varying sizes and scales. Following the SPP the Path Aggregation Network is applied, which is used to improve the object detection process through the preservation of spatial details. This enables the model to create more accurate bounding boxes.

The last component is the detection heads that analyze the resulting feature maps. The YOLOv4 model, like YOLOv3, uses three detection heads for the detection and classification process. These detection heads predict bounding box coordinates along with class probabilities for the objects identified within the input image.

2.3.2 Backbone network of YOLOv4

YOLOv4 supports various backbones, including VGG16 (Simonyan and Zisserman, 2015), ResNet-50 (He et al., 2015a), SpineNet (Du et al., 2020), EfficientNet-B0/B7 (Tan and Le, 2020), CSPResNeXt50 (Bochkovskiy et al., 2020), and CSPDarknet-53 (Bochkovskiy et al., 2020). From the above backbones, as indicated also above, the selected one is CSPDarknet-53. It is an updated version of Darknet-53 that integrates the CSP connections at the top, while the feature extraction remains at the bottom, as seen in Darknet-53.

Figure 2.8 illustrates Darknet-53 that includes a total of 53 convolutional layers, with a mix of $1x1$ and $3x3$ filter sizes. Specifically, 29 of these layers have filters of size of $3x3$ to increase the depth of the network and its capacity for feature extraction. Each convolution layer within Darknet-53 is connected to a batch normalization (BN) layer and a leaky-ReLU activation layer.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.8 *Representation of the Darknet-53 backbone (Redmon and Farhadi, 2018)*

With the application of CSP, the network divides the feature map of the primary layer into two different paths and then combines them to produce an output feature map and proceeds in **successive** layers as shown in Figure 2.9. In the CSP block, the feature map is divided into two paths: Part 1 and Part 2. Part 1 processes the feature map using an $1x1$ convolutional layer with a stride of 1, which retains spatial information. Part 2 processes the feature map using a $1x1$ convolution followed by a $3x3$ convolution, both with a stride of 1, and Mish activation functions, which help reduce dimensions and capture spatial information. After these processes, the resulting feature map of Part 2 is concatenated with the output feature map of Part 1, leading to a more complete feature map.

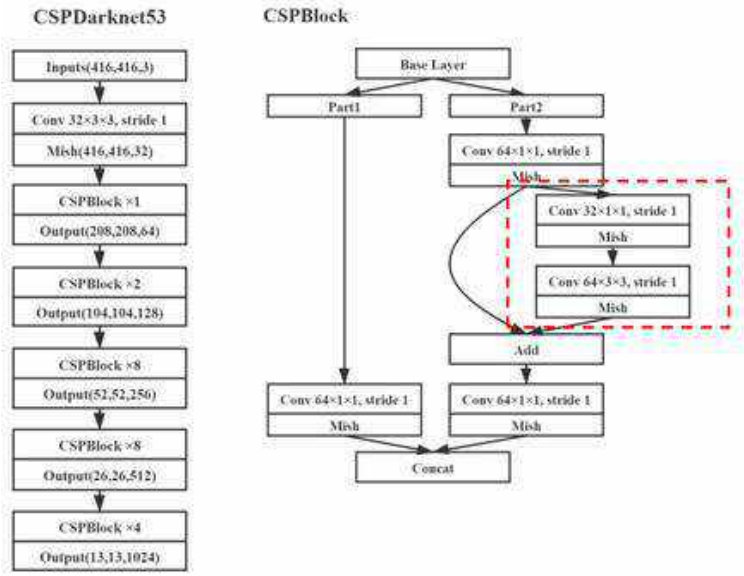

Figure 2.9 *Representation of the CSPDarknet-53 backbone (Xu et al., 2021)*

The CSPDarknet-53 architecture demonstrates increased accuracy in object detection and its classification efficiency can be further optimized through the integration of methodologies such as Mish, as elaborated in Chapter 3.

2.3.3 YOLOv4 network analysis

This Subsection provides a more detailed analysis on YOLOv4's neural network architecture, focusing on its backbone, neck, and head components, as shown in Figure 2.10. We begin with the nomenclature of the components contained in Figure 2.10.

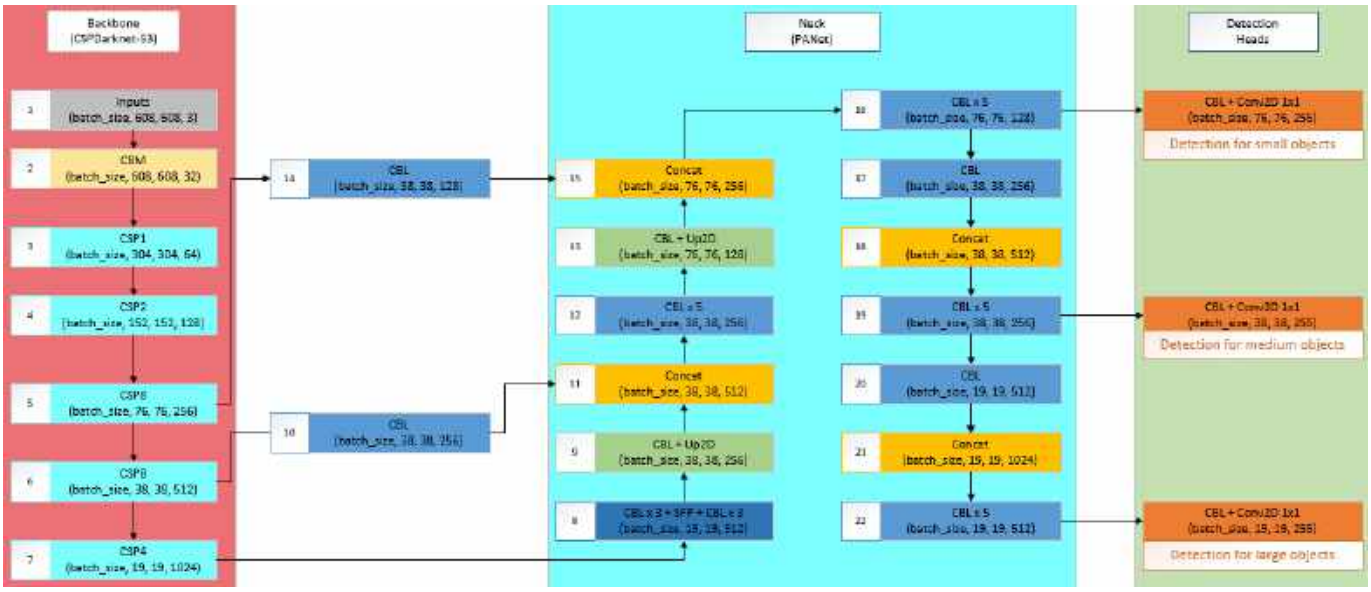

Figure 2.10 *YOLOv4 neural network architecture (Tsang, 2022)*

Nomenclature of the network's components

In steps 9 and 13 of Figure 2.10, "Up2D" refers to the operation of upsampling, which is discussed in appendix C.7. In the same Figure "Concat" represents the concatenation operation which is achieved through the use of route layers (see Appendix C.9). In Figure 2.11 the term "CSPX" represents the cross-stage partial structure. Lastly, Figure 2.12 illustrates the "CBL" layer, which combines a convolution layer (see Appendix C.3) with batch normalization and the Leaky ReLU activation function. Similarly, the "CBM" layer combines a convolution layer with batch normalization and the Mish activation function (Zhang et al., 2022).

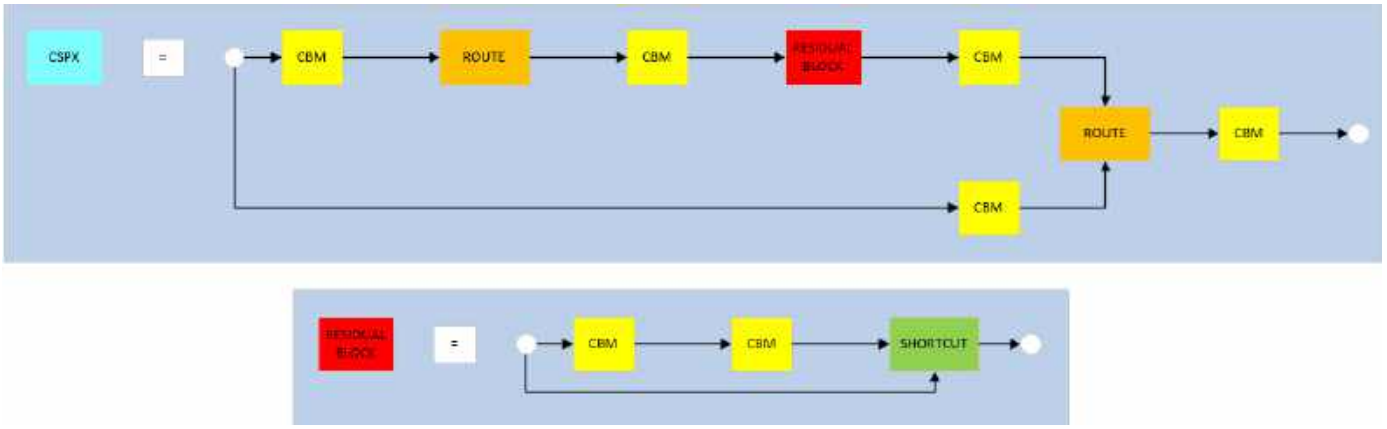

Figure 2.11 *Representation of the CSP block in the top box and the residual block in the bottom box*

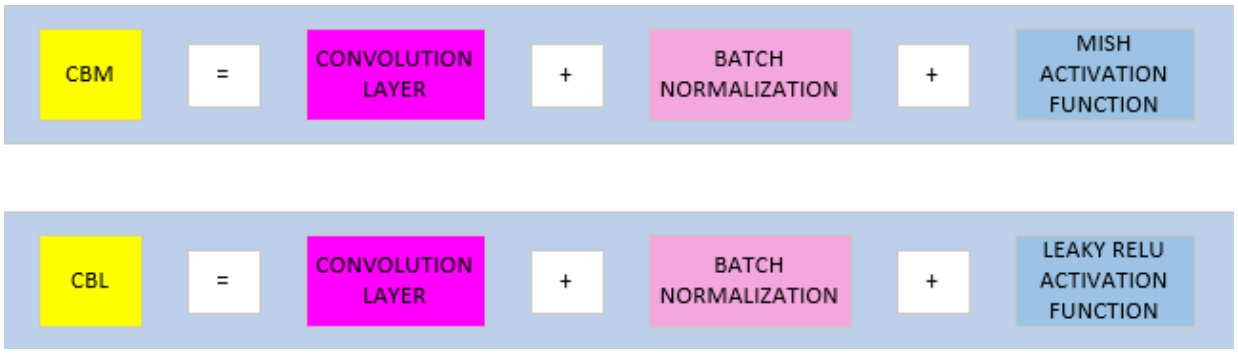

Figure 2.12 *Representation of the CBM layer in the top box and the CBL layer in the bottom box (Tsang, 2022)*

The network's backbone component

At the very beginning of the algorithm, a $608x608$ image is fed into the CSPDarknet-53 and then passes through a pair of convolutional layers (see Figure 2.10). The first convolutional layer (step 1) uses 32 filters of size $3x3$ and a stride of 1, resulting in an output feature map of $608x608x32$ (see Appendix C.2). Subsequently, the output is fed into the second convolutional layer (step 2), which uses 64 filters of size $3x3$ with a stride of 2. This results in a feature map with dimensions of $304x304x64$. The reduction in dimensions from 608 to 304 is achieved through the filtering process by applying a stride of 2, while the increased depth to 64 corresponds to the number of filters used in that convolutional layer.

Subsequently, the downsampled image is processed through the CSP blocks, which are in total five (corresponding to steps 3 through 7). As shown in Figure 2.10, all CSP blocks contain residual blocks, CBM and route layers, except for CSP1, which contains only one residual block (see Appendix C.8). As the downsampled image with feature map of $304x304x64$ progresses through the CSP1 block (step 3), it undergoes:

- A convolutional layer with 64 filters (the same number of filters as before), using a stride of 1 and a size of $1x1$. This results in a feature map of $304x304x64$
- A route layer that concatenates feature maps from previous layers to allow information flow across different scales and resolutions. This results in a feature map of $304x304x64$

- A convolutional layer with 64 filters (the same number of filters as before), using a stride of 1 and a size of $1x1$. This results in a feature map of $304x304x64$
- A convolutional layer with 32 filters (halves the number of filters), using a stride of 1 and a size of $1x1$. This results in a feature map of $304x304x32$
- A convolutional layer with64 filters (doubles the number of filters to preserve spatial information due to using a larger size for this layer), with a stride of 1 and a size of $3x3$. This results in a feature map of $304x304x64$
- A shortcut layer that functions as a residual block. This results in a feature map of $304x304x64$
- A convolutional layer with 64 filters using a stride of 1 and a size of $1x1$. This results in a feature map of $304x304x64$
- A route layer that concatenates feature maps from previous layers. This results in a feature map of $304x304x64$
- A convolutional layer with 64 filters using a stride of 1 and a size of $1x1$. This results in a feature map of $304x304x64$.

The CSP1 block concludes with an output feature map of $304x304x64$. Following that, the downsampled image progresses through CSP2 (step 4), CSP8 (step 5), CSP8 (step 6), and CSP4 (step 7). Before entering the CSP2 (step 4) block, a downsampling layer (explained in Appendices C.1 and C.6) reduces the input image size by half, doubles the number of filters and uses a filter size of $3x3$. Consequently, the feature map after the downsampling operation is $152x152x128$. Then, the downsampled image moves through the CSP2 block, which is similar to CSP1 but includes an additional residual block. This extra residual block is added as follows:

- A convolutional layer with 64 filters (halves the number of filters), using a stride of 1 and a size of $1x1$. This results in a feature map of $152x152x64$
- A route layer that concatenates feature maps from previous layers. This results in a feature map of $152x152x128$
- A convolutional layer with 64 filters using a stride of 1 and a size of $1x1$. This results in a feature map of $152x152x64$
- A convolutional layer with 64 filters, using a stride of 1 and a size of $1x1$. This results in a feature map of $152x152x64$
- A convolutional layer with 64 filters using a stride of 1 and a size of $3x3$. This results in a feature map of $152x152x64$
- A shortcut layer that functions as a residual block. This results in a feature map of $152x152x64$
- The following three layers represent a single residual block, where an additional one is included in CSP2
    - A convolutional layer with 64 filters using a stride of 1 and a size of $1x1$. This results in a feature map of $152x152x64$
    - A convolutional layer with 64 filters using a stride of 1 and a size of $3x3$. This results in a feature map of $152x152x64$

    - A shortcut layer that functions as a residual block. This results in a feature map of $152x152x64$.
- A convolutional layer with 64 filters, using a stride of 1 and a size of $1x1$. This results in a feature map of $152x152x64$
- A route layer that concatenates feature maps from previous layers. This results in a feature map of $152x152x64$
- A convolutional layer with 128 filters (doubles the number of filters), using a stride of 1 and a size of $1x1$. This results in a feature map of $152x152x128$.

The CSP2 block concludes with an output feature map of $152x152x128$. The remaining CSP blocks function the same way as CSP2 and follow the same pattern. For instance, CSP8 is like CSP2, but with six more residual blocks added. Notice the number that follows CSP indicates the number of residual blocks. Then, we add them in the same way as we added the extra block in CSP2. Consequently, the resulting feature maps of CSP8, CSP8 and CSP4 blocks are:

- The resulting feature map of CSP8 (step 5) block is $76x76x256$
- The resulting feature map of CSP8 (step 6) block is $38x38x512$
- The resulting feature map of CSP4 (step 7) block is $19x19x1024$.

Before entering the neck component of the YOLOv4 architecture, there are three convolutional layers (CBL). These layers are located between the conclusion of CSP4 block and the beginning of the SPP (see Figure 2.13). They are described as follows:

- A convolutional layer with 512 filters (halves the number of filters in the resulting CSP4 feature map), using a stride of 1 and a size of $1x1$. This results in a feature map of $19x19x512$
- A convolutional layer with 1024 filters (doubles the number of filters to preserve spatial information due to using a larger size for this layer), with a stride of 1 and a size of $3x3$. This results in a feature map of $19x19x1024$
- A convolutional layer with 512 filters (halves the number of filters), using a stride of 1 and a size of $1x1$. This results in a feature map of $19x19x512$.

The final three convolutional layers of the backbone component result in a feature map of $19x19x512$, which passes to SPP.

The neck component of the network

The description of the network's neck component (depicted by the cyan box of Figure 2.10) performs the following steps:

- In step 8, the model applies Spatial Pyramid Pooling (SPP). This module is positioned between the backbone and the neck. After applying three different pooling processes in 13x13, 9x9, and 5x5 dimensions, the feature maps of varying scales are merged with the original feature maps to generate the combined output. Moreover, convolutional layers (CBL) are implemented three times

both before (as mentioned at the end of backbone component) and after the SPP, totaling to six instances of CBL application (see Figure 2.13). This results in a feature map with dimensions $19x19x512$

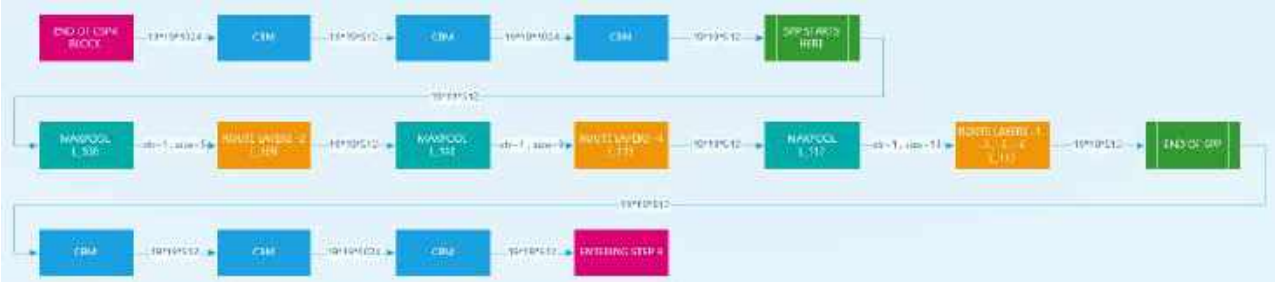

Figure 2.13 *Representation of the SPP block together with the six CBM layers*

- Step 9: A single CBL layer and upsampling operation are executed, resulting in an output dimension of $38x38x256$
- Step 10: Is a CBL layer with a feature map of $38x38x256$
- Step 11: Involves concatenation operation and produces an output dimension of $38x38x512$, while extracting data information from the feature maps of steps 9 and 10
- Step 12: Uses five CBL layers, resulting in an output dimension of $38x38x256$
- Step 13: A single CBL layer and upsampling operation are executed, resulting in an output dimension of $76x76x128$
- Step 14: Is a CBL layer with a feature map of $38x38x128$
- Step 15: Involves concatenation operation and produces an output dimension of $76x76x256$, while extracting data information from the feature maps of steps 13 and 14
- Step 16: Uses five CBL layers, which are performed before the detection of small objects
- Step 17: Uses a stride 2 operation and a CBL layer to generate an output dimension of $38x38x256$
- Step 18: Includes only concatenation operation, resulting in an output dimension of $38x38x512$
- Step 19: Uses five CBL layers, which are applied before the detection for medium objects
- Step 20: Uses a stride 2 operation and a CBL layer to generate an output dimension of $19x19x512$
- Step 21: Includes only concatenation operation, resulting in an output dimension of $19x19x1024$
- Step 22: Uses five CBL layers, which are performed before the detection for large objects.

<u>The detection heads component of the network</u>

The green box of Figure 2.10 contains three sets of feature maps with varying dimensions ($76x76$, $38x38$, and $19x19$), derived from steps 16, 19, and 22, respectively. That way the model can identify objects of diverse scales within the image.

In the original YOLOv4 algorithm, the number of filters before each detection head is set to 255. That is determined by the following equation:

$$Number\ of\ filters\ =\ (n+5)\ *\ 3 \tag{2.1}$$

In Equation 2.1, n represents the number of different classes on which YOLOv4 was trained. Since it was trained on the COCO dataset, which includes 80 distinct classes $n = 80$. 5 is a combined measure that comprises four anchor boxes and one objectness score, while "3" indicates the number of anchors integrated into the YOLOv4 model.

For instance, a single convolutional layer featuring 255 filters with dimensions $1x1$ and a stride of 1 is applied before the (large) detection head to reduce the output dimensionality from 512 to 255. The resulting $19x19x255$ feature map is used as input for detection and classification processes that locate large objects. Every grid cell produces three one-dimensional arrays, each representing one of the three anchors containing different features. These include box coordinates (b_x, b_y, b_h, b_w), the objectness score (p_0), and class probabilities (p_i) for each class (c). Thus, since in this case the training dataset contains 80 classes, the length of the individual one-dimensional array is 85 (comprising $b_x, b_y, b_h, b_w + p_o + 80$ class probabilities) for each anchor within every grid cell (Tepteris et al., 2023).

2.3.4 Bag of Freebies (BoF)

YOLOv4 introduced various techniques known as "Bag of Freebies" (BoF), which are included in the backbone and detector components of the network (Bochkovskiy et al., 2020). By utilizing BoF, the network requires less computational power during the training process, while increasing its overall accuracy. The majority of BoF focuses on data augmentation techniques, therefore the network is capable to generate new training instances from existing data. Allowing the model to experience a broader range of scenarios that might not otherwise be examined.

The techniques applied to the backbone and the detector components of the network are (Bochkovskiy et al., 2020):

- **For the backbone**:
  - ✓ **Data Augmentation Techniques:** CutMix (Yun et al., 2019a) and Mosaic data augmentation (Bochkovskiy et al., 2020)
  - ✓ **Regularization Techniques:** DropBlock regularization (Ghiasi et al., 2018) and Class label smoothing (Müller et al., 2020)
- **For the detector**:
  - ✓ **Loss Function and Optimization Techniques:** Complete Intersection over Union Loss (CIoU-Loss) (Zheng et al., 2019a), cosine annealing scheduler (Loshchilov and Hutter, 2017), Application of Optimal Hyperparameters (Bochkovskiy et al., 2020)
  - ✓ **Data Augmentation and Regularization Techniques:** Mosaic data augmentation, DropBlock regularization, Cross-mini Batch Normalization (CmBN) (Bochkovskiy et al., 2020), Self-Adversarial Training (SAT) (Zhao et al., 2022)
  - ✓ **Training Efficiency and Model Robustness Enhancements:** Cross-mini Batch Normalization (CmBN), Elimination of Grid Sensitivity (Bochkovskiy et al., 2020), Application of Multiple Anchors for a Single Ground Truth (Bochkovskiy et al., 2020)

In this thesis, we will only cover the BoF that was used to modify the configuration files before the execution of the experiments. The selected technique is:

- **For the backbone**:
  - ✓ **Data Augmentation Techniques:** Mosaic data augmentation
- **For the detector**:
  - ✓ **Data Augmentation and Regularization Techniques:** Mosaic data augmentation

and it will be described in Chapter 3.

2.3.5 Bag of Specials (BoS)

In addition to the "Bag of Freebies" (BoF), YOLOv4 introduces the "Bag of Specials" (BoS), which comprises a collection of more advanced techniques requiring more computational power. They ares are included in the backbone and the detector components of YOLOv4 network, similar to BoF (Bochkovskiy et al., 2020).

The techniques applied to the backbone and the detector components of the network are (Bochkovskiy et al., 2020):

- **For the backbone**:
  - ✓ **Activation function:** Mish activation function (Misra, 2020)
  - ✓ **Regularization Techniques:** Cross-Stage Partial connections (CSP) (Wang et al., 2019)
  - ✓ **Regularization and Optimization Techniques:** Multi-input Weighted Residual Connections (MiWRC) (Bochkovskiy et al., 2020)
  - ✓ **Training Efficiency and Model Robustness Enhancements:** Mish activation function, Cross-Stage Partial connections (CSP), Multi-input Weighted Residual Connections (MiWRC)
- **For the detector**:
  - ✓ **Activation function**: Mish activation function
  - ✓ **Optimization Techniques:** Distance-Intersection over Union - Non-Maximum Suppression (DIoU-NMS) (Zheng et al., 2019a)
  - ✓ **Training Efficiency and Model Robustness Enhancements:** Mish activation function, Spatial Pyramid Pooling (SPP) (He et al., 2014), Spatial Attention Module (SAM) (Woo et al., 2018), Path Aggregation Network (PANet) (Liu et al., 2018), Distance-Intersection over Union - Non-Maximum Suppression (DIoU-NMS)

In this thesis, like BoF techniques, we will only cover the BoS that were used to modify the configuration files before the execution of the experiments. The selected techniques are:

- **For the backbone**:
  - ✓ **Activation function:** Mish activation function
  - ✓ **Training Efficiency and Model Robustness Enhancements:** Mish activation function
- **For the detector**:
  - ✓ **Activation function**: Mish activation function
  - ✓ **Optimization Techniques:** Distance-Intersection over Union - Non-Maximum Suppression (DIoU-NMS)
  - ✓ **Training Efficiency and Model Robustness Enhancements:** Mish activation function, Distance-Intersection over Union - Non-Maximum Suppression (DIoU-NMS)

and they will be described in Chapter 3.

2.3.6 YOLOv4's loss function

To begin with, the loss function has multiple uses in the YOLOv4 model. It is not used only as an evaluation metric for training performance, but it also drives the optimization mechanism to adjust model parameters, such as weights (DataRobot, 2018). The loss function applied in the YOLOv4 model contains three different losses:

- **Regression loss:** The regression loss function (L_{CIoU}) is used to reduce the disparity existing between the predicted bounding box coordinates and those of the ground truth bounding box (Wu et al., 2021). Thereby, it is designed to improve the accuracy in object localization, and its mathematical representation is as follows:

$$L_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + au \tag{2.2}$$

where,

- b and b^{gt}: are the central coordinates of the boxes B and B^{gt}, respectively
- ρ: is the Euclidean distance between the centroids of the predicted and ground truth bounding boxes
- c: is the length of the diagonal line that covers both the predicted and ground truth bounding boxes when they are enclosed in the smallest possible shape
- α: can be adjusted to balance the importance of the distance between objects and their size differences when calculating CIoU.
- u: functions as a standardizing factor, accommodating the difference in aspect ratio between the predicted and ground truth bounding boxes
- IoU: is the Intersection over Union, as it is discussed in Appendix A.1.

The CIoU and IoU are further discussed in Appendix A.

- **Confidence loss:** The confidence loss ($L_{confidence}$) is used in the training process to improve the model's object detection performance. If the model produces poor results, it applies constraints to itself based on its confidence estimates about the presence of objects in different grid cells and its predictions of bounding boxes (Nguyen et al., 2022). The mathematical representation for this loss is as follows:

$$\begin{aligned} L_{confidence} = & \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i\right) \log(1 - C_i)\right] \\ & - \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i\right) \log(1 - C_i)\right] \end{aligned} \tag{2.3}$$

where,

- S^2: represents the total number of grid cells

- B: indicates the number of bounding boxes located per grid cell
- I_{ij}^{obj}: an indicator function representing the presence of an object within cell i associated with anchor j, taking the value 1 to indicate its presence and 0 otherwise
- I_{ij}^{noobj}: an indicator function representing the absence of an object within cell i and anchor j, assigning the value 1 to denote this absence and 0 otherwise
- C_i: represents the ground truth confidence score for the presence of an object in cell i
- $\hat{C}_i$: represents the predicted confidence score for the presence of an object in cell i
- $\log(C_i)$: computes the natural logarithm of the ground truth confidence score
- λ_{noobj}: provides the weighting term for cells without any object presence.

❖ **Classification loss:** The classification loss ($L_{classification}$) is used for training the object detection model (Nguyen et al., 2022). Its role involves applying limitations for incorrect classifications related to each bounding box, and its mathematical representation is as follows:

$$L_{classification} = \sum_{i=0}^{S^2} I_{ij}^{obj} \sum_{c \in classes} \left[\hat{p}_i(c) \log\left(p_i(c)\right) - \left(1 - \hat{p}_i(c)\right) \log\left(1 - p_i(c)\right)\right] \quad (2.4)$$

where,

- S^2: represents the total number of grid cells
- c: represents each individual class that the model is trained to detect
- $classes$: includes all categories of objects that the model is trained to identify
- I_{ij}^{obj}: an indicator function denoting the presence of an object within cell i associated with anchor j, taking the value 1 to indicate its presence and 0 otherwise
- $p_i(c)$: represents the true probability that object i belongs to class c
- $\hat{p}_i(c)$: represents the predicted probability that object i belongs to class c.

Consequently, the overall loss function of YOLOv4's model (T_L) is described as follows:

$$\begin{aligned} T_L = L_{CIoU} - L_{confidence} - L_{classification} \\ = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + au \\ - \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i\right) \log(1 - C_i)\right] \\ - \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i\right) \log(1 - C_i)\right] \\ - \sum_{i=0}^{S^2} I_{ij}^{obj} \sum_{c \in classes} \left[\hat{p}_i(c) \log\left(p_i(c)\right) - \left(1 - \hat{p}_i(c)\right) \log\left(1 - p_i(c)\right)\right] \end{aligned} \quad (2.5)$$

In Equation 2.5, S^2 is the configuration of S × S grid, where B candidate boxes within each grid are generated. This results in a total of $S\ x\ S\ x\ B$ bounding boxes. In instances where no object ($noobj$) is present within a box, the computation only includes the confidence loss associated with the box. This confidence loss function, which uses cross entropy error, is divided into two distinct components: one addressing the presence of an object (obj), and the other addressing its absence ($noobj$). The introduction of the weight coefficient λ within the loss function of $noobj$ is designed to reduce the contribution weight assigned to calculations concerning scenarios with no object. Furthermore, the classification loss function also uses cross entropy error. In scenarios where the j-th anchor box of the i-th grid is tasked with delineating specific ground truth, the classification loss function is applied to the bounding box generated by this anchor box (Wu et al., 2021).

2.4 Scaled-YOLOv4 models

The present Section overviews the evolution from YOLOv4 to Scaled-YOLOv4, which is designed to achieve an optimal balance between speed and accuracy. Specifically, the Scaled version of YOLOv4 introduces a fully optimized CSP referred to YOLOv4-P5, which is used as the foundational architecture for following evolutions to YOLOv4-P6 (see Figure 2.14 and 2.15) and YOLOv4-P7 (Wang et al., 2020).

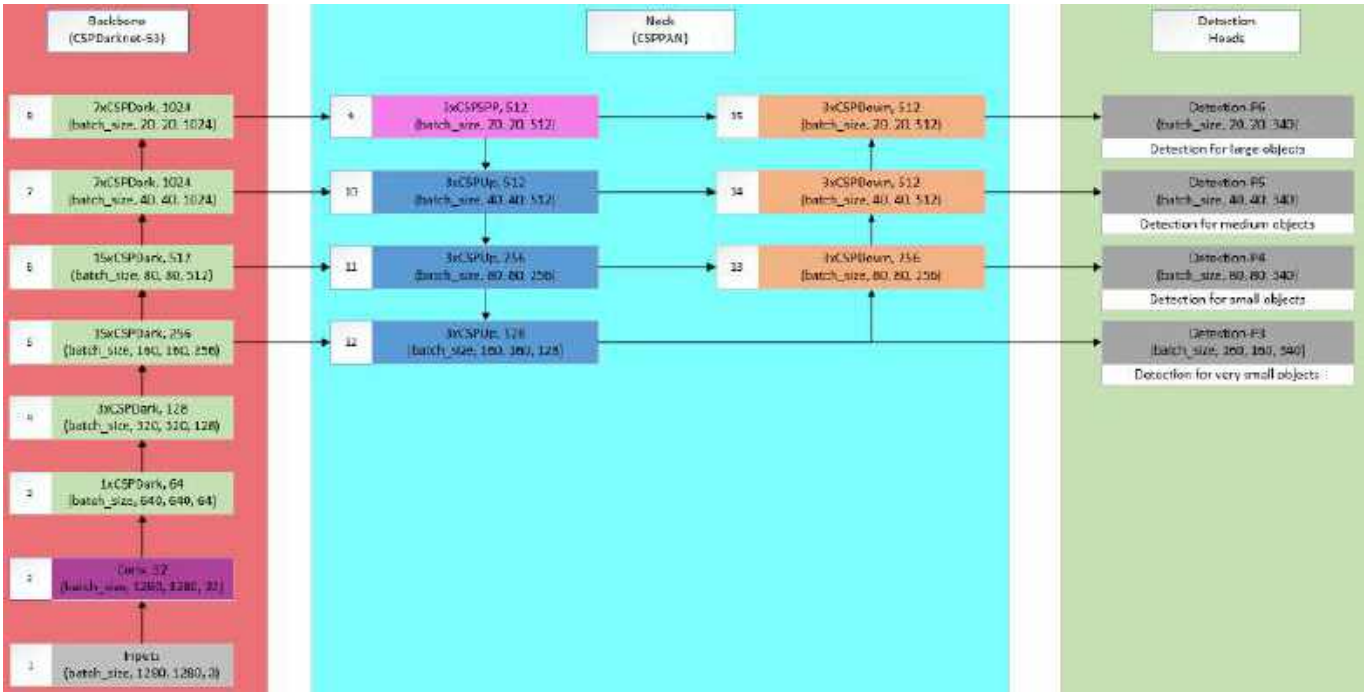

Figure 2.14 *YOLOv4-p6 neural network architecture (Wang et al., 2020)*

The backbone of the Scaled network optimizes both its size and the number of computational steps by applying compound scaling. This approach deals with adjusting both the size ($size^{input}$) of the input images and the number of steps (#stage) together (Wang et al., 2020). During the training process in each stage (CSPDark), the model increases the number of filters and the number of residual blocks used, but it decreases the size of the input images. Specifically, the size of the images is divided by two, and the number of filters is multiplied by two as it moves from one stage to another until it reaches the SPP

module. In addition, the number of residual blocks within each stage follows a specific pattern represented by the array [1, 3, 15, 15, 7, 7]. For instance, step 5 contains fifteen residual blocks inside its CSPDark block (see Figure 2.14).

However, the YOLOv4-p6 model limits the number of filters to 1024 to prevent issues such as overfitting and high computational demands. In steps 8 and 9, it uses 1024 filters, which is its maximum use.

Figure 2.15 represents the CSP block that the model uses to process the input image throughout the backbone component of the network. All CSP blocks have the same pattern, they begin with a $3x3$ convolution layer and produce a feature map with c channels. This feature map is then split into two paths, each with $\frac{c}{2}$ channels. The reason for reducing the channels to $\frac{c}{2}$ is to improve feature selection and computational efficiency by decreasing the computational load and memory use. The first path is further processed with a series of residual blocks. Each one of these includes an $1x1$ convolution, followed by a $3x3$ convolution and a shortcut layer. After the residual blocks, the resulting output feature map is processed through one more $1x1$ convolution. Meanwhile, the second path skips these additional convolutions, preserving the original split feature map. Afterwards, the two paths are concatenated again to create a combined feature map with c channels. Finally, this combined feature map is processed through an $1x1$ convolution to adjust the number of output channels by merging the information from both paths.

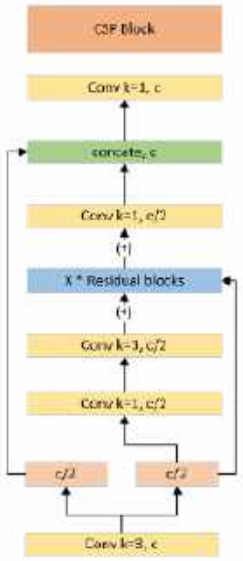

Figure 2.15 *Representation of the Scaled-YOLOv4 backbone's CSP block (Shaikh et al., 2023)*

The neck of the Scaled network functions similarly to YOLOv4. Specifically, it uses SPP to handle objects of different sizes and scales within the network and PANet to improve the object detection process by maintaining spatial details. Specifically, the model optimizes computational efficiency by incorporating the PANet architecture, which combines features obtained from various feature pyramids (see Figure 2.16). Figure 2.16 illustrates two configurations: (a) reversed dark layers (SPP) and (b) reversed CSP dark layers (SPP).

Both models use two different notations to describe their functionality, which are:

- k: is the filter size of the convolutional layer. For instance, $k = 1$ indicates that the convolutional layer uses a $1x1$ kernel, and $k = 3$ indicates that it uses a $3x3$ kernel
- b: is the number of filters in the convolutional layer.

The model (a) includes a series of convolutional layers with the following structure: $conv\ k = 1, \frac{b}{2}$, $conv\ k = 3, b$, and $conv\ k = 1, \frac{b}{2}$. The SPP is positioned in the center of the computation pipeline. Following the SPP module, the structure changes to $conv\ k = 3, b$ and $conv\ k = 1, \frac{b}{2}$.

The model (b) begins with a $conv\ k = 1, b$ layer, followed by a split into two paths $\left(\frac{b}{2}\right)$. The first path skips all convolutional layers and SPP to proceed to the concatenation operation, while the second path continues with $conv\ k = 3, \frac{b}{2}$ and $conv\ k = 1, \frac{b}{2}$ layers. Then, it passes through the SPP, which is positioned again in the center of the computation pipeline. Following the SPP, there is one more $conv\ k = 3, \frac{b}{2}$ before reaching the concatenation operation, where it merges the feature map of the first path with the second one.

After combining features from different feature pyramids (with the use of a route layer), the resulting feature map is $19x19x512$. This feature map then passes through a convolutional layer (with stride of 1 and filters size $1x1$), resulting in a $19x19x1024$ feature map. Subsequently, it proceeds through two series of reversed Darknet residual layers (before and after SPP), while avoiding shortcut connections. However, since the Scaled-YOLOv4 implements the CSP connection to its entire network, it leads into a modified neck component (see Figure 2.16). Following the successful integration of CSP, the modified module results in a 40% reduction of computational load (Wang et al., 2020).

As depicted in Figure 2.16, SPP was first added to the central position in the computation pipeline of the neck. Similarly, in the CSPPAN, the SPP module was positioned at the middle of the computation group pipeline (Wang et al., 2020).

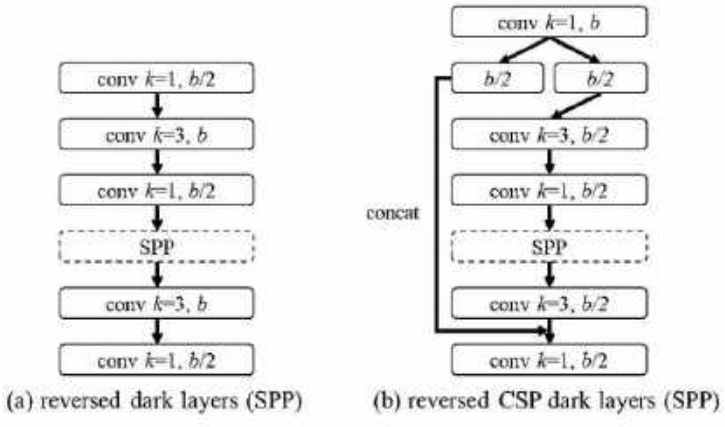

Figure 2.16 *Depicts the computational blocks of the reversed Dark layer (SPP) on the left and the reversed CSP dark layers (SPP) on the right (Wang et al., 2020)*

Experimental findings confirm that YOLOv4-P6 reaches real-time operational efficiency, achieving a video processing rate of 30 frames per second (fps) when the width scaling factor is set to 1. For YOLOv4-P7, real-time functionality is achieved at a video processing rate of 15 fps, depending on a width scaling factor of 1.25 (Wang et al., 2020).

2.5 Comparison between YOLO and other detection models

To demonstrate the progressive advancements in YOLO object detection algorithms, Table 2.1 offers a comparative analysis between the foundational YOLO models and the innovative Scaled-YOLOv4. This comparison not only shows the evolution of the YOLO family over time, but also highlights improvements in key areas of the model's design.

Table 2.1 *Comparison between YOLOv1, YOLOv2, YOLOv3 , YOLOv4, and Scaled-YOLOv4 models*

<table>
<tr><th>Differences in:</th><th>YOLOv1</th><th>YOLOv2</th><th>YOLOv3</th><th>YOLOv4</th><th>Scaled-YOLOv4</th></tr>
<tr><td rowspan="4">Architecture</td><td>Combines region proposal with object categorization in a single neural network forward pass</td><td>Uses higher resolution classifier (from 224x224 to 448x448)</td><td rowspan="4">Added residual blocks</td><td>Added BoF (Bag of Freebies) and BoS (Bag of Specials)</td><td rowspan="4">Added CSP (cross-stage partial) connections</td></tr>
<tr><td rowspan="3">Contains two fully connected layers at the end of the architecture</td><td>Fully connected layers were removed</td><td>Uses CIoU (Complete Intersection Over Union)</td></tr>
<tr><td rowspan="2">Batch Normalization was added</td><td>Utilizes an updated NMS: Greedy-NMS</td></tr>
<tr><td>Updated loss function</td></tr>
<tr><td>Backbone</td><td></td><td>Darknet-19</td><td>Darknet-53</td><td>CSPDarknet-53</td><td>Modified CSPDarknet-53 with stages</td></tr>
<tr><td>Neck</td><td>-</td><td>-</td><td>-</td><td>Added SPP and PANet</td><td>-</td></tr>
</table>

<table>
<tr><th>Differences in:</th><th>YOLOv1</th><th>YOLOv2</th><th>YOLOv3</th><th>YOLOv4</th><th>Scaled-YOLOv4</th></tr>
<tr><td rowspan="4">Head</td><td>One detection head</td><td>One detection head</td><td>Three detection heads were added to classify small, medium, and large objects respectively</td><td rowspan="4">Three detection heads</td><td rowspan="4">There are four detection heads in total, to classify very small, small, medium and large objects respectively</td></tr>
<tr><td rowspan="3">Used a grid of dimension SxS to extract the feature map from images</td><td>Utilization of anchor boxes</td><td>The accuracy increased for small objects but decreased for medium and large objects</td></tr>
<tr><td rowspan="2">Bounding box prediction: predicts location coordinates in relation to the grid cell's location.</td><td>Multi-label prediction was added</td></tr>
<tr><td>Objectness score was added</td></tr>
</table>

Table 2.1 compares the various versions of YOLO models. The initial version, YOLOv1, pioneered the integration of region proposal and object categorization within a single neural network forward pass. It used a grid of dimensions $S\ x\ S$ to extract feature maps from images, followed by two fully connected layers before the object detection stage. YOLOv2 introduced architectural modifications by replacing the fully connected layers with anchor boxes, incorporating Darknet-19 as the backbone and batch normalization in convolutional layers. Additionally, YOLOv2 allowed for training on higher resolution images (from 224x224 to 448x448) to improve the bounding box prediction accuracy.

YOLOv3 introduced further advancements with the adoption of Darknet-53 as the backbone. The model added residual blocks to its entire architecture and three detection heads at the end to identify small, medium, and large objects respectively. Additionally, multi-label prediction and objectness score were integrated. All these advancements led to improved accuracy for small objects, but with reduced performance for medium and large objects. Subsequently, YOLOv4 introduced significant updates such as Bag of Freebies (BoF) and Bag of Specials (BoS) techniques, refining its loss function and adopting Greedy-NMS for non-maximum suppression. Furthermore, it introduced a) switching to CSPDarknet-53 as its backbone and introducing) a neck structure after the backbone comprising Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PANet). Even if it extended its architectural depth, it retained the utility of three detection heads.

In the evolution from YOLOv4 to Scaled-YOLOv4, further improvements were introduced: Extension of the integration of CSP connections throughout the model's architecture, along with modifications in the backbone and neck components. The backbone is a modified version of CSPDarknet-53, with the number of layers varying depending on the Scaled YOLO model version used; YOLO-p5 (CSP-P5), YOLO-p6 (CSP-P6), and YOLO-p7 (CSP-P7) incorporate progressively more layers. The neck component incorporates CSP connections, creating CSPPAN. The CSPPAN has the same utility as PANet, but it is even better by incorporating CSP connections. Finally, the number of detection heads varied across Scaled YOLO versions, with YOLO-p5 featuring three heads, YOLO-p6 featuring four, and YOLO-p7 featuring five, respectively.

Table 2.2 compares different detection models based on the metric AP (Average Precision), which is detailed in Section 2.6. AP is a metric that evaluates the precision of an object detection model by averaging the precision values over a range of IoU thresholds, specifically from 0.5 to 0.95 in steps of 0.05. In contrast, AP50 assesses the precision at a single IoU threshold of 0.50, which indicates the model's accuracy when the overlap between predicted and ground truth bounding boxes is equal to or greater than 50%. While AP provides a complete perspective by considering various overlap levels, AP50 is less demanding and generally results in higher precision values due to the lower overlap requirement.

All models were trained on the COCO dataset to provide more accurate results. The objective of this table is to determine which detection model has the highest rate of object detection and classification.

Table 2.2 *Comparison between various detection models that are trained on coco dataset (Redmon and Farhadi, 2016a) (Redmon and Farhadi, 2018) (Bochkovskiy et al., 2020) (Wang et al., 2020)*

Model	Size	Backbone	AP	AP50
Faster R-CNN	-	-	21.9%	42.7%
SSD300	-	-	23.2%	41.2%
SSD512	-	-	26.8%	46.5%
YOLOv2	-	Darknet-19	21.6%	44%
YOLOv3	416x416	Darknet-53	31%	55.3%
YOLOv3	608x608	Darknet-53	33%	57.9%
YOLOv4	416x416	CSPDarknet-53	41.2%	62.8%
YOLOv4	512x512	CSPDarknet-53	43%	64.9%
YOLOv4-p5	896x896	CSP-P5	51.8%	70.3%
YOLOv4-p6	1280x1280	CSP-P6	54.5%	72.6%
YOLOv4-p7	1536x1536	CSP-P7	55.5%	73.4%

Table 2.2 illustrates that the highest result was produced from YOLOv4-p7, which belongs to Scaled-YOLOv4 models.

The analysis in Section 2.5 shows that the Scaled-YOLOv4 models have better Average Precision (AP) results. If we examine the development of the features introduced in each YOLO update, it is clear that

these improvements have been implemented with a reduction in training speed, as a compromise to improve prediction accuracy.

In this thesis we selected to use Scaled-YOLOv4 models, since they feature the best detection and classification results. From the three models that were included in the Scaled-YOLOv4 family, the YOLOv4-p6 algorithm was chosen. Despite a marginal superiority in AP demonstrated by YOLOv4-p7, it was avoided due to its slower training speed.

2.6 Analysis of performance metrics

Performance metrics are used to evaluate the detection performance of a model. Important metrics include precision, recall, F1-score, Average Precision (AP) and mean Average Precision (mAP). All are based on the model's classification and detection results, that are identified as True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN). The focus of this Section is to provide an overview of these metrics, which are significant for all object detection models.

True positives (TP)/ False positives (FP)/ False Negatives (FN)/ True Negatives (TN)

To describe these four concepts (Google, 2022), we assumed the $IoU_{threshold}$ to be 0.5 (see Table 2.3). As shown in Appendix A.1, IoU indicates the overlap between the predicted bounding box and the ground truth bounding box.

- True positives (TP) occur when a) $IoU \geq IoU_{threshold}$ meaning that the predicted bounding box overlapped beyond the threshold (or even matched) the ground truth bounding box, and b) the object has been classified correctly. As a result, a true positive result indicates that the predicted bounding box correctly predicted the location and class of an object within a ground truth bounding box.
- False positives (FP) can occur in four different scenarios regarding to the predicted bounding boxes:
    - When $IoU \geq IoU_{threshold}$, but the object has been misclassified (classified in the wrong class)
    - When $IoU < IoU_{threshold}$, that is, the predicted bounding box mislocated the object
    - If there are multiple predicted bounding boxes with $IoU \geq IoU_{threshold}$ (say IoU = 0.88, 0.71, and 0.75). Only the one with the highest IoU (e.g. 0.88) is considered a true positive. The remaining predicted bounding boxes are classified as false positives
    - When a predicted bounding box appeared without a corresponding ground truth bounding box with the respective class.

  As a result, a false positive result indicates that the predicted bounding box did not correctly predict the class or location of an object within a ground truth bounding box.
- False negatives (FN) can occur in three different scenarios regarding to the ground truth bounding boxes:
    - Similar to the scenario of FP, when $IoU \geq IoU_{threshold}$, the model has missed to correctly predict the class of the object of the ground truth bounding box. For instance, in Figure 2.17, example (c) represents a pair of FP and FN results. Both were generated because the

predicted bounding box misclassified the class of the object of the ground truth bounding box, resulting in an unsuccessful detection. FP comes from the fact that the model predicted a small vehicle which is not the predicted object (person). FN comes from the fact that the model did not predict the person (negative), although it exists).

- Similar to the scenario of FP, when $IoU \leq IoU_{threshold}$, the model has failed to successfully detect the location of the object of the ground truth bounding box. For instance, in Figure 2.17, example (c) represents a pair of FP and FN results. Both were generated because the object of the predicted bounding box was mislocated due to insufficient overlap between the predicted and ground truth boxes, resulting in an unsuccessful detection, even if the model correctly predicted the class of the ground truth object. FP comes from the fact that the model predicted a person which does not exist at the predicted location. FN comes from the fact that the model did not predict the object that exists in the location of the ground truth box.
- When a an object and its ground truth bounding box are present, but the model does not predict a bounding box at all in the said location. This means that there was an object, but the model did not detect it

As a result, a false negative indicates that the object included in the ground truth bounding box was not predicted by the predicted bounding box.

- True negatives (TN) are the cases in which the model correctly identifies the absence of an object.

Table 2.3 *Representation of TP/FP/FN/TN*

	An object exists in reality?	
An object was identified by the model?	**Yes**	**No**
Yes	TP (correct class and $IoU \geq IoU_{threshold}$) FP (wrong class or $IoU < IoU_{threshold}$)	FP
No	FN	TN

Figure 2.17 depicts three different examples using Intersection over Union (IoU) to evaluate the accuracy of predicted bounding boxes compared to ground truth bounding boxes. The green box represents the ground truth bounding box, and the purple box represents the predicted bounding box. In the following examples, the $IoU_{threshold}$ has been set to 0.5 (50%).

In Example (a), the IoU is 0.84, which is greater than the $IoU_{threshold}$. This means that the predicted bounding box overlapped the ground truth box sufficiently, and it also classified the class “person” correctly. As a result, Example (a) is a true positive (TP).

In Example (b), the IoU of the two boxes is 0.74, which is greater than the $IoU_{threshold}$. This means that the predicted bounding box overlapped the ground truth box sufficiently. However, the model misclassified the object as a "small vehicle" instead of a "person”. Consequently, this misclassification results in a false positive (FP), since the object was incorrectly identified (purple arrow). Due to this misclassification, the model did not correctly predict the "person" present in the ground truth bounding

box. Therefore, it also results in a false negative (FN) because the model did not correctly detect the actual object (a "person") present in the ground truth bounding box (green arrow).

In Example (c), the IoU of the two boxes is 0.33, which is less than the $IoU_{threshold}$. This means that the predicted bounding box did not overlap sufficiently with the ground truth bounding box. Although the classification of the object (in the two boxes) is correct the result is considered a false positive (purple arrow), because the $IoU < IoU_{threshold}$ $(0{,}33 < 0.5)$. Due to the insufficient overlap (low IoU), the model's prediction (predicted bounding box) does not sufficiently match the ground truth bounding box. Therefore, this scenario also results in a false negative (green arrow) because the ground truth object is not sufficiently detected within the predicted bounding box according to the $IoU_{threshold}$.

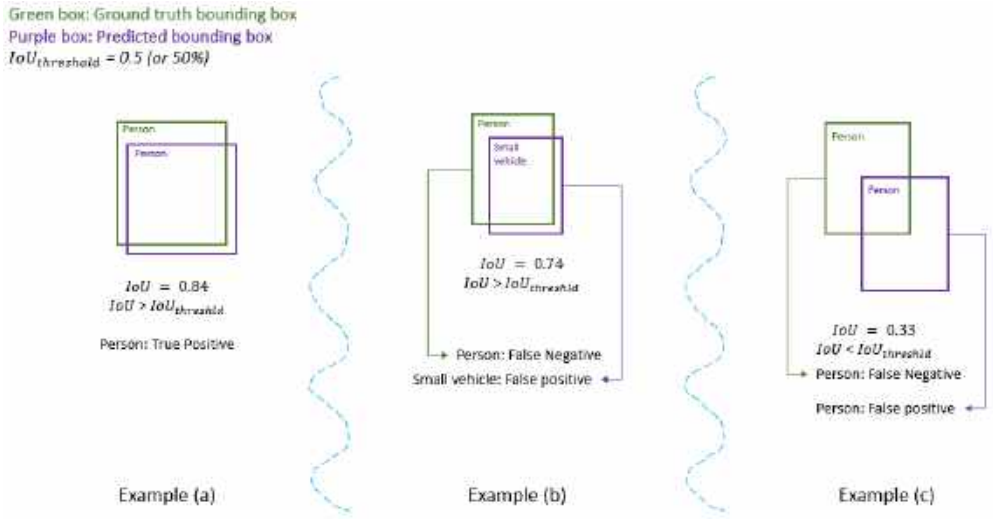

Figure 2.17 *Examples of True Positives, False Negatives, and False Positives*

Figure 2.18 illustrates the two different negatives that can occur from the use of bounding boxes. These are false negatives and false positives. On the left, the green box represents a ground truth bounding box where a person is present, but there is not a corresponding predicted bounding box (purple). This results in a false negative because an object was not detected. On the right, a purple predicted box appears without a corresponding green ground truth box. This results in a false positive because a non-existent object was predicted.

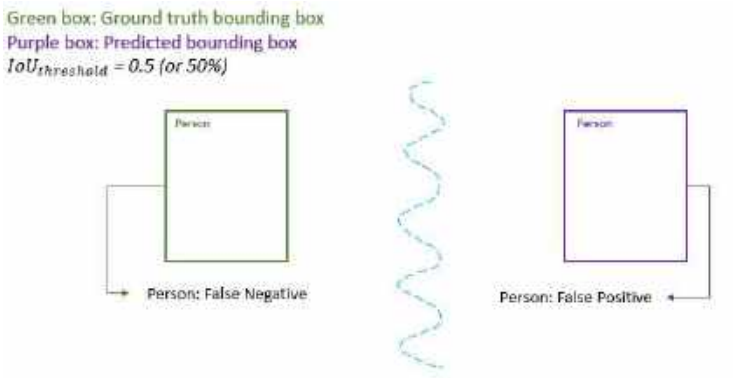

Figure 2.18 *Examples of False Negative and False Positive*

In Figure 2.19, there are two predicted bounding boxes, each overlapping the ground truth bounding box. The first predicted bounding box has an IoU of 0.76, which exceeds the $IoU_{threshold}$. Similarly, the second predicted bounding box has an IoU of 0.89, also surpassing the $IoU_{threshold}$. Both predicted bounding boxes correctly classify the object as "person". However, the second predicted bounding box has higher IoU than the first ($IoU_{threshold} \leq 0.76 \leq 0.89$). Consequently, the prediction with the highest IoU is true positive, while the other is false positive.

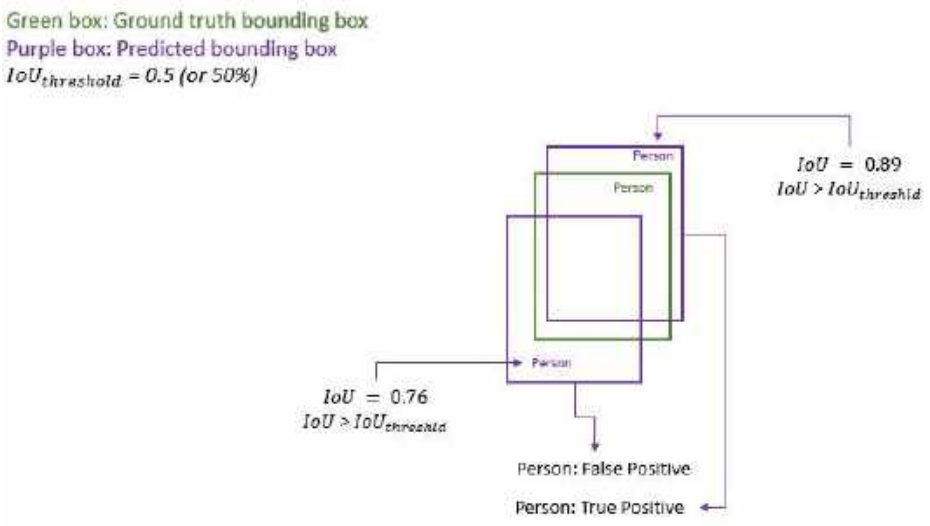

Figure 2.19 *Illustration of multiple bounding box predictions for a single object*

Precision

The precision metric evaluates the accuracy of the model when identifying objects. It is calculated by dividing the number of correctly classified objects (true positives) by the total number of instances identified as positives (true positives and false positives) (Vakili et al., 2020).

Thus, Precision is provided by:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.6}$$

Increased precision is observed in two circumstances (Gad, 2020):

- When the model generates many accurate positive classifications, thus maximizing true positives
- When the model minimizes the frequency of incorrect positive classifications, thus reducing the number of false positives.

Recall

The recall metric evaluates the ability of the detector to locate the objects within the image. It is calculated as the number of true positive instances (correctly identified objects) divided by the total number of object instances (true positives and false negatives) (Vakili et al., 2020).

Recall is mathematically represented as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.7}$$

High recall shows that the model finds most of the objects, thus avoiding missing objects in the image. In contrast, reduced recall indicates that the model is missing a number of objects, which can lead to inaccurate object detection and inferior performance in certain applications (iguazio, 2022).

F1-score

Precision and recall measure quite different model abilities. For example, assume an image with one object that is identified correctly by the model (in terms of both class and bounding box). Then recall = 1. However, precision may be 0.1 if the model has identified 10 objects in the image (one TP and nine FP). The F-1 metric attempts to combine these two measures to one that assesses the overall performance of the model (Vakili et al., 2020). It is represented as the harmonic mean of precision and recall:

$$\text{F1} - \text{score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.8}$$

The result from Equation 2.8 ranges from 0 to 1 (0%-100%) and represents the balance between precision and recall (Kundu, 2022).

Average Precision (AP)

Consider the relationship between recall and precision across many images. Specifically, for any recall value r (i.e. the ability of the model to identify objects within an image), consider the corresponding precision value $p(r)$ (how many of the positives identified are true positives).

Average Precision (AP) is the mean precision across the recall range that is generated by the model across these multiple images; that is (Anwar, 2022):

$$AP_i = \int_{r=0}^{1} p(r)dr \tag{2.9}$$

where,

- AP_i: represents the Average Precision calculated for each class i
- $p(r)$: represents the precision-recall curve across multiple images
- r: represents the recall values ranging from 0 to 1.

In Section 2.5, Table 2.2 displays the results in AP and AP50. The difference between them is that AP50 measures the average precision at a single IoU threshold of 0.5 (50%). This metric represents the model's accuracy when the overlap between the predicted and ground truth bounding boxes equals or exceeds 50%.

As an example, let's create a PR curve for one class using the data provided below:

Table 2.4 *Experimental data on recall and precision*

Images	Recall	Precision
1	20%*	100%**
2	30%	100%
3	30%*	67%**
4	50%	75%
5	50%	62%
6	70%	66%
7	70%	57%
8	70%	52%
9	90%	49%
10	100%	53%

* In the first example 20% of the objects in the image were identified correctly, while in the second one 30% of the objects were identified correctly

** In the first example, all the identified objects were true, and in the second only 2/3 of the identified objects were true.

Based on Table 2.4, we created the PR curve shown below:

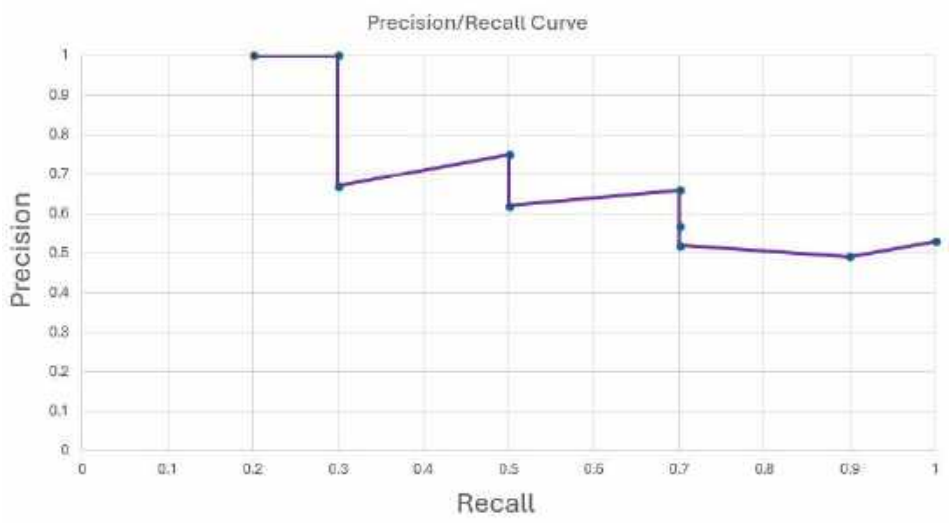

Figure 2.20 *The Precision/Recall curve from the data of Table 2.4*

To evaluate AP we may use the method from the 2007 PASCAL VOC challenge (Everingham et al., 2010), which is also used in the YOLOv4 and YOLOv4-p6 models in the validation and testing processes. This method takes Precision values at 11 equally distributed Recall points: 0, 0.1, 0.2, 0.3, ..., 1.0. For each Recall value, Precision is interpolated as the highest Precision observed at any higher Recall value ($p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$). In other words, it is the maximum Precision value to the right. Based on this method, Average Precision is computed as:

$$P = \frac{1}{11} \sum_{\substack{i=0 \\ step=0.1}}^{1} Precision(Recall) \tag{2.10}$$

Consequently, based on the 11-point interpolation method, we created the following plot:

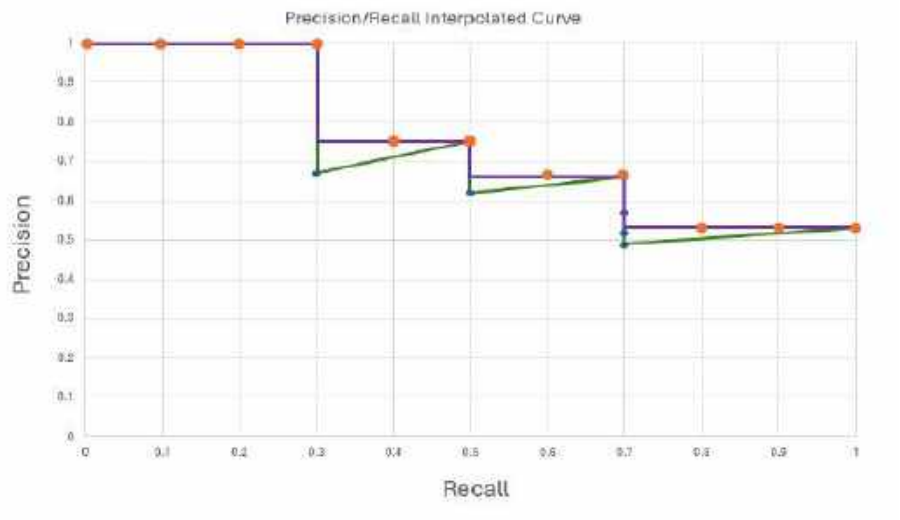

Figure 2.21 *Represents the Precision/Recall interpolated curve*

From Figure 2.21, the AP is calculated as follows:

$$\begin{aligned} AP = \frac{1}{11} \sum_{\substack{i=0 \\ step=0.1}}^{1} Precision(Recall) &= \frac{1}{11}\left(AP_r(0) + AP_r(0.1) + AP_r(0.2) + \ldots + AP_r(1)\right) \\ &= \frac{1}{11}\left((4 * 1) + (2 * 0.75) + (2 * 0.66) + (3 * 0.53)\right) = \frac{8.41}{11} = 0.7645 \\ &= 76{,}45\% \end{aligned} \tag{2.11}$$

The reason for interpolating the PR curve in the above way is to reduce the effect of small fluctuations in the PR curve, which are caused by small variations in the ranking of examples (Hui, 2019). This ensures that AP is not too sensitive to small variations in precision at lower recall values.

mean Average Precision (mAP)

The mean Average Precision (mAP) metric summarizes the Average Precision (AP) of each individual class and then divides the total AP value by the sum of the classes. It is mathematically represented as (Henderson and Ferrari, 2017):

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{2.12}$$

where,

- AP_i: represents the Average Precision calculated for each class
- N: represents the total number of classes.

In the fourth Chapter of this study, we used the mean average precision (mAP) metric to evaluate the performance of our modified YOLOv4-p6 algorithm. This metric can be used to evaluate the model's performance at different stages of training to identify optimal breakpoints to reduce computational resources without reducing the quality of the results. Another reason for choosing mAP is that it can be used for models trained with multiple classes, addressing class imbalances (the distribution between its classes is uneven) and detection difficulties to ensure unbiased performance across all classes (Shah, 2022). As a result, many detection models like YOLO algorithms are evaluated with the mAP metric, since they use datasets with many classes such as the COCO dataset (COCO, 2017), and because of the metric's robustness and extensive capabilities to optimize detection models.

Chapter 3 Data preparation and parameter selection for training the YOLOv4-p6 algorithm

In this third Chapter we present key aspects that are leveraged to train the YOLOv4-p6 model in detecting objects within recorded images by a UAV. The method focuses on two important training aspects:

- The selection of the training datasets
- Selection of training hyperparameters and the way to adjust them in order to optimize the efficiency of the trained model as well as the model's detection and classification performance.

In this Chapter we also present the experimental setup (hardware and software) used to conduct the training process.

3.1 Training data selection

The effectiveness of object detection and classification model is directly related to the amount and quality of data used during training. Specifically, effective training requires the application of large and high quality datasets to reduce errors, including those that are due to overfitting and bias.

Numerous resources, including datasets from cloud repositories, web platforms, universities and research institutions, offer annotated image collections that can be downloaded and used for research purposes in computer vision tasks. While certain datasets are freely accessible, others require payment or subscription for use.

- **Cloud repositories** are online platforms dedicated to storing and sharing datasets, with illustrative instances including:
  - GitHub (GitHub, 2008)
  - GitLab (GitLab, 2011)
- **Web platforms** provide tools for searching, browsing, and downloading datasets, with illustrative instances including:
  - MS COCO (COCO, 2017)
  - Kaggle (Kaggle, 2010)
- **Universities and research institutions** make available image datasets to assist others with their research. Examples of such datasets include:
  - Stanford University (Stanford University Computational Vision and Geometry Lab, 2009)
  - Massachusetts Institute of Technology (MIT) (MIT Lincoln Laboratory 1951, 1998)

Datasets relevant to the current research are those that contain UAV recorded images that contain at least four object classes:

- Person
- Small vehicle
- Large vehicle
- Ship.

Those object classes are related to the identification of unauthorized intrusion activities within the secure perimeter of logistics facilities, such as ports and logistics centers. The selection of datasets should be based on both quantitative and qualitative characteristics, including:

- **Quantitative characteristics:**
    - Large quantity of data for the purpose of training the object detector
    - Representation of images captured from various perspectives and heights (e.g. differing dimensions of large vehicles like buses, commercial vehicles, etc)
- **Qualitative characteristics:**
    - Inclusion of multicolored images depicting objects targeted for detection, exhibiting a diverse range of colors
    - The presence of numerous objects depicted in the images
    - Variation in lighting conditions across images, including different times of the day (e.g. morning, afternoon, and evening).
    - Capturing images affected by noise disturbances (e.g., images featuring rain or fog).
    - Variations in both the object's spatial placement within the environment and the characteristics of the environment itself.

Considering the four required classes, and the above quantitative and qualitative characteristics, we selected the following UAV datasets for training the YOLOv4-p6 model:

- The **aerial vehicle dataset** classifies the objects within the images into five distinct classes: car, truck, bus, minibus, and cyclist. It contains 134 annotated images captured in both urban and rural environments, featuring varied image resolutions spanning from 684x547 to 5,820x8,784 pixels (Kharuzhy, 2018).
- **DOTA (v1.5 & v2.0 combined) dataset**:
    - The **DOTAv1.5 dataset** comprises a training set and a validation set containing 1,869 annotated images captured across various urban and rural environments, featuring resolutions ranging from 353x851 to 13,383x4,287 pixels. The objects contained in the images of the training and validation sets are classified into sixteen distinct classes: plane, ship, storage tank, baseball diamond, tennis court, basketball court, ground track field, harbor, bridge, large vehicle, small vehicle, helicopter, roundabout, soccer ball field, swimming pool and container crane (Xia et al., 2021).
    - The **DOTAv2.0 dataset** comprises a training set and validation set containing 548 annotated images captured across various urban and rural environments, featuring resolutions ranging from 1,024x1,024, to 7,360x4,912 pixels. The objects contained in the images are classified into nineteen distinct classes: plane, ship, storage tank, baseball diamond, tennis court, basketball court, ground track field, harbor, bridge, large vehicle,

small vehicle, helicopter, roundabout, soccer ball field, swimming pool, container crane, airport, and helipad (Xia et al., 2021).

- The **VisDrone-DET dataset**[1] includes a total of 10,209 annotated images. Within the scope of this thesis, only the training and validation sets are used, which comprise 8,629 annotated images acquired from urban and rural landscapes. These images feature diverse resolutions ranging from 480x360 to 2,000x1,500 pixels. They contain ten distinct classes, namely pedestrian, persons, bicycle, car, van, truck, tricycle, awning-tricycle, bus, and motor (Zhu et al., 2021).
- The **Stanford drone dataset** includes images that contain six classes of objects: pedestrian, biker, skater, cart, car, bus. Overall, it contains 7,486 annotated images, featuring varied image resolutions spanning from 1,184x1,759, to 1,983x1,088 pixels (Robicquet et al., 2016).
- From the entire DJI-DAC dataset this thesis uses only a portion of the **DAC-SDC dataset**, which includes only three classes: car, ship, and person. Additionally, it contains 58,186 annotated images from urban and rural areas with an image resolution of 640x360 pixels (Xu et al., 2018).

The above datasets fulfill the quantitative and qualitative characteristics and possess a considerable volume of data for identifying the selected classes, as represented in Section 3.2.

3.2 Annotation adjustments in UAV datasets

In order to merge the datasets described above into a single, all inclusive dataset, the annotations of each individual dataset were adjusted. This is because each dataset contains different classes and follows different annotation formats. Hence, the process requires standardizing the annotation format to match YOLO specifications (see Appendix D.1) and aligning the label names across all datasets. This ensures consistency in both the structure of the annotations and the labelling of the classes.

The end objective is to provide a combined dataset that contains four different classes, namely “person”, “small vehicle”, “large vehicle”, and “ship”. Specifically, "person" corresponds to class zero, "small vehicle" to class one, "large vehicle" to class two, and "ship" to class three, with this information recorded in a text file and subsequently stored within the directory designated as “darknet”. Appendix D.2 provides a detailed analysis of the modifications made to each dataset.

Table 3.1 and Figure 3.1 provide the number of images and the data content in terms of the four classes discussed above.

Table 3.1 *The labelled objects per dataset*

		Number of objects			
Datasets	**Number of images**	**Number of persons**	**Number of small vehicles**	**Number of large vehicles**	**Number of ships**
Aerial cars	154	-	3,787	238	-
DOTA (v1.5 & v2.0)	2,417	-	219,328	29,872	51,860
VisDrone-DET	8,629	147,747	219,707	25,401	-

[1] (VisDrone, 2023)

Datasets	Number of images	Number of objects			
		Number of persons	Number of small vehicles	Number of large vehicles	Number of ships
Stanford drone	7,486	93,020	26,332	910	-
DAC-SDC	58,186	27,965	25,014	-	5,207

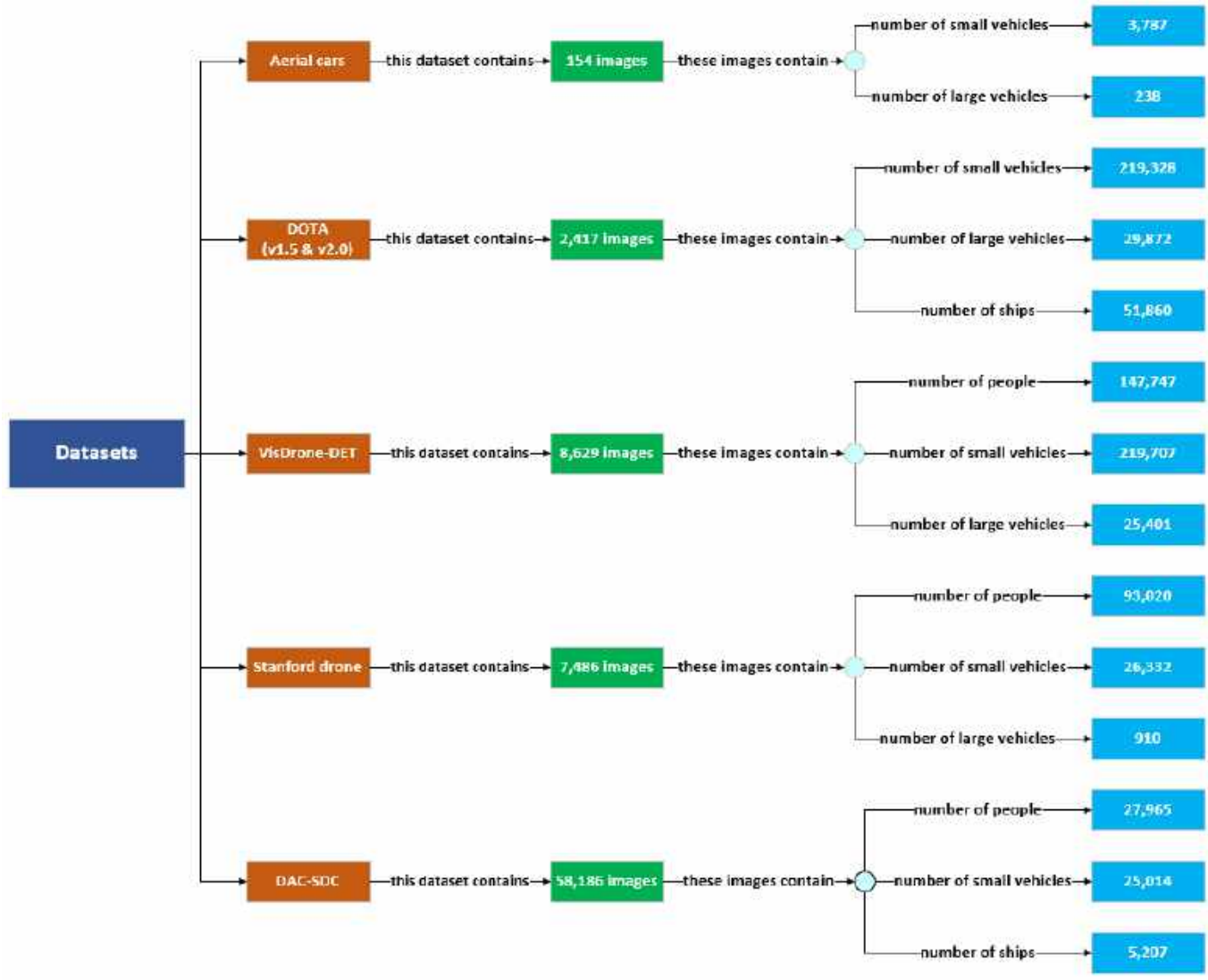

Figure 3.1 *The number of labelled objects present within the datasets*

3.3 Training, validation and testing datasets

After creating a single dataset, we created three different subsets as follows (see Figure 3.2):

- The **training set**, consisting 80% of the images in the integrates set, is used for the initial training of the model. Within this set, there are 61,429 images including 214,905 annotations labeled as

'person', 361,501 annotations labeled as “small vehicle”, 45,073 annotations labeled as “large vehicle”, and 44,027 annotations labeled as “ship”

- During the training process, the model’s performance is evaluated using the **validation set**, consisting of 10% of the original annotated set. Within this set, there are 7,678 images including 26,615 annotations labeled as “person”, 67,350 annotations labeled as “small vehicle”, 5,665 annotations labeled as “large vehicle”, and 6,368 annotations labeled as “ship”
- The **testing set**, consisting of 10% of the original annotated set, is designed to provide an evaluation of the model's performance. Within this set, there are 7,680 images including 27,212 annotations labeled as “person”, 62,366 annotations labeled as “small vehicle”, 5,580 annotations labeled as “large vehicle”, and 6,672 annotations labeled as “ship”.

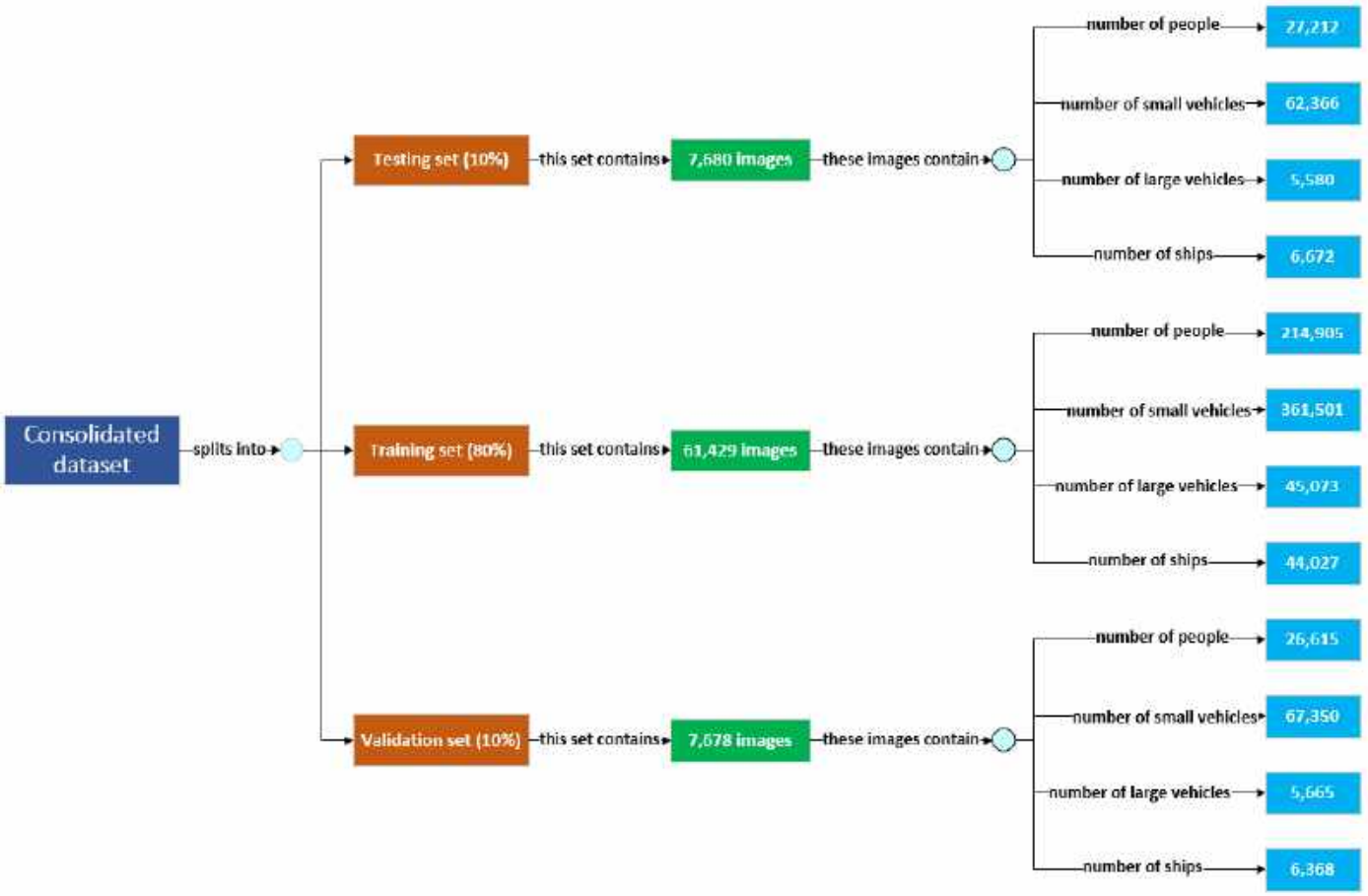

Figure 3.2 *The percentage distribution of data into training, validation, and testing sets*

3.4: Experimental setup

Training of the YOLOv4-p6 model requires substantial computational resources to achieve effective results. Key characteristics of the system used to perform this study are presented in Table 3.2:

Table 3.2 *The hardware configuration of the system*

Hardware	
Central Processing Unit (CPU)	AMD Ryzen 9 3900X 12-core Processor
Graphical Processing Unit (GPU)	Nvidia GeForce RTX 3090/PCIe/SSE2
Graphic Card Memory (GCM)	24GB
Random Access Memory (RAM)	32GB
Hard Disk Drive (HDD)	ATA TOSHIBA HDWD240 4TB
Solid State Drive (SSD)	ADATA SX8200PNP 512 GB

The Nvidia GeForce RTX 3090/PCIe/SSE2 graphics card is ideal for deep learning tasks and has impressive specifications such as 328 tensor cores, 10496 CUDA cores, and 24GB of GDDR6X memory.

The system operates on Ubuntu 20.04 LTS with the installed programs of Table 3.3

Table 3.3 *The software components of the system*

Software	
Operating System (OS)	Ubuntu 20.04 LTS
System type	x64-based
OpenCV	4.5.4 Version
CUDA Drivers	11.4 Version
cuDNN	8.3 Version

OpenCV (Open-Source Computer Vision Library) is implemented and used in the YOLOv4 and YOLOv4-p6 algorithms for real-time object detection and classification. OpenCV aids in preprocessing input images, ensuring they are correctly formatted and optimized for the selected YOLO model. Additionally, OpenCV's DNN (Deep Neural Network) module supports the loading and execution of YOLO models.

The CUDA and cuDNN drivers are installed to maximize the utilization of the RTX 3090 graphics card. CUDA functions as a parallel computing framework that enables developers to harness the power of Nvidia GPUs to improve the efficiency of computationally demanding tasks. cuDNN functions as a specialized library containing developed deep neural network protocols designed for efficient use with Nvidia GPUs.

3.5: Important hyperparameters for YOLOv4-p6 model training

The objective of this research is to train the model in the most effective way in order to provide near optimal mean Average Precision (mAP) results. Model training depends on several hyperparameters that affect the efficiency of the training process and the effectiveness of the trained model. There are two different types of hyperparameters: The ones that are set based on the characteristics of the training dataset, and the ones the “trainer” selects to optimize training performance of the selected model.

The first set of hyperparameters are presented in Table 3.4.

Table 3.4 *Hyperparameters that are set based on the characteristics of the training dataset*

First set of hyperparameters
The number of classes
Number of max batches
Number of steps (during which the learning rate is kept constant)
Number of filters of the convolutional layer before each detection head

Subsection 3.5.1, presents the description of each hyperparameter of the first set and the values used during our study based on the UAV dataset characteristics. The correct configuration of these hyperparameters is essential to improve the mAP results of the model.

The second set of hyperparameters concerns those that directly influence the model's architecture and operation (see Table 3.5). By fine-tuning these hyperparameters, the user can intentionally modify the model's interpretation and processing of the input data (images or video frames), resulting in better (or worse) detection and classification results. Unlike the first set of hyperparameters, the adjustments of which are based on the characteristics of the UAV dataset, the second set requires thorough testing of various hyperparameters and their combinations. The process of experimenting with different combinations of hyperparameters is essential for achieving optimal configuration for training.

Table 3.5 lists all the candidates for the second set of hyperparameters, and those selected for the tuning of our experiments.

Table 3.5 *Hyperparameters that influence the model's architecture and operation*

Candidates for the second set of hyperparameters	Selected for tuning
Box loss	
Image resize	✓
Anchor box dimensions	✓
Network depth	
Num anchors	
Num heads	
NMS (Non-Maximum Suppression)	✓
Learning rate	
Freeze layer	
Batch size	
Stride size	
Data augmentation techniques	✓
Activation function in YOLO layers	
Activation function in structure	✓

The reasons for choosing the hyperparameters of Table 3.5 are outlined below:

- Image resize determines the input size of images and directly affects both training and inference processes, as well as detection accuracy. When a model is trained with high resolution input images, it captures more details (corners, shapes, etc.) from the data, leading to the extraction of more accurate information that can improve the detection and classification results. However, high image resolution requires more computational resources and slows down the training process.
- Anchor box dimensions refine the predetermined anchor boxes to generate better bounding boxes, which affects the accuracy of localizing and detecting objects of different scales and aspect ratios
- NMS ensures that only the most suitable bounding boxes are retained, thus improving the model's predictions
- Data augmentation techniques enhance the model's robustness by increasing the diversity and quantity of training data
- Activation function determines the model's non-linear behavior, impacting its capacity to learn complex features from the input data. In YOLOv4-p6, the Mish activation function is used in the backbone and neck parts of the architecture, while the Sigmoid activation function is used in the convolutional layer (included in neck) before each detection head.

Conversely, the remaining hyperparameters were not selected to be adjusted, for the reasons listed below. In general, however, we had to limit the number of parameters to be tested in this work in order to limit the number of parameter combinations (and, thus, save computational effort) without compromising the point of this research.

- Box loss is used for optimizing bounding box predictions during training. YOLOv4 and Scaled YOLOv4 apply CIoU loss, which has been found to be superior to other loss mechanisms, such as DIoU or GIoU in many cases – this is why we used CIoU and did not test the other IoU approaches
- As mentioned in Chapter 2, Scaled YOLOv4 models include three different versions known as YOLOv4-P5, YOLOv4-P6, and YOLOv4-P7. Each version has a different number of layers to support their respective number of detection heads, with YOLOv4-P5 having three heads, YOLOv4-P6 having four heads, and YOLOv4-P7 having five heads. YOLOv4-P6 and YOLOv4-P7 showed similar and better mAP results than YOLOv4-P5, but YOLOv4-P7 required more training time compared to the other two versions. Creating a new network with six or seven detection heads requires more layers, which would further worsen the problem of the excessive training time. As a result, we chose the YOLOv4-P6 architecture because it provides performance results close to YOLOv4-P7, but with faster training time. This model features a total of 304 layers (network depth) configured to support four detection heads, with each head using four anchors to provide optimal performance.
- The learning rate affects the speed at which the model parameters converge during training. YOLOv4-p6 performed effectively with its default learning rate setting throughout training, so we

did not change it to prevent premature convergence to a local optimum or slowing down the training process and preventing the model from converging (Zulkifli, 2018)

- Batch size was included in the priority (for adjustment) hyperparameters, since decreasing it can speed up the training process but may reduce the performance of the model. However, increasing the batch size together with the learning rate can produce positive results (Devansh, 2023). Since the learning rate remains unchanged and YOLOv4-p6 specifies a batch size of 64, we used this configuration for our experiments. Regarding stride size, the YOLO developers did not provide any adjustments for smaller objects in YOLOv4-p6, since of the existing four heads in the model's architecture two are designed to detect very small and small objects. Moreover, the convolutional layers preceding each detection head use the sigmoid activation function, which is compatible for UAV tasks. Therefore no modifications were made to these layers (Shatravin et al., 2022)
- The freeze layers hyperparameter was tested in a couple of experiments. It turned out to be functional only in YOLOv4-p5, while in YOLOv4-p6 and YOLOv4-p7, the models either failed to generate any mAP results or produced poor mAP results.

Lastly, we have ensured that the selected hyperparameters are present in all components of YOLOv4-p6's architecture, from the backbone and neck to the detection heads. We even considered including and adjusting hyperparameters at the net section of the YOLOv4-p6 configuration file, which influence the input data either at the beginning or during the training process.

3.5.1. Hyperparameters defined based on the characteristics of the dataset

In the default YOLOv4-p6 model the hyperparameters belonging to the first set (see Table 3.4) were set based on the COCO dataset, which the model's developers used for training. In our case we adjusted these hyperparameters to optimize training efficiency and tailor the model to match the characteristics of our modified UAV dataset.

The number of classes

The number of classes represents the variety of different object categories on which the model is trained for detection tasks. In the conducted experiments, four different classes are sought, i.e. person, small vehicle, large vehicle, and ship. Consequently, the "classes" hyperparameter of the original configuration file is adjusted from 80 to 4 classes, as depicted in Figure 3.3 (Patel et al., 2021).

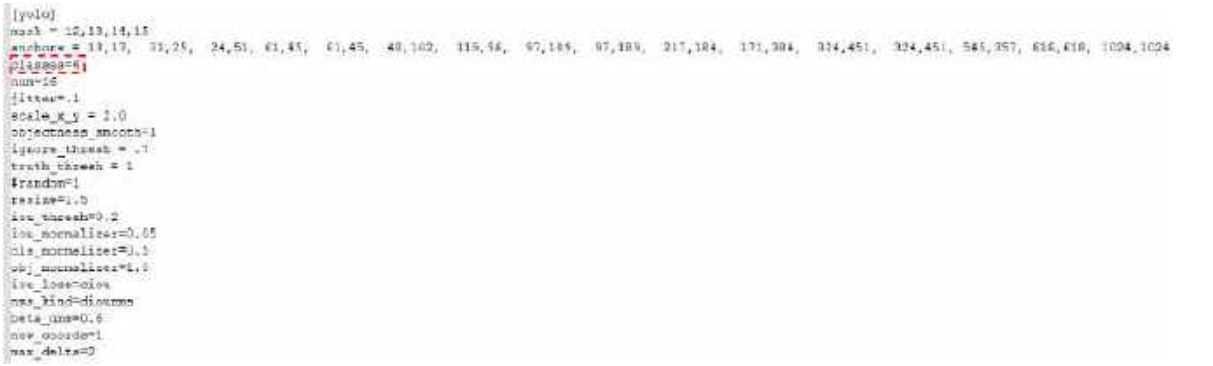

Figure 3.3 *Illustrates the location of "classes" within the configuration file*

The number of max batches

The number of max batches is the total number of iterations completed by the model throughout its training process. The reduction in the number of max batches allows the training process to be accelerated, which is advantageous when computational resources are limited. In the default configuration file of YOLOv4-p6, the "max batches" hyperparameter is initially set to 500,500. We adjusted this to 8,000 (see Figure 3.4) according to the following Equation 3.1 (Solawetz et al., 2020):

$$Max\_batches = 2000 \times n \tag{3.1}$$

where n is the number of classes. For our UAV dataset $n = 4$.

Equation 3.1 was provided by the model developers to determine the number of max batches for training based on a certain dataset. However, for large datasets such as COCO, this equation might not be ideal. For example, the original Scaled YOLOv4 models use 500,500 max batches instead of 160,000 $\left(2000 \times 80_{classes\ of\ the\ COCO\ dataset}\right)$. This is based on the fact that the COCO dataset is very large, and the developers attempted to achieve better results.

In our case, we used Equation 3.1 suggested by the developers to adjust the number of max batches for our experiments with the modified UAV dataset. Since this dataset contains 61,429 images in the training set, the training process passes through the entire training set 8 times (epochs) before it is completed.

The number of epochs is calculated as follows (GeeksforGeeks, 2023):

$$number\ of\ epochs = \frac{number\ of\ max\ batches}{total\ number\ of\ iterations\ per\ epoch} = \frac{8{,}000}{959.82} = 8{,}33 \approx 8 \tag{3.2}$$

where,

$$total\ number\ of\ iterations\ per\ epoch = \frac{number\ of\ images\ of\ the\ training\ set}{batch\ size} = \frac{61{,}429}{64} = 959.82 \tag{3.3}$$

Therefore, we did not follow the practices of the COCO dataset, as the modified UAV dataset has fewer classes and images in the training set (COCO has approximately 118,000 training images). If we had followed the COCO practices, the increasing number of max batches might cause overfitting due to the corresponding increase in the number of epochs (Ghosh et al., 2021), and it would have slowed down the training process.

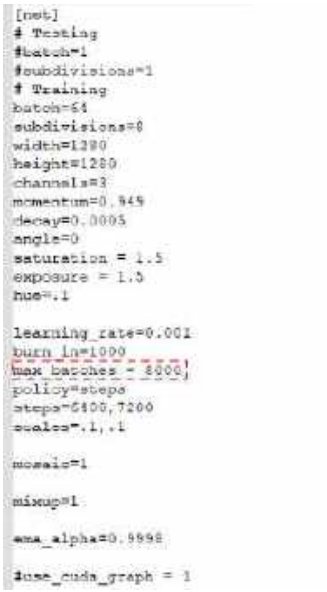

Figure 3.4 *Illustrates the location of "max_batches" within the configuration file*

The number of steps

The “steps” hyperparameter defines the iteration numbers at which the learning rate is adjusted during training. Initially, the learning rate remains constant for several iterations and then decreases at these specified points. Typically, these points are 80% and 90% of the maximum batch value used. Thus, since in our case max batches is equal to 8000, the steps hyperparameter is (Solawetz et al., 2020):

$$0{,}8 \ * \ max\ batches = 0{,}8 \ * \ 8000 = 6{,}400 \quad (3.4)$$

$$0{,}9 \ * \ max\ batches = 0{,}8 \ * \ 8000 = 7{,}200 \quad (3.5)$$

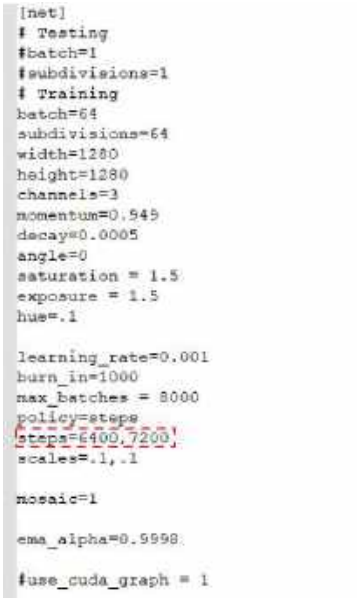

Figure 3.5 *Illustrates the location of "steps" within the configuration file*

The number of filters of the convolutional layer before each detection head

As shown in Figure 3.6, the filters placed in front of the convolution layer at each detection head are responsible for generating a specific number of outputs, which include number of classes, anchor box coordinates, objectness score and number of anchors. The filters are calculated by following the subsequent equation (Patel et al., 2021):

$$Filters = (number\ of\ classes + 5)\ *\ number\ of\ anchors \quad (3.6)$$

In Equation 3.6, the term "5" represents the following values:

- 4 values for the anchor box coordinates $(t_x, t_y, t_w\ and\ t_h)$
- 1 value for the objectness score

The number of filters placed in front of the convolution layer must follow Equation 3.6, because YOLO architectures are designed with a specific output dimension for each detection head (based on the number of anchors and the number of classes). A higher or lower number of filters will cause the algorithm not to run.

The YOLOv4-p6 model includes four detection heads specialized for very small, small, medium, and large objects, with each head containing four anchors. As per Equation 3.6, all our experiments have four anchors and four classes, thus requiring 36 filters. Consequently, we must adjust the filters before each detection head, reducing the number from 255 to 36.

```
[convolutional]
size=1
stride=1
pad=1
filters=36
activation=logistic
#activation=linear
# use linear for Pytorch-Scaled-YOLOv4, and logistic for Darknet

[yolo]
mask = 12,13,14,15
anchors = 13,17,  31,25,  24,51,  61,45,  61,45,  48,102,  119,96,  97,189,  97,189,  217,184,  171,384,  324,451,  324,451,  545,357,  616,618,  1024,1024
classes=4
num=16
jitter=.1
scale_x_y = 2.0
objectness_smooth=1
ignore_thresh = .7
truth_thresh = 1
#random=1
resize=1.5
iou_thresh=0.2
iou_normalizer=0.05
cls_normalizer=0.5
obj_normalizer=1.0
iou_loss=ciou
nms_kind=diounms
beta_nms=0.6
new_coords=1
max_delta=2
```

Figure 3.6 *Illustrates the location of "filters" within the configuration file*

In summary, the adjustments to the first set of hyperparameters are presented in Table 3.6:

Table 3.6 *Represents the adjusted values of the first set of hyperparameters*

First set of hyperparameters	Default	Adjusted
The number of classes	80	4
The number of max batches	500,500	8000
The number of steps	400,000, 450,000	6,400, 7,200
The number of filters of the convolutional layer before each detection head	255	36

3.5.2 Hyperparameters related to the model's architecture and operation

Image resize

At the beginning of the YOLOv4-p6 configuration file, many hyperparameters are specified in the [net] section, including "width" and "height" which adjust the input image dimensions. As shown in Figure 3.7, YOLOv4-p6 is trained with an image resolution of $1280x1280$ pixels. This high resolution helps the model collect more information as the input progresses through the network, but it also increases training time.

To investigate the trade-off between high image resolution and model effectiveness, we modified the "width" and "height" hyperparameters to reduced values for two main reasons:

- To determine if YOLOv4-p6 can still achieve good results with reduced input image size
- To identify the optimal combination of hyperparameters that produces the highest mAP. With a smaller input image size, training is completed faster, allowing us to test different hyperparameter combinations and identify the one that achieves the highest mAP result. We will then use this optimal combination to train the model at a higher input resolution to achieve even better results.

Therefore, we used two levels of image resolution: The default one of $1280x1280$, and a reduced one of $960x960$ pixels. Figure 3.7 shows how we adjusted the "height" and "width" hyperparameters.

Image resolution of 1280x1280

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=64
width=1280
height=1280
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
```

Image resolution of 960x960

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=8
width=960
height=960
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
```

Figure 3.7 *The locations of "width" and "height" within the configuration file*

Anchor box dimensions

The architecture of YOLOv4-p6 includes four detection heads, each with two specific hyperparameters: "mask" and "anchors" (refer to Figure 3.8). The hyperparameter "anchors" refers to anchor boxes, which are predefined bounding boxes of various sizes and aspect ratios, described in detail in the Appendix B.2 . As shown in Figure 3.8, the default YOLOv4-p6 model uses the following anchor box sizes: [13,17, 31,25, 24,51, 61,45, 48,102, 119,96, 97,189, 97,189, 217,184, 171,384, 324,451, 545,357, 616,618, 1024,1024]. Each detection head uses a different set of boxes to predict objects, which is selected by the "masks" hyperparameter at each head. The "masks" hyperparameter is an index containing four values that specify the anchor boxes used within each grid cell. Figure 3.8 illustrates a small detection head configured with the "masks" index [0, 1, 2, 3]. Specifically, index 0 corresponds to the anchor box dimensions of 13x17, index 1 to 31x25, and index 2 to 24x51.

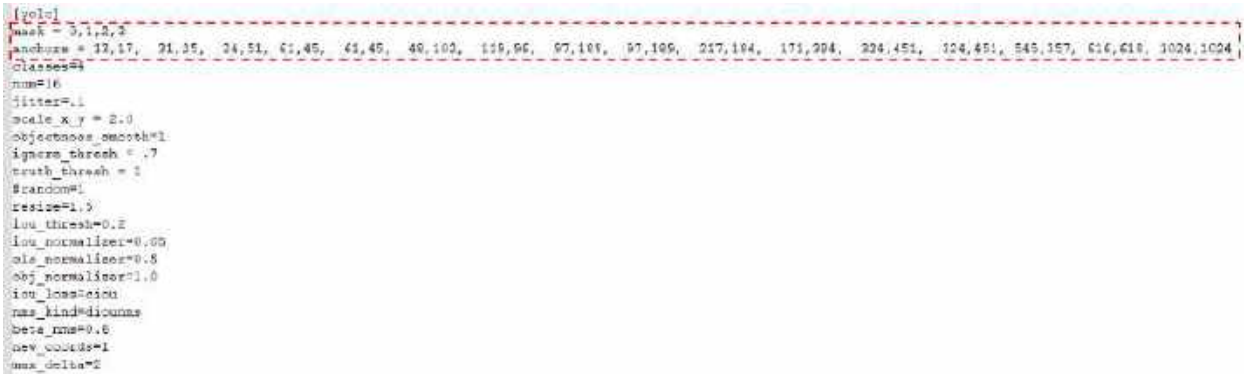

Figure 3.8 *The locations of "masks" and "anchors" within the configuration file*

Although anchor boxes are refined during training to better enclose objects, it is essential to initialize them correctly before training. This ensures that they can be refined effectively and faster during training. The original anchor boxes in YOLOv4-p6 were generated using the k-means algorithm applied to the COCO dataset. K-means clustering groups objects by their spatial characteristics and identifies centroids that define optimal anchor box dimensions (Oti et al., 2021). As shown in the Appendix B.2 (description of the command), we applied the same algorithm to our UAV dataset using the following command:

./darknet detector calc_anchors cfg/"NAME_OF_THE_DATA_FILE".data -num_of_clusters "NUMBER_OF_CLUSTERS" -width "NUMBER_OF_IMAGE_WIDTH" -height "NUMBER_OF_IMAGE_HEIGHT"
An example by applying the code: ./darknet detector calc_anchors cfg/traffic_lights.data -num_of_clusters 16 -width 1280 -height 1280

Figure 3.9 *Representation of the use of the k-means algorithm*

The application of k-means clustering is to calculate sixteen new optimal anchor box sizes based on our UAV dataset. Therefore, we used the k-means algorithm to identify clusters of anchor box sizes that best fit the objects in our images of the specified size.

As mentioned before, our tests use two different image resolutions: $1280x1280$ and $960x960$. Consequently, we executed the k-means algorithm twice to find the sixteen optimal anchor box sizes for

both resolutions. As shown in Table 3.7, we adjusted the "anchors" hyperparameters across all the detection heads in our tests according to the image resolution of each test, with different anchor box sizes applied for images of 1280x1280 and 960x960 pixels, respectively.

Table 3.7 *Representation of the anchor bounding boxes before and after the application of k-means algorithm*

Predetermined anchor box sizes	COCO	K-means 960x960	K-means 1280x1280
Scale/aspect ratio	[13,17, 31,25, 24,51, 61,45, 48,102, 119,96, 97,189, 97,189, 217,184, 171,384, 324,451, 324,451, 545,357, 616,618, 1024,1024]	[3,4, 10,9, 6,17, 11,28, 21,17, 26,31, 17,51, 52,34, 35,52, 27,104, 50,84, 79,57, 81,117, 49,198, 142,165, 145,382]	[4,5, 7,17, 16,13, 14,36, 29,24, 33,42, 21,67, 64,43, 45,69, 36,137, 99,72, 66,111, 110,150, 66,261, 189,221, 193,510]

Non-maximum suppression (NMS)

This methodology preserves the most accurate bounding box while effectively reducing duplications caused by overlapping candidates. To do so, it considers both the confidence score assigned by the model and the degree of overlap, measured by the Intersection over Union (IOU) metric, among the bounding boxes (Hosang et al., 2017). For instance, in Figure 3.10, the model generates multiple bounding boxes, along with their respective confidence scores, to identify the car in the image. After the application of the NMS algorithm, only one bounding box will be selected.

NMS takes as input a list of bounding boxes that have predicted an object, along with their confidence scores and the overlap threshold. For example, the $IoU_{threshold}$ can be set to 0.5, which means that all correctly placed bounding boxes have $IoU \geq IoU_{threshold}$. The output of the algorithm is a list of filtered proposals, where each object instance corresponds to one optimal bounding box.

The approach for determining the optimal bounding box through the application of NMS is described below (Singh, 2024):

1. Begin by selecting the bounding box with the highest confidence score
2. Transfer this bounding box from the input list to the final proposal list
3. Calculate and compare the IoU (overlap) of this chosen bounding box with the remaining bounding boxes
4. Remove any proposals from the input list that have an IoU greater than the $IoU_{threshold}$. This ensures that highly overlapping proposals are filtered out
5. If there are still bounding boxes remaining in the input list, repeat steps 1 through 4 until there are no more bounding boxes that meet the IoU condition or are left in the input list.

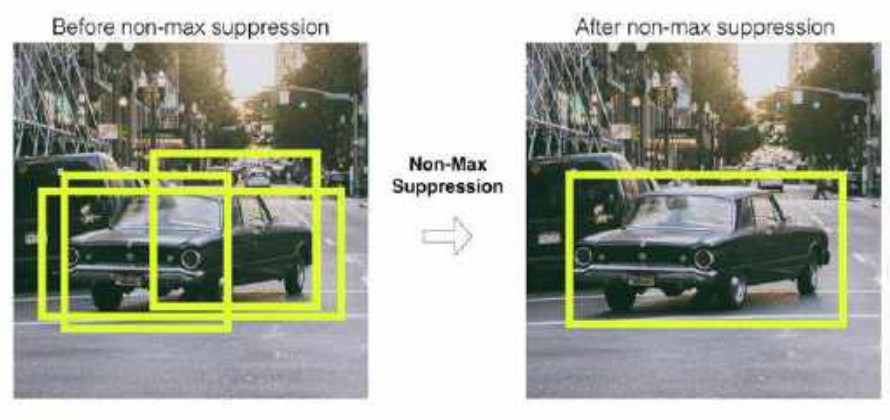

Figure 3.10 *The bounding boxes preceding and following the implementation of the Non-Max Suppresion (NMS) algorithm(Świeżewski, 2020)*

The YOLOv4 and YOLOv4-p6 algorithms use two different variations of Non-Maximum Suppression (NMS). Specifically, YOLOv4 uses Greedy-NMS, while YOLOv4-p6 uses Distance-Intersection over Union - Non-Maximum Suppression (DIoU-NMS).

In this study, we propose to use Greedy-NMS, in addition to DIoU-NMS, for YOLOv4-p6. Our objective is to determine whether Greedy-NMS or DIoU-NMS produces superior or inferior results when combined with the other selected hyperparameters.

Greedy-NMS

Greedy Non-Maximum Suppression (Greedy-NMS) is designed to refine detection results by iteratively selecting the bounding box with the highest confidence score and suppressing others that overlap it. Specifically, it is used to reduce duplicate detections and reduce false positives. However, in scenarios where objects are crowded together, Greedy-NMS faces difficulties. Even with a reliable detector that accurately identifies bounding boxes that match the ground truth bounding boxes, Greedy-NMS can still struggle due to its dependence on a set $IoU_{threshold}$ (usually set at 0.5). This threshold determines when overlapping boxes are considered duplicates and thus suppressed. A lower IoU threshold might fail to filter out highly overlapped objects effectively, leading to more false positives in the final detections. Conversely, increasing the $IoU_{threshold}$ could result in more objects being incorrectly suppressed, potentially missing legitimate detections. Therefore, while Greedy-NMS offers a simple approach to post-processing in object detection, its effectiveness can vary depending on the density and arrangement of objects within an image (Liu and Huang, 2019).

Distance-Intersection over Union - Non-Maximum Suppression (DIoU-NMS)

The Distance Intersection over Union (DIoU) method (Zheng et al., 2019b) improves the effectiveness of NMS by considering both the IoU and the distance between the central points of the bounding boxes.

Equation 3.8 defines the criterion for retaining or removing a bounding box B_i based on its confidence score s_i. Equation 3.7 defines the DIoU metric $R_{DIoU}(M, B_i), M$ is the bounding box with the highest confidence score.

$$R_{DIoU}(M, B_i) = \frac{\rho^2(b, b^{gt})}{c^2} \tag{3.7}$$

where,

- b and b^{gt}: represent the central points of the predicted bounding box (B) and the ground truth bounding box (B^{gt}), respectively
- ρ: denotes the Euclidean distance between the central points b and b^{gt}
- c: represents the diagonal length of the smallest enclosing box that includes the predicted bounding box and the ground truth bounding box.

The suppression criterion is set by ε, which determines whether a box should be suppressed. In the YOLOv4-p6 model, this suppression criterion is based on the $IoU_{threshold}$, which is set to 0.2. If the difference between IoU and DIoU is less than ε, the box B_i is retained (i.e., its confidence score s_i remains unchanged). Otherwise, if this difference is greater than or equal to ε, the box B_i is removed (i.e., its confidence score s_i is set to 0).

$$s_i = \begin{cases} S_i \,, IoU - R_{DIoU}(M, Bi) < \varepsilon \\ 0 \,, IoU - R_{DIoU}(M, Bi) \geq \varepsilon \end{cases} \tag{3.8}$$

Figure 3.11 compares the performance of Non-Maximum Suppression (NMS) with Distance-IoU Non-Maximum Suppression (DIoU-NMS) for detecting ships in UAV images. The original images (top row) show multiple ships scattered across the water. The middle row with NMS results indicates areas where some ships are missed, marked by green arrows. The bottom row with DIoU-NMS results shows improved detection, identifying more ships in areas where NMS was less effective. The green arrows highlight the locations where DIoU-NMS successfully detected ships that NMS missed.

Figure 3.11 *Illustrates a comparison between NMS and DIoU-NMS for ship detection (Chen et al., 2023)*

The application of the two NMS levels requires only a minimal implementation in the code. As shown in Figure 3.12, we can adjust the NMS within each detection head by modifying the "nms_kind" hyperparameter. Specifically, DIoU-NMS is used by adding “diounms” next to “nms_kind”, and Greedy-NMS is used by adding “greedynms” next to “nms_kind”, respectively.

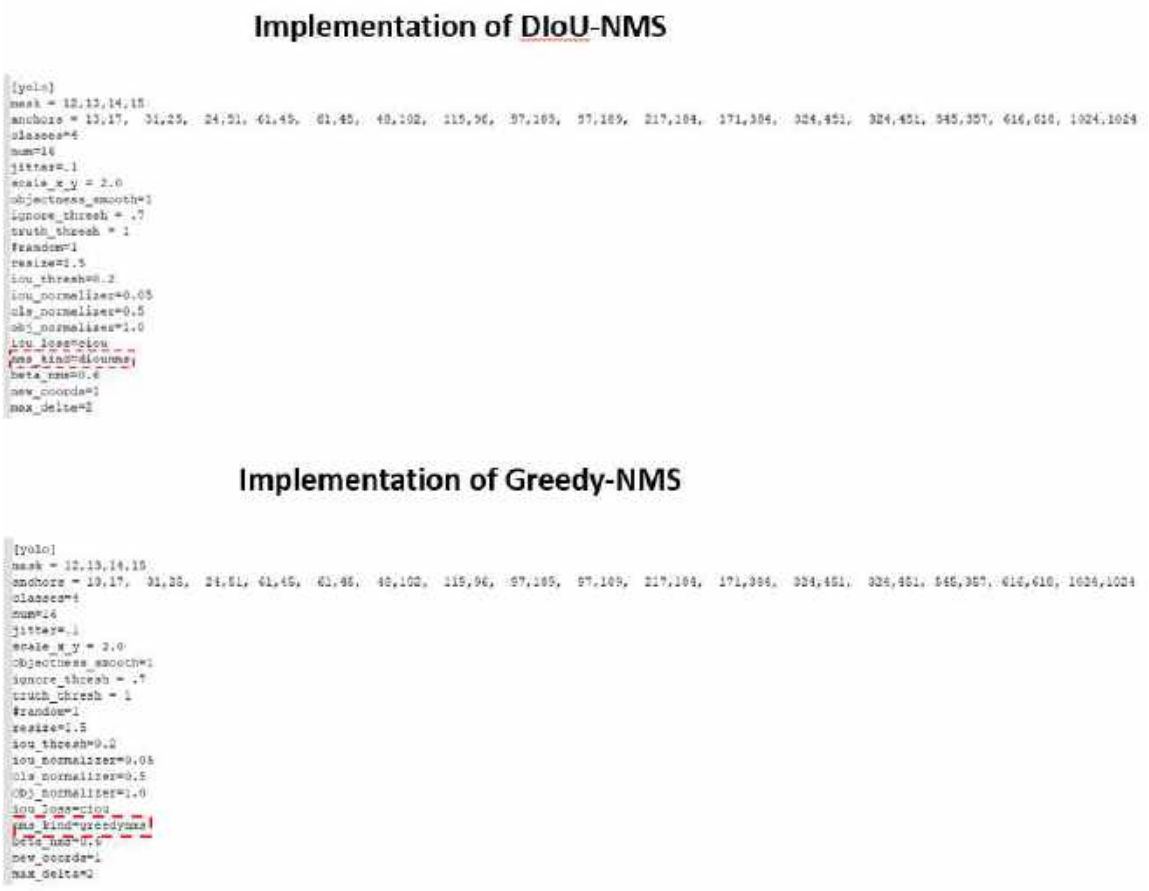

Figure 3.12 *Illustrates the nms_kind location within the configuration file and the implementation of DIoU-NMS and Greedy-NMS*

Data augmentation techniques

YOLOv4 introduces innovative data augmentation techniques that are used to improve the training performance of our model. Specifically, data augmentation techniques produce altered versions of the input images while preserving their annotations, thus expanding the training set (Solawetz, 2020a). As a result, these augmentation techniques were created to enhance the model's training process by enriching the training dataset.

For the creation of our tests, we used either the Mosaic technique alone or the combination of Mosaic and Mixup techniques. Our objective is to determine which of the two approaches produces superior or inferior results when combined with the other selected hyperparameters. The two techniques are described below:

Mosaic

The “Mosaic” data augmentation technique has been introduced by Bochkovskiy et al. in 2020 as part of the YOLOv4 model. It combines four randomly selected input images (along with their annotations) during the training process (refer to Figure 3.13). Therefore, it merges different sections from four images into one. This approach challenges the model with different backgrounds and objects within each training batch. As a result, it helps to improve the model's ability to detect objects in unlikely scenarios (Bochkovskiy et al., 2020).

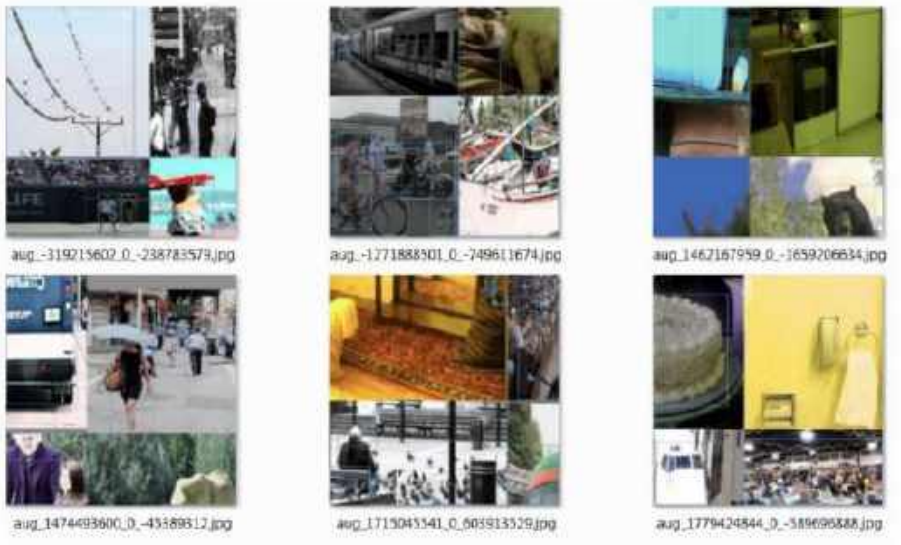

Figure 3.13 *Implementing the Mosaic technique on image datasets (Solawetz, 2020b)*

This method is specified within the [net] section of the configuration file, as depicted in Figure 3.14.

```
[net]
batch=64
subdivisions=8
# Training
#width=512
#height=512
width=608
height=608
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.0013
burn_in=1000
max_batches = 500500
policy=steps
steps=400000,450000
scales=.1,.1

mosaic=1
```

Figure 3.14 *Illustrates the location of the mosaic data augmentation technique*

Mixup

Mixup is a data augmentation technique designed for classification tasks, which works by combining two randomly selected training instances and their corresponding annotations to create a new sample (Zhang et al., 2018).

As illustrated in Figure 3.15, the process involves creating a new synthetic image by blending two images of dogs: one of a St. Bernard and another of a Poodle. The Mixup technique generates this new image by forming a weighted combination of the original images and their associated annotations. The weights for this combination are determined by a mixing coefficient, which is randomly chosen from a beta distribution. The beta distribution is a probability distribution defined on the interval [0, 1].

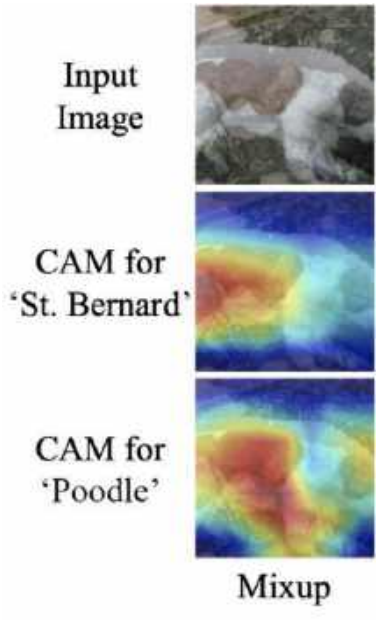

Figure 3.15 *Implementing the MixUp technique on image datasets (Yun et al., 2019b)*

In the example with the St. Bernard and Poodle images, consider the following mixing coefficient value $\lambda_1 = 0.3$. According to the MixUp method, a new synthetic image x and label y are generated using the following equations:

$$x = \lambda_1 * x_{St.Bernard} + (1 - \lambda_1) * x_{Poodle} \quad (3.9)$$

$$y = \lambda_1 * y_{St.Bernard} + (1 - \lambda_1) * y_{Poodle} \quad (3.10)$$

Here, $x_{St.Bernard}$ and x_{Poodle} are the image representations of the St. Bernard and Poodle, respectively, and $y_{St.Bernard}$ and y_{Poodle} are their corresponding labels. With $\lambda_1 = 0.3$, the new image x mostly reflects the features of the Poodle image (70% influence), while also incorporating some characteristics of the St. Bernard image (30% influence). Similarly, the label y for this new synthetic sample blends the labels "St. Bernard" and "Poodle" based on the same ratio.

By sampling from the beta distribution, Mixup obtains random mixing coefficients. These coefficients determine the degree to which each instance contributes to the final merged sample. Consequently, the new sample and its label represent a blend of the characteristics of the St. Bernard and Poodle images, with the influence of each instance varying according to the sampled coefficient.

To apply the two data augmentation levels, simply add them to the [net] section of the configuration file and set their values to 1, as shown in Figure 3.16. This way we can implement mosaic, mixup, or any other data augmentation technique.

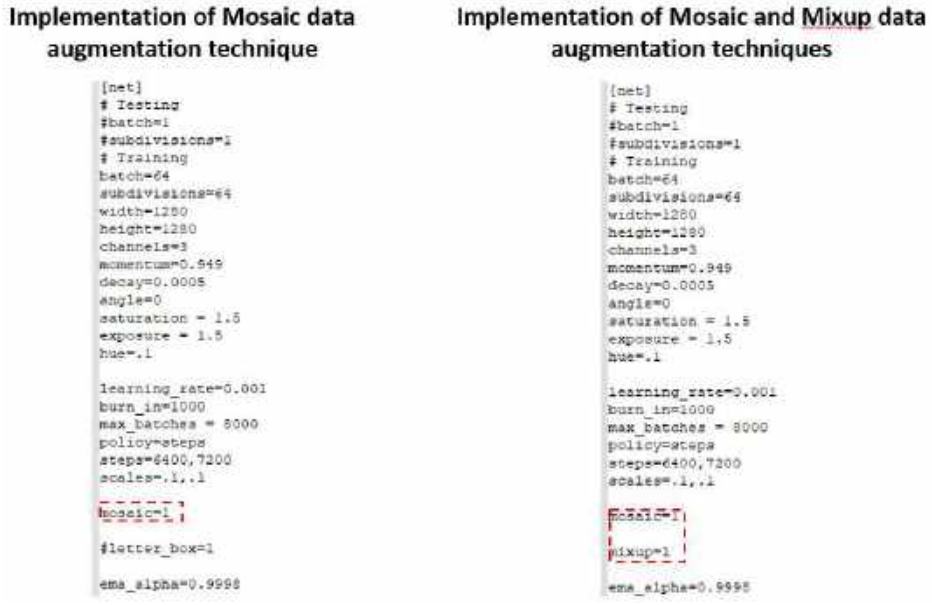

Figure 3.16 *Illustrates the placement of data augmentation techniques within the configuration file*

Activation function

Activation functions are included in neural networks to add non-linear characteristics to the outputs of neurons in network layers. Thereby, with the application of activation functions, the model is capable of detecting complicated patterns.

For the creation of our tests, we applied two activation functions: Mish and Swish. Specifically, we used the Mish activation function for a certain number of tests, and we used the Swish activation function for the remaining ones. Our objective is to determine whether the Mish or Swish activation function produces superior or inferior results when combined with the other selected hyperparameters. These two activation functions are described below:

Swish activation function

The Swish is a smooth and non-monotonic activation function, which consistently shows performance comparable to or better than the Rectified Linear Unit (ReLU). As depicted in Figure 3.17, it is unbounded

in its upper range and bounded in its lower range, deriving its distinctive non-monotonic characteristic (Ramachandran et al., 2017).

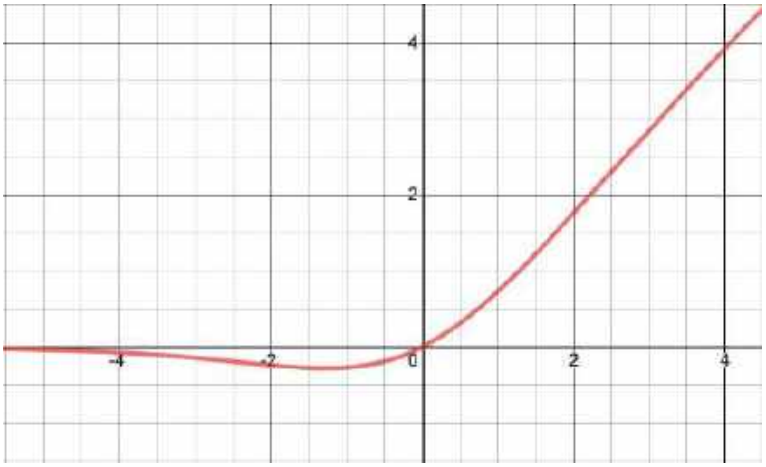

Figure 3.17 *Illustration of Swish activation function (Singh, 2020)*

Swish is formally expressed as an activation function represented by the following equation:

$$f(x) = x * \sigma(\beta x) \tag{3.11}$$

where,

- σ: symbolizes the sigmoid activation function (Nwankpa et al., 2018), which is expressed as:

$$\sigma(x) = \frac{1}{(1 + e^{-x})} \tag{3.12}$$

- β: can be either a constant or a trainable parameter. This parameter results into the following cases:
  - If $\beta = 0$, the Swish activation function is simplified to a scaled linear function
  - As β tends to infinity, the Swish function approximates the ReLU function
- x: is the input value for the Swish activation function.

In Equation 3.11, the Swish activation function is using the self-gating method, where the function uses its own value to control or "gate" itself. Instead of using an external or different value to influence the output (like ReLU function), the function initially modifies the input using $\sigma(\beta x)$ and then combines this modified value with the original input x. Here, the input x is multiplied with the $\sigma(\beta x)$.

Mish activation function

Mish is the updated version of the Swish activation function, designed to have similar characteristics to Swish (Misra, 2019). That is, Mish is also a smooth and non-monotonic activation function like Swish, and it is unbounded in its upper range and bounded in its lower range (see Figure 3.18).

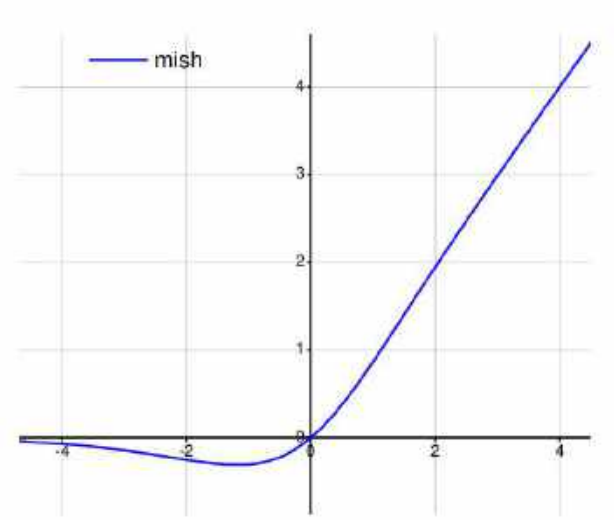

Figure 3.18 *Illustration of Mish activation function (Li et al., 2022)*

The Mish activation function is characterized by the subsequent equation:

$$f(x) = x * \tanh(\ln(1 + e^x)) \tag{3.13}$$

Similarly to Swish, Mish uses the self-gating method. As shown in Equation 3.13, the input x is multiplied with $\tanh(\ln(1 + e^x))$.

Thanks to Equation 3.13, Mish provides a smoother transition from negative to positive values to improve the flow of information during training. Consequently, it has improved training stability, resulting in the reduction of vanishing gradients issue (see Figure 3.18, Mish has a sharper rise and narrower spread on the graph). Despite these improvements, both activation functions are considered effective for training the YOLOv4-p6 model.

Mish was applied to the original YOLOv4 and Scaled YOLOv4 models. It was applied to all convolutional layers throughout their structures, except for the convolutional layers preceding each detection head. These convolutional layers use logistic activations, which are referred to as sigmoid function (Nwankpa et al., 2018).

Figure 3.19 illustrates the application of Mish activation function in the configuration file. To convert the Mish into the Swish activation function, replace “activation = mish” with “activation = swish” in all convolutional layers throughout the structure, except for the convolutional layers preceding each detection head.

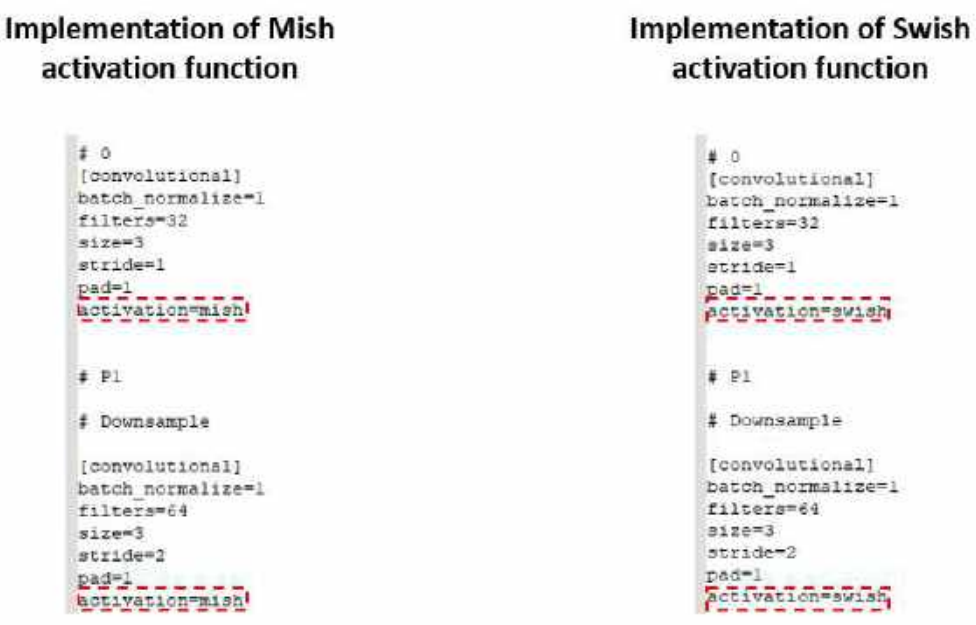

Figure 3.19 *Illustrates the implementation of activation functions within the configuration file*

Table 3.8 presents the second set of hyperparameters, adjusted to the following two levels, which will be used to create our experiments:

Table 3.8 *Represents the two levels of the second set of hyperparameters*

Second set of hyperparameters	First level (default)	Second level
Image resize	1280 x 1280 pixels	960 x 960 pixels
Anchor box dimensions	[13,17, 31,25, 24,51, 61,45, 48,102, 119,96, 97,189, 97,189, 217,184, 171,384, 324,451, 324,451, 545,357, 616,618, 1024,1024]	For 1280 x 1280: [4,5, 7,17, 16,13, 14,36, 29,24, 33,42, 21,67, 64,43, 45,69, 36,137, 99,72, 66,111, 110,150, 66,261, 189,221, 193,510]
		For 960 x 960: [3,4, 10,9, 6,17, 11,28, 21,17, 26,31, 17,51, 52,34, 35,52, 27,104, 50,84, 79,57, 81,117, 49,198, 142,165, 145,382]
Non-maximum suppression (NMS)	DIoU-NMS	Greedy-NMS
Data augmentation techniques	Mosaic	Mosaic + Mixup
Activation function	Mish	Swish

Chapter 4 Experimental Investigation

In this Chapter, we present the experiments we conducted to optimize the performance of the YOLOv4-p6 model by systematically adjusting its hyperparameters that affect model training. The performance of the model was assessed by the mean Average Precision (mAP) metric. To assess the effect of each hyperparameter and each hyperparameter interaction on mAP we used Analysis of Variance (ANOVA).

4.1 Full Factorial Experiments

Full factorial designs are used to ensure that the effects of all hyperparameters and of all their interactions may be determined from the experimental results. One major drawback is that full factorial designs are more extensive compared to fractional factorial designs. In our case, however, the limited number of factors (5) and levels (2) allows the use of the full factorial design (see also (JMP, 2023). The design comprises thirty-two (2^5) experiments presented in Table 4.1.

Table 4.1 *The 2^5 full factorial design*

Model Number	Image resize	Activation function in the structure*	Non-maximum suppression	Data augmentation	Anchor box dimensions
1	1280*1280	mish	diounms	mosaic	default
2	960*960	mish	diounms	mosaic	default
3	1280*1280	swish	diounms	mosaic	default
4	960*960	swish	diounms	mosaic	default
5	1280*1280	mish	greedynms	mosaic	default
6	960*960	mish	greedynms	mosaic	default
7	1280*1280	swish	greedynms	mosaic	default
8	960*960	swish	greedynms	mosaic	default
9	1280*1280	mish	diounms	mosaic + mixup	default
10	960*960	mish	diounms	mosaic + mixup	default
11	1280*1280	swish	diounms	mosaic + mixup	default
12	960*960	swish	diounms	mosaic + mixup	default
13	1280*1280	mish	greedynms	mosaic + mixup	default
14	960*960	mish	greedynms	mosaic + mixup	default
15	1280*1280	swish	greedynms	mosaic + mixup	default
16	960*960	swish	greedynms	mosaic + mixup	default
17	1280*1280	mish	diounms	mosaic	new
18	960*960	mish	diounms	mosaic	new
19	1280*1280	swish	diounms	mosaic	new
20	960*960	swish	diounms	mosaic	new
21	1280*1280	mish	greedynms	mosaic	new
22	960*960	mish	greedynms	mosaic	new
23	1280*1280	swish	greedynms	mosaic	new

Model Number	Image resize	Activation function in the structure*	Non-maximum suppression	Data augmentation	Anchor box dimensions
24	960*960	swish	greedynms	mosaic	new
25	1280*1280	mish	diounms	mosaic + mixup	new
26	960*960	mish	diounms	mosaic + mixup	new
27	1280*1280	swish	diounms	mosaic + mixup	new
28	960*960	swish	diounms	mosaic + mixup	new
29	1280*1280	mish	greedynms	mosaic + mixup	new
30	960*960	mish	greedynms	mosaic + mixup	new
31	1280*1280	swish	greedynms	mosaic + mixup	new
32	960*960	swish	greedynms	mosaic + mixup	new

* Activation functions in structure are those used in all convolutional layers of the model, except for the convolutional layers before each detection head

The above Table contains all possible combinations of the second set of hyperparameters (as defined in Subsection 3.5.2), using both levels (default and new). Note that for the first set of hyperparameters we applied the adjustments described in Subsection 3.5.1 and we kept them invariant throughout the experimental study. The remaining hyperparameters (those not included in the first and second sets), such as batch size, were kept at their default levels.

4.2 Experiment execution

According to Table 4.1, we created thirty-two configuration files (.cfg files) and placed them in a folder, as shown in Figure 4.1. Each .cfg file contains a combination of hyperparameters as shown in Table 4.1.

Figure 4.1 *Folder including both .cfg and .data files*

Together with each .cfg file, there is a .data file with the same name. For our experimentation, we named each file "YV4P6" followed by a number from 1 to 32 to indicate the combination presented in Table 4.1. Figure 4.2 presents the information included in each .data file.

```
classes= 4
train  = /media/deopsys/Hard_Disk/panos/Final_dataset(swapped_classes)/Paths/Train.txt
valid  = /media/deopsys/Hard_Disk/panos/Final_dataset(swapped_classes)/Paths/Val.txt
names = /home/deopsys/Documents/darknet/data/yolov4_4_classes.names
backup = /media/deopsys/Hard_Disk/panos/Experiments/cfg1
```

Figure 4.2 *Information included in the first .data file corresponding to the first .cfg file used for training*

where,

- classes: shows that the model is being trained to detect four different object classes (person, small vehicle, large vehicle, and ship)
- train: specifies the path to a text file (Train.txt) that lists the file paths of all training images. As mentioned in Section 3.3, we split our modified UAV dataset into three sets: training, validation, and testing sets
- valid: specifies the path to a text file (Val.txt) that lists the file paths of all validation images
- names: specifies the path to a text file that contains the names of the object classes (see Figure 4.3)

```
person
small vehicle
large vehicle
ship
```

Figure 4.3 *Txt file including the object classes*

- backup: specifies the directory where the trained model weights will be saved. During training, we set up the algorithm to save the trained weights every 1000 iterations, as will be explained later in this Section.

The only difference between each .data file is the "backup" path, which changes (corresponding to each .cfg file) to prevent overwriting the generated training weights.

The training process is conducted through the following command:

```
for i in {1..32}; do ./darknet detector train cfg/cfg_panos/Experiments/YV4P6_${i}.data
cfg/cfg_panos/Experiments/YV4P6_${i}.cfg yolov4-p6.conv.289 -map | tee
/media/deopsys/Hard_Disk/panos/Experiments/cfg${i}.txt; done
```

Figure 4.4 *Execution command for conducting the experiments*

This contains a loop from 1 to 32, which selects iteratively the .cfg file (e.g. YV4P6_1.cfg) and the corresponding .data file (e.g. YV4P6_1.data). Additionally, it uses the pre-trained weights "yolov4-p6.conv.289" that the developers published as initial weights in the process. The command also saves the training progress chart (see Figure 4.9) in the Darknet directory and terminal information in line reports

(as shown in Figures 4.5 and 4.6) in a chosen path. Thus, once the training of the first model is completed, the process moves to the second model with the same pre-trained weights, saving the training progress chart and terminal information in line reports under a unique file name to avoid overwriting the results of the previous model. This continues until all thirty-two experiments have been trained. As discussed already, with the application of this command, we obtained the training progress chart, model weights, and a text file containing the report information for each experiment.

Once all thirty-two experiments were completed, we re-trained them all over again, resulting in a total of sixty-four experiments. The reasons for conducting each experiment twice are the following:

- To facilitate the analysis of the results (using ANOVA) by quantifying the variability within groups
- To evaluate the robustness of training under the same hyperparameter settings. Small differences between the mAP results of the two runs indicate that the model is trained consistently.

In Section 3.4, we have presented the components of our lab's system. Despite its computational power, the YOLOv4-p6 required 60 hours to complete one experiment with an image resolution of 1280x1280, and approximately 36 hours for an experiment with a 960x960 image resolution. With the application of the command presented in Figure 4.4, the first run of thirty-two experiments took forty-five days to complete, and ninety days to complete both runs. As a result, the primary and only problem we encountered during training was the extended training time.

The report presented in Figure 4.5 provides the evolution of key quantities of the training process per iteration.

```
1606: 11.410105, 9.284472 avg loss, 0.001000 rate, 16.538288 seconds, 102784 images, 48.912915 hours left
```

Figure 4.5 *Key quantities of the training process*

In this Figure,

- 1,606 is the ID of the current training iteration
- 11.410105 is the total loss value
- 9.284472: is the average loss. The purpose is to minimize the average loss, approaching zero if possible
- 0.001000 is the current learning rate, which is defined in the YOLOv4-p6 configuration file
- 16.538288 is the total time spent processing the batch identified by iteration ID 1,606
- 102,784: is the total number of images used in training up to this batch. This total is calculated by multiplying the number of batches (1,606) by the batch size (64), resulting in 102,784 images.

Part of the overall training process is the validation process. The latter tests the weights that the algorithm creates during training and generates a mAP result for each validation run (see Figure 4.6). We set up our models to begin their first validation run in the 600th iteration and then repeat it every 100 iterations (e.g.,

700, 800, 900, 1000, …) until the 8000th training iteration, where the training process of one model is completed. This means that each experiment included seventy-five validation runs and generated seventy-five different mAP results. Furthermore, the trained weights were saved every 1000 iterations in the path we set in the .data file (backup path).

Figure 4.6 presents a representative example of the results of one validation run (Tepteris et al., 2023). The results change from one validation run to another as the model constantly updates its training weights.

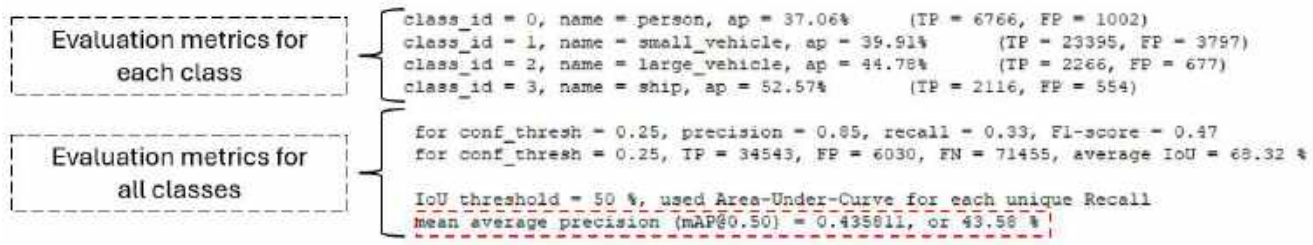

Figure 4.6 *Display of key class metrics during validation*

In the validation report, for each class, the following are provided:

- $class\_id$: For example, index value 3 represents the class "ship"
- $name$: the name of the class
- $AP\ (Average\ Precision)$: provides the average precision result of the class
- $TP\ (True\ Positive)$: is the number of correctly identified class objects
- $FP\ (False\ Positive)$: is the number of incorrectly identified class objects
- $Recall$: is the ratio of true positives to the sum of false negatives and true positives for the particular class
- $avg\ IoU$: represents the average Intersection over Union (IoU) across all images in the validation set for the particular class.

It is reminded that the evaluation metrics for all classes in all training images are:

- $Precision$: is the overall prediction accuracy across all classes
- $Recall$: evaluates the ability of the detector to locate the trained objects within the image for all classes
- $F1-score$: is the harmonic mean of precision and recall
- $TP\ (True\ Positive)$: is the number of correctly detected objects across all classes
- $FP\ (False\ Positive)$: is the number of incorrectly detected objects across all classes
- $FN\ (False\ Negative)$: is the number of missed detections across all classes
- $average\ IoU$: represents the average IoU across all images in the validation set for all classes
- $mAP\ (mean\ Average\ Precision)$: summarizes the Average Precision (AP) of each individual class and then divides the total AP value by the number of the classes.

After the training process was completed, we performed the testing process for each of our thirty-two (times two) experiments. Both in training and testing, the model generates multiple results used for evaluation, as presented in Figure 4.6. From all the metrics contained in the line reports, we used the mean Average Precision (mAP) metric to evaluate the performance of the models (from training and testing). However, in each of these stages, the mAP results are used for different purposes:

- During the training process, the model adjusts its parameters to better detect the objects of interest in the input data. These adjusted parameters are stored in the weights. To ensure the model is learning correctly from the data, these weights are used in validation, an intermediate process. In this process, the model applies the trained weights to the validation set to test its detection and classification capabilities. During the validation process the model's detection metrics, including mAP, are computed and presented throughout training and are displayed in the progress chart (showing only mAP results). Consequently, the mAP results during training are used to confirm whether the models is being trained correctly. Models achieving high mAP values in validation indicate that overfitting is avoided and training in the related dataset is progressing well
- During the testing process, we use the best weights of the model (which is generated in training and contains the parameters that achieved the highest mAP result during validation) on the testing set. The testing results (including mAP) confirm the detection and classification performance of the model in independent data. If the value of the testing mAP is close to the better mAP values obtained from validation, then the model has been trained and performs as expected. If the testing mAP value is significantly lower than the better validation mAP values, this indicates that even under the same type of data the performance of the model is lower than expected (and obtained during validation), and, thus, training has not been successful.

For testing our trained models, we used the testing set of the modified UAV dataset (as discussed in Section 3.3) and the respective best weights (which achieved the highest mAP in validation) obtained from both training runs. Furthermore, we modified the information contained in the .data files and left the .cfg files unchanged (see Figure 4.1). Specifically, for testing, all .data files contained the information displayed in Figure 4.7. To execute the command for performing the testing process, as shown in Figure 4.8, we replicated and numbered the same .data file thirty-two times (as shown in Figure 4.1) to match each one with a corresponding .cfg file.

Figure 4.7 presents the information included in the .data files used for testing our trained YOLOv4-p6 models.

```
1  classes= 4
2  valid  = /media/deopsys/Hard_Disk/panos/Final_dataset(swapped_classes)/Paths/Test.txt
3  names = /home/deopsys/Documents/darknet/data/yolov4_4_classes.names
```

Figure 4.7 *Information included in all .data files used for testing*

In Fig. 4.7, the “classes” and “names” parameters remain the same as those used in training. The “valid” parameter was changed to specify the path to a text file (Test.txt) that contains the file paths of all testing images. Additionally, the “train” and “backup” parameters were omitted in the .data files for testing, as they are only used for training.

The following command is used to execute the testing process after completing both runs in training per hyperparameter combination.

```
for i in {1..32}; do ./darknet detector map cfg/cfg_panos/Testing/YV4P6_${i}.data
cfg/cfg_panos/Testing/YV4P6_${i}.cfg
/media/deopsys/Hard_Disk/panos/Weights/First_run/4V4P6_${i}_best.weights -points 101 -thresh
0.25 -iou_thresh 0.5 > /media/deopsys/Hard_Disk/panos/Testing/testing_${i}.txt; done
```

Figure 4.8 *Execution command for testing the experiments*

The command contains a loop from 1 to 32, which selects iteratively the .cfg file (e.g. YV4P6_1.cfg), the corresponding .data file (e.g. YV4P6_1.data) and the best weights file created from the first training run of the corresponding .cfg file. The command also saves the terminal information in line reports in a chosen path. Thus, once the testing of the first model (first .cfg file) is completed, the process moves to the second model with its corresponding best weights file, saving the terminal information in line reports under a unique file name to avoid overwriting the results of the previous model. This continues until all thirty-two models have been tested using their corresponding best weights from the first training run. Once testing of the 32 models are completed, we execute the same command using the best weights files from the second training run. Consequently, from both runs of the testing process, we obtained line reports that contain the mAP metric of testing for each model of each run.

4.3 Experimental results and analysis

In this Section, we analyze the results obtained from training (generated during the validation process) and testing runs.

- To characterize the training performance of each model, we extracted the highest mAP (referred to as the best mAP) produced during the model’s validation process. Note that during each training session, 75 validation mAP values were obtained (one every 100 iterations after iteration 600), and the highest among them is designated as the best mAP.
- In the case of testing, we used the single mAP result obtained from applying the model on the testing dataset.

Table 4.2 presents the mAP results from the training and testing runs of YOLOv4-p6 using the modified UAV dataset. In this table, we used both the best and average mAP to analyze our results. Since each process (training and testing) was executed twice, we also used the average mAP, which is the average of the best mAP from the two runs of each model during training or testing. This metric provides a clearer and more balanced view of the results by incorporating both runs, rather than relying on a single run's outcome. Consequently, the best mAP indicates the peak performance of each model in either training or testing, while the average mAP reflects the overall performance across both runs.

The differences between the two runs (Δ % (1-2)) characterize the consistency of the models per hyperparameter combination. Small differences suggest that the models are consistent and stable, while larger differences indicate inconsistency. The trained YOLOv4-p6 models demonstrated consistent performance in both the training and testing processes, as there are no significant differences between the two runs (under 4-5% difference between the two runs in both processes).

What is more important is the last column of Table 4.2 that indicates the differences between the average mAP values of validation/training vs testing. Two important observations are relevant here:

1. The difference values in the last column are low. Moreover, the testing average mAP is higher than the validation average mAP, indicating robust model performance and no overfitting during training. Of course, the testing dataset is (an independent) part of the modified UAV dataset, and thus similar performance is expected between validation and testing in case of proper training. The performance in totally new image dataset (such as the DeOPSys one) is expected to be lower.
2. The effect of the hyperparameters appears to be similar in training/validation and in testing. This is analyzed further below.

Table 4.2 *mAP results obtained from training and testing*

Model	Training process				Testing process				Difference between the average mAP values (training - testing)
	Best mAP run 1(%)	Best mAP run 2 (%)	Δ % (1-2)	Average best mAP % (1-2)	mAP run 1 (%)	mAP run 2 (%)	Δ % (1-2)	Average mAP % (1-2)	
1	49.2	51.6	-2.4	50.4	51.4	53.4	-2.0	52.4	-2.0
2	44.3	43.0	1.3	43.7	46.8	47.2	-0.4	47.0	-3.3
3	46.8	49.3	-2.5	48.0	49.6	51.3	-1.7	50.5	-2.4
4	43.4	39.1	4.3	41.3	47.5	43.2	4.2	45.3	-4.1
5	52.3	50.1	2.3	51.2	53.5	52.9	0.6	53.2	-2.0
6	44.2	41.0	3.1	42.6	48.0	44.0	4.0	46.0	-3.4
7	45.0	47.8	-2.7	46.4	47.3	51.3	-4.1	49.3	-2.9
8	39.7	40.0	-0.3	39.8	43.8	44.2	-0.3	44.0	-4.2
9	48.5	50.1	-1.7	49.3	51.5	52.6	-1.2	52.1	-2.8
10	41.2	43.9	-2.7	42.5	46.5	48.0	-1.5	47.3	-4.7
11	49.1	46.0	3.0	47.6	51.0	48.7	2.2	49.9	-2.3
12	39.6	40.4	-0.8	40.0	44.0	45.5	-1.5	44.7	-4.7
13	51.4	47.8	3.6	49.6	52.7	49.9	2.7	51.3	-1.7
14	44.1	44.3	-0.2	44.2	47.0	48.4	-1.4	47.7	-3.5
15	48.2	46.9	1.3	47.5	49.8	50.4	-0.6	50.1	-2.5
16	39.2	40.9	-1.7	40.0	42.9	45.5	-2.5	44.2	-4.2
17	50.7	52.5	-1.8	51.6	52.1	53.5	-1.4	52.8	-1.1
18	43.9	45.7	-1.8	44.8	47.6	48.8	-1.2	48.2	-3.4
19	51.3	49.1	2.2	50.2	51.9	50.2	1.7	51.0	-0.8
20	38.8	42.5	-3.7	40.7	41.4	46.7	-5.3	44.1	-3.4
21	51.9	50.4	1.5	51.1	52.9	51.9	1.1	52.4	-1.3

Model	Training process				Testing process				Difference between the average mAP values (training - testing)
	Best mAP run 1(%)	Best mAP run 2 (%)	Δ % (1-2)	Average best mAP % (1-2)	mAP run 1 (%)	mAP run 2 (%)	Δ % (1-2)	Average mAP % (1-2)	
22	45.7	47.3	-1.5	46.5	48.4	49.6	-1.2	49.0	-2.5
23	49.1	50.2	-1.1	49.6	51.1	51.3	-0.2	51.2	-1.5
24	44.1	41.0	3.1	42.6	46.6	43.9	2.8	45.3	-2.7
25	51.6	52.3	-0.7	52.0	53.2	53.3	-0.1	53.3	-1.3
26	47.8	44.0	3.8	45.9	50.6	47.3	3.3	48.9	-3.0
27	49.0	52.4	-3.4	50.7	50.3	53.9	-3.6	52.1	-1.4
28	43.6	45.9	-2.4	44.8	45.6	47.3	-1.8	46.5	-1.7
29	52.2	51.8	0.4	52.0	53.4	53.2	0.3	53.3	-1.3
30	44.7	42.5	2.2	43.6	47.9	46.1	1.9	47.0	-3.4
31	47.4	48.7	-1.3	48.1	48.9	49.1	-0.3	49.0	-1.0
32	43.9	41.5	2.4	42.7	46.5	45.6	0.9	46.1	-3.3

Based on Table 4.2, the results of the training process indicate the highest average mAP in two different models: the 25th model with 52% mAP (51.6% best mAP in the 1st run and 52.3% best mAP in the 2nd run) and the 29th model with 52% mAP (52.2% best mAP in the 1st run and 51.8% best mAP in the 2nd run).

Figure 4.9 displays the training progress chart of the 29th experiment (1st run). The x-axis shows the number of iterations as training progresses. The y-axis measures two different values: the value of the loss function, and mAP. The blue curve of the graph represents the training loss curve, and the red one represents the mAP results during training. The training loss curve shows whether the model has adapted well to the dataset, with lower values indicating better results. During training, this curve decreases and then stabilizes with some fluctuation within the interval of (4, 8). The red curve shows the model's ability to detect the specified objects in the input data during validation. The increasing mAP values observed in the red curve, ranging from 10% to 52%, indicate that the model is performing well. However, there are signs of overfitting, notably around the 7000th iteration, where the mAP drops to 18%. Despite this overfitting, the model demonstrates effective object detection and classification capabilities on the modified UAV dataset, achieving a maximum mAP of 52%.

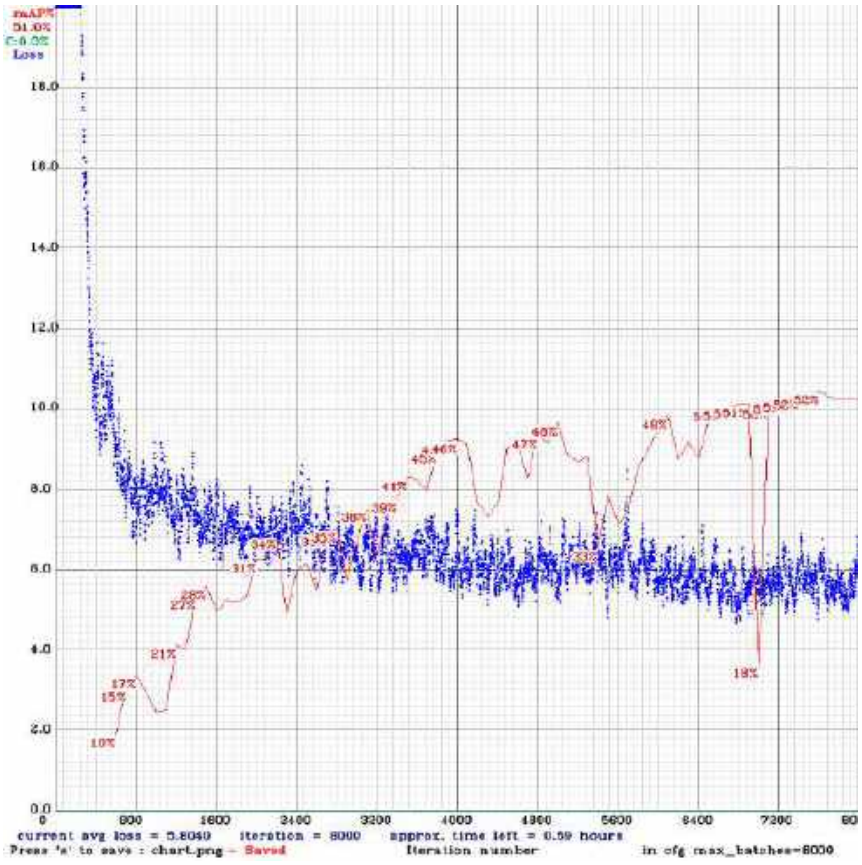

Figure 4.9 *Training progress chart on the modified UAV dataset*

Continuing with the results of Table 4.2, the results of the testing runs indicated that the 25th and 29th models also achieved the highest average mAP. Specifically, the 25th model with 53.3% mAP (53.2% best mAP in the 1st run and 52.3% in the 2nd run) and the 29th model with 53.3% mAP (53.4% best mAP in the 1st run and 52.2% best mAP in the 2nd run).

Based on the above findings, we can draw the following conclusions:

- The 25th and 29th models achieved the highest average mAP results in both the training and testing runs
- There is good agreement between the training/validation vs. testing results; i.e. the highest mAP values in both sets of results correspond to the same hyperparameter combinations. This also holds for the lowest mAP values
- The effects of the hyperparameters on mAP are very significant. Indicatively, the best performing hyperparameter combination has a mAP value of 52%, while the worst performing combination has a mAP value of 39.8%. This indicates the great importance of selecting the appropriate hyperparameters in training, depending on the characteristics of the training dataset
- Conclusions on the effects of the hyperparameters and their combinations on mAP may be drawn from either the validation or the testing map.

Tables 4.3 presents the average precision (AP) results of the four objects under consideration of all models of Table 4.2. For each hyperparameter combination, the Table presents the average APs of the two runs for both validation (first five columns) and testing (five last columns). For instance, according to the first five columns of Table 4.3, model number 20 achieved an average AP of 32.1% for the person class, 36% for the small vehicle class, 45.9% for the large vehicle class, and 48.8% for the ship class in training. The average mAP is $(32.1 + 36 + 45.9 + 48.8)/4 = 40.7\%$.

Table 4.3 *Training and testing results: Class AP values*

Model	Training process					Testing process				
	Average AP values per class %				Average mAP %	Average AP values per class %				Average mAP %
	Person	Small vehicle	Large vehicle	Ship		Person	Small vehicle	Large vehicle	Ship	
1	43.5	40.5	54.9	62.8	50.4	43.3	57.6	54.9	53.8	52.4
2	37.5	38.1	51.4	47.7	43.7	37.3	54.4	50.7	45.5	47.0
3	42.1	40.7	53.3	56.1	48.0	41.5	57.8	54.2	48.4	50.5
4	37.7	37.1	50.2	40.2	41.3	37.5	53.0	49.4	41.4	45.3
5	43.6	40.9	55.1	65.1	51.2	43.1	58.1	55.8	55.8	53.2
6	35.3	37.8	50.3	47.1	42.6	34.7	53.8	49.2	46.2	46.0
7	39.2	40.1	53.6	52.8	46.4	38.9	57.2	53.5	47.5	49.3
8	34.8	37.8	50.9	35.7	39.8	34.7	54.1	50.1	37.1	44.0
9	44.9	40.2	54.4	57.8	49.3	44.5	57.3	55.1	51.3	52.1
10	38.9	38.3	52.3	40.7	42.5	38.2	54.5	51.4	45.0	47.3
11	42.8	40.1	53.8	53.6	47.6	41.7	57.2	53.7	46.8	49.9
12	35.8	37.4	50.4	36.3	40.0	35.1	53.4	49.6	40.7	44.7
13	43.2	40.1	52.9	62.1	49.6	42.5	57.0	53.4	52.3	51.3
14	38.6	38.1	50.9	49.1	44.2	38.2	54.5	50.3	47.7	47.7
15	40.1	40.2	51.8	58.1	47.5	40.0	57.0	53.1	50.2	50.1

Model	Training process					Testing process				
	Average AP values per class %				Average mAP %	Average AP values per class %				Average mAP %
	Person	Small vehicle	Large vehicle	Ship		Person	Small vehicle	Large vehicle	Ship	
16	38.1	37.6	51.7	32.7	40.0	38.0	53.9	51.0	33.9	44.2
17	43.3	41.8	54.1	67.4	51.6	42.4	59.0	54.1	55.5	52.8
18	40.7	38.7	48.5	51.2	44.8	40.8	55.6	48.5	47.9	48.2
19	44.8	42.2	50.8	63.1	50.2	44.4	59.2	49.7	50.8	51.0
20	32.1	36.0	45.9	48.8	40.7	32.3	51.8	45.4	46.8	44.1
21	43.6	41.2	51.9	67.8	51.1	43.1	58.2	51.8	56.5	52.4
22	39.0	39.4	50.8	56.8	46.5	39.2	56.0	50.2	50.7	49.0
23	38.8	40.3	51.2	68.2	49.6	37.9	57.4	52.1	57.2	51.2
24	35.4	37.5	48.1	49.3	42.6	34.9	53.6	47.5	45.0	45.3
25	43.8	42.2	52.6	69.4	52.0	43.5	59.4	54.0	56.1	53.3
26	41.6	39.5	48.8	53.9	45.9	41.1	56.2	48.3	50.2	48.9
27	42.4	41.1	53.8	65.5	50.7	41.5	58.4	54.8	53.8	52.1
28	36.0	38.1	46.1	58.8	44.8	35.2	54.6	45.1	50.9	46.5
29	45.1	42.2	53.8	66.9	52.0	44.7	59.5	53.8	55.1	53.3
30	38.2	38.2	49.6	48.5	43.6	37.6	54.7	49.9	45.8	47.0
31	37.3	41.2	50.4	63.4	48.1	36.6	58.1	50.8	50.5	49.0
32	36.9	37.4	49.5	47.1	42.7	36.4	53.2	48.8	45.9	46.1

As shown in Table 4.3, the highest average AP values achieved during training for each class were mostly produced by different models across both runs. The highest average AP values (in training) for each class were 45.1% (29th model) for person, 42.2% (19th and 29th models) for small vehicle, 55.1% (5th model) for large vehicle, and 69.4% (25th model) for ship. Similarly, the highest AP values in the testing process were also from different models. The highest average AP values (in testing) for each class were 44.7% (29th model) for person, 59.5% (29th model) for small vehicle, 55.8% (5th model) for large vehicle, and 57.2% (23rd model) for ship.

Table 4.4, presents the AP values for the model achieving the highest average mAP value, as well as the models that produced the highest AP values per class in training and testing, respectively. Since two different models achieved the highest average mAP value, we selected the 29th model to study (the one analyzed in Figure 4.9). Then, we calculated the deviation of the best performing model ($\lfloor AP\ of\ the\ best\ model - best\ AP \rfloor / best\ \ AP$) to determine whether the best performing model has a high or low deviation from the best AP values in training or testing. For instance, the deviation of the 29th model for large vehicle class is -2.36%: ($\lfloor 53.8 - 55.1 \rfloor / 55.1$).

Table 4.4 *Average Precision (AP) deviation of the best performing model in training and testing*

		Training process						Testing process					
		Best performing model	Models produced the best AP per class				Deviation w.r.t. the 29th model %	Best performing model	Models produced the best AP per class				Deviation performing w.r.t. the 29th model %
Model		29th	29th	29th	5th	25th		29th	29th	29th	5th	23rd	
Classes	Person	45.1	45.1				0	44.7	44.7				0
	Small vehicle	42.2		42.2			0	59.5		59.5			0
	Large vehicle	53.8			55.1		-2.36%	53.8			55.8		-3.58%
	Ship	66.9				69.4	-3.60%	55.1				57.2	-3.67%

Table 4.4 clearly indicates that the 29th model that corresponds to the highest mAP does have a robust performance in terms of AP for the classes of interest, as evidenced by the very limited deviation from the best AP values.

4.4 Hyperparameter effects on mean Average Precision (mAP)

By applying Analysis of Variance (ANOVA), we can evaluate the effect of each factor (hyperparameter) and of the factor interactions on mAP. This allows us to identify the significant factors (hyperparameters) and the combination(s) of factors that result in optimal training of the YOLOv4-p6 model in terms of mAP.

As explained in Appendix D, the ANOVA tests the following hypotheses:

- The null hypothesis (H_o) states that a factor or a factor interaction has no significant effect on mAP
- The alternative hypothesis (H_a) states that a factor or a factor interaction has a significant effect on mAP.

We performed two ANOVA analyses with the MiniTab software, which utilized the experimental results shown in Table 4.1.

The first ANOVA drill down was conducted using the best mAP results from the training/validation runs. The purpose of this analysis was to identify the effect of the hyperparameters and their combinations in the effectiveness of training. Identifying these effects could assist developers in creating better detection models, reducing the risk of overfitting and improving detection performance.

The second ANOVA drill down was conducted using the best mAP results from the testing runs. The purpose of this analysis was to verify that the hyperparameter combinations resulting in effective training, show consistent superior performance during testing.

The following Table 4.5 provides the "design summary" of both analyses.

Table 4.5 *Full factorial design summary*

Full Factorial Design			
Design Summary			
Factors	5	Base Design	5, 32
Runs	64	Replicates	2
Blocks	1	Center pts (total)	0
All terms are free from aliasing.			

As indicated in Table 4.5, the experimental design encompasses five factors (our selected second set of hyperparameters):

- Image resolution
- Activation function - structure
- Non-Maximum Suppression (NMS)
- Data augmentation technique
- Anchor dimensions.

Within the experimental setup a single block is applied, meaning that all experiments are performed under uniform conditions. This reduces variability. In addition, we selected zero center points before the execution of ANOVA, in order not to perform experimental runs at the mid-level of each factor in the design. Although this limits the test's ability to evaluate nonlinear effects, we assumed that such effects are not present. The base design includes 5 factors, and since each factor has 2 levels, this leads to 32 experimental runs (i.e., $2^5 = 32$). With 2 replicates, the total number of runs increases to 64. Lastly, the "all terms are free from aliasing" means that none of the factors and their interactions are mixed with each other and can be estimated independently.

4.4.1 ANOVA on best mAP results from training runs

The ANOVA analysis of the training (validation) results is presented in Table 4.6. Note the following:

- **Adjusted Sum of Squares** (Adj SS) isolates and measures the variance in mAP that is explained by each factor or interaction after removing the effects of other factors and their interactions in the model. For instance, the Adjusted Sum of Squares for image resolution is 0.075196. This value represents the variance in mAP explained by changes in image resolution (Kutner, 2005).
- **Adjusted Mean Squares** (Adj MS) is the average variance associated with each factor or interaction, accounting for the relevant degrees of freedom in the model. For instance, the Adjusted Mean Square for "image resolution" is 0.075196. This value is calculated by dividing the Adjusted Sum of Squares for "image resolution" by its corresponding degrees of freedom, which in this case is 1 (Kutner, 2005).

The Adjusted Mean Squares is:

$$MS_{adj} = \frac{SS_{adj}}{DF} \tag{4.1}$$

- The **F-value** is the ratio of the Adjusted Mean Square of a factor or interaction to the Adjusted Mean Square of the error (see Equation 4.2). A high F-value (beyond a certain threshold provided in tables based on the DOF, risk, etc.) indicates that the factor or interaction has a statistically significant impact on mAP. For example, “image resolution”, “activation function in the structure” and “anchor dimensions” have high F-values, indicating that they have a statistically significant effect on mAP. In contrast, factors like “NMS” and “data augmentation technique” have lower F-values, indicating that they do not have a significant effect on mAP (Archdeacon, 1994).

  The F-Value is provided from:

$$F - Value = \frac{MS_{adj}}{MS_{adj/error}} \tag{4.2}$$

- The **p-value** represents the probability of obtaining results (e.g. F-value) as extreme as those observed, assuming the null hypothesis is true. A small p-value (less than 0.05) shows that the observed results are statistically significant, leading to the rejection of the null hypothesis (H_o). Conversely, a high p-value suggests that the observed results could reasonably occur by chance, leading to accepting the null hypothesis. For example, “image resolution”, “activation function in the structure” and “anchor dimensions” have low p-values, indicating that they have a significant effect on mAP. In contrast, “NMS” and “data augmentation technique” have a high p-value, indications that they have no significant effect on mAP (Archdeacon, 1994).

At the end of Table 4.6, two additional terms are presented:

- **Error**: represents the unexplained variance in the model, showing the difference between the observed results and the results predicted by the model. In our model, $SS_{error} = 0.00883$ (Adj SS) represents the unexplained variance in the mAP that is not accounted for by the model. It equals to the sum of squares of the differences between each observed result and the result predicted by the model, as presented below (Kutner, 2005):

$$SS_{error} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 \tag{4.3}$$

  where,
  - y_i: is the observed result for the i-th observation
  - $\hat{y}_i$: is the predicted result for the i-th observation
  - n: is the number of observations.

- **Total**: represents the total variance that is separated into explained variance (model) and unexplained variance (error). It provides a baseline against which the model's performance is

evaluated. In our model, $SS_{total} = 0.106443$ represents the total variance in the mAP. It is calculated as the sum of squares of the differences between each observed result and the overall mean of the dependent variable, as presented below (Kutner, 2005):

$$SS_{total} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \quad (4.4)$$

where,

- y_i: is the observed result for the i-th observation
- $\bar{y}$: is the mean of the observed results
- n: is the number of observations.

The small relative value of unexplained variance in the model shows that a large portion of the total variance is accounted for by the variables included in the model. This indicates that the model is effective in explaining the variability in the data, which makes the model valuable for understanding the results we are studying.

Table 4.6 presents the significance of each factor and their interactions on the mAP.

Table 4.6 *ANOVA training results*

Factorial Regression					
Analysis of Variance					
Source	DF	Adj SS	Adj MS	F-Value	p-Value
Model	31	0.097611	0.003149	11.41	0
Linear	5	0.092651	0.01853	67.14	0
Image resolution	1	0.075196	0.075196	272.47	0
Activation function - structure	1	0.010521	0.010521	38.12	0
NMS	1	0.000216	0.000216	0.78	0.383
Data augmentation technique	1	0	0	0	0.979
Anchor dimensions	1	0.006718	0.006718	24.34	0
2-Way Interactions	10	0.001182	0.000118	0.43	0.922
Image resolution*Activation function - structure	1	0.000053	0.000053	0.19	0.663
Image resolution*NMS	1	0.000048	0.000048	0.17	0.679
Image resolution*Data augmentation technique	1	0.000094	0.000094	0.34	0.565
Image resolution*Anchor dimensions	1	0.000029	0.000029	0.11	0.748
Activation function - structure*NMS	1	0.000301	0.000301	1.09	0.304
Activation function - structure*Data augmentation technique	1	0.000187	0.000187	0.68	0.416
Activation function - structure*Anchor dimensions	1	0.000133	0.000133	0.48	0.493
NMS*Data augmentation technique	1	0.000106	0.000106	0.38	0.54

Factorial Regression					
Analysis of Variance					
NMS*Anchor dimensions	1	0.000056	0.000056	0.2	0.655
Data augmentation technique*Anchor dimensions	1	0.000175	0.000175	0.63	0.432
3-Way Interactions	10	0.001765	0.000176	0.64	0.769
Image resolution*Activation function - structure*NMS	1	0.000092	0.000092	0.33	0.568
Image resolution*Activation function - structure*Data augmentation technique	1	0.000071	0.000071	0.26	0.616
Image resolution*Activation function - structure*Anchor dimensions	1	0.000006	0.000006	0.02	0.885
Image resolution*NMS*Data augmentation technique	1	0.000067	0.000067	0.24	0.625
Image resolution*NMS*Anchor dimensions	1	0.000063	0.000063	0.23	0.635
Image resolution*Data augmentation technique*Anchor dimensions	1	0.000006	0.000006	0.02	0.881
Activation function - structure*NMS*Data augmentation technique	1	0.000015	0.000015	0.05	0.817
Activation function - structure*NMS*Anchor dimensions	1	0.000038	0.000038	0.14	0.712
Activation function - structure*Data augmentation technique*Anchor dimensions	1	0.00002	0.00002	0.07	0.788
NMS*Data augmentation technique*Anchor dimensions	1	0.001386	0.001386	5.02	0.032
4-Way Interactions	5	0.001788	0.000358	1.3	0.29
Image resolution*Activation function - structure*NMS*Data augmentation technique	1	0.000004	0.000004	0.01	0.909
Image resolution*Activation function - structure*NMS*Anchor dimensions	1	0.00004	0.00004	0.14	0.706
Image resolution*Activation function - structure*Data augmentation technique*Anchor dimensions	1	0.001102	0.001102	3.99	0.054
Image resolution*NMS*Data augmentation technique*Anchor dimensions	1	0.000575	0.000575	2.08	0.159
Activation function - structure*NMS*Data augmentation technique*Anchor dimensions	1	0.000068	0.000068	0.25	0.623
5-Way Interactions	1	0.000225	0.000225	0.82	0.373
Image resolution*Activation function - structure*NMS*Data augmentation technique*Anchor dimensions	1	0.000225	0.000225	0.82	0.373

Factorial Regression				
Analysis of Variance				
Error	32	0.008831	0.000276	
Total	63	0.106443		

MiniTab also generates convenient graphical representations to help the assessment of the factor/interaction effects. These include:

- Pareto chart of the standardized effects
- Main effects plot for mAP
- Interaction plot for mAP.

Pareto chart of the standardized effects

As shown in Figure 4.10, the Pareto chart of the standardized effects illustrates the impact of various factors and interactions on the mAP. The factors are labeled A through E, representing the following variables:

- A: Image resolution
- B: Activation function in the structure
- C: NMS (Non-Maximum Suppression)
- D: Data augmentation technique
- E: Anchor dimensions.

The red dashed line at the standardized effect value of 2.04 shows the significance threshold at $a = 0.05$. Factors with bars extending beyond this red dashed line have a significant impact on mAP. However, this also means that there is a 5% ($a = 0.05$) risk of identifying a factor or an interaction of factors as significant while it is not.

The significant factors and combination of factors for training according to the Pareto chart are the following:

- Image resolution (A) has the greatest standardized effect on mAP
- Activation function in the structure (B)
- Anchor dimensions (E)
- The combination of factors CDE (NMS, data augmentation technique and anchor dimensions).

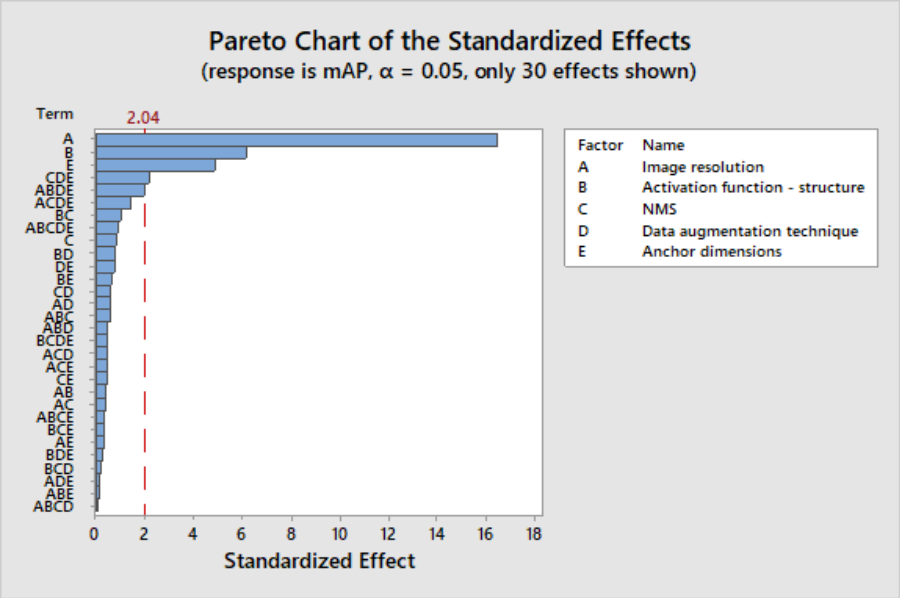

Figure 4.10 *First Pareto chart of the standardized effects*

The remaining hyperparameters or their combinations do not have a significant effect on mAP, since their values are less than 2.04 (Navarro Tuch et al., 2019).

Commenting on the above results:

- "Image resolution" changes the resolution of the input data, which greatly influences the training process. Larger image resolutions provide better detection and classification results
- "Activation function in the structure" is significant for feature extraction from the input data. Newer versions of activation functions offer better non-linear characteristics, which help to capture more details (information) from the available images
- "Anchor dimensions" assist the model to generate more appropriate bounding boxes by adjusting the size of the anchor bounding boxes during the training process. The most suitable configuration of anchor box dimensions, in relation to the dataset that is used to train the model, is likely to result in improved model performance.

Main effects plot for mAP

Figure 4.11 presents the main effects plot for training (Kim et al., 2007) that quantifies the effect in mAP for factors A, B, C, D and E, respectively. The following observations can be made regarding these five factors:

- Varying image resolution from 969x960 to 1280x1280 improves mAP by $|42.9\% - 49.7\%| = 6.8\%$
- Varying activation function from Swish to Mish improves mAP by $|45\% - 47.5\%| = 2.5\%$
- Varying anchor box sizes from the original set to new set improves mAP by $|45.2\% - 47.3\%| = 2.1\%$.

Therefore, the best options for our significant factors are:

- 1280x1280 for image resolution
- Mish function for the activation function in the structure
- New set of anchor box sizes for anchor dimensions.

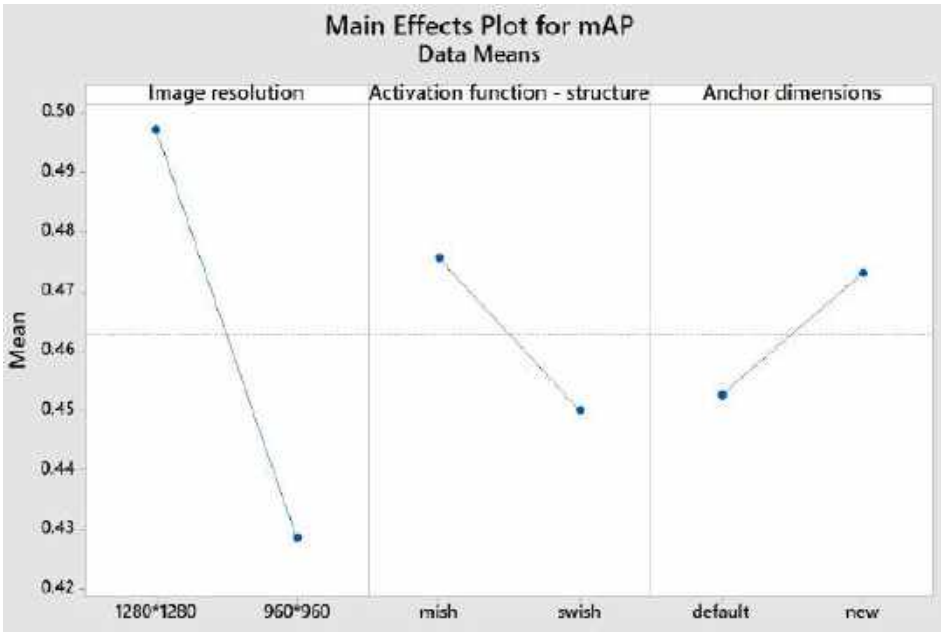

Figure 4.11 *Main effects plot for mAP (training) of A, B and E factors*

Interaction plot for CDE

Figure 4.12 presents an interaction plot (Ibrahim et al., 2011) showing the only significant three-way interaction CDE (involving C: NMS, D: Data augmentation techniques, and E: Anchor dimensions) emerging from the training analysis. Note that the interaction plots in MiniTab display the interaction between only two factors per plot. As a result, our three-way interaction is analyzed in three different two-way interaction plots.

Overall, the interaction plot shows that the combination of CDE factors yields better mAP results with:

- Distance-Intersection over Union - Non-Maximum Suppression (DIoU-NMS) for NMS
- Mosaic + mixup for data augmentation
- New set of anchor box sizes for anchor dimensions.

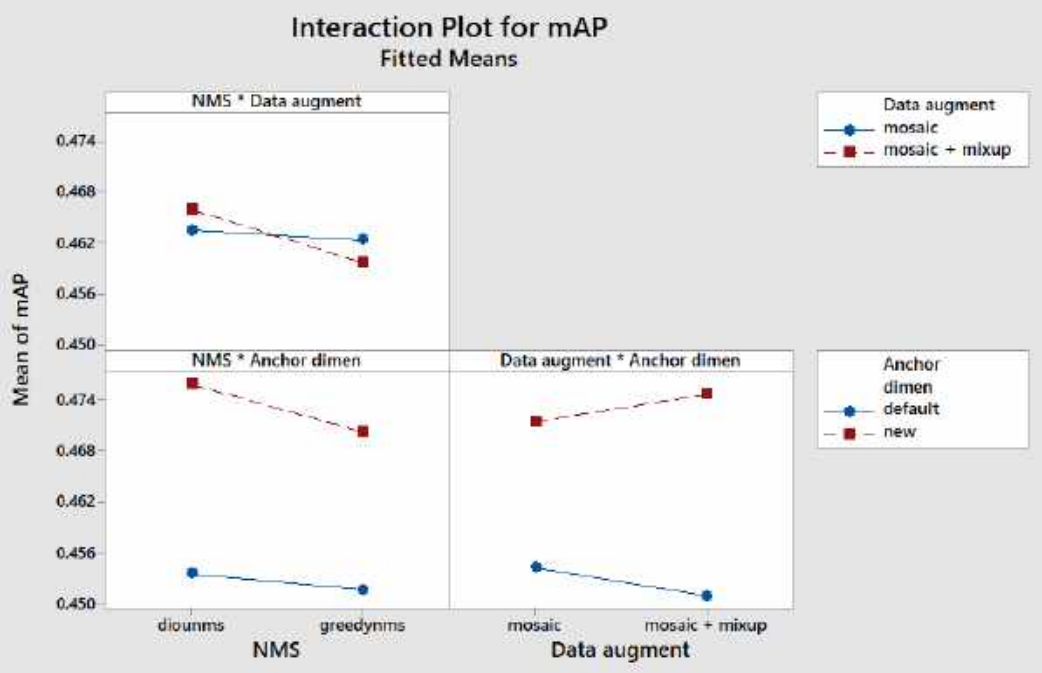

Figure 4.12 *Interaction plot for CDE*

4.4.2 ANOVA on best mAP results from testing runs

We created a similar Table (see Table 4.7) for the ANOVA results of the testing runs.

Table 4.7 *ANOVA testing results*

Factorial Regression					
Analysis of Variance					
Source	DF	Adj SS	Adj MS	F-Value	p-Value
Model	31	0.056872	0.001835	7.28	0
Linear	5	0.053675	0.010735	42.59	0
Image resolution	1	0.04256	0.04256	168.85	0
Activation function - structure	1	0.009347	0.009347	37.08	0
NMS	1	0.000303	0.000303	1.2	0.281
Data augmentation technique	1	0.000019	0.000019	0.07	0.787
Anchor dimensions	1	0.001446	0.001446	5.74	0.023
2-Way Interactions	10	0.000594	0.000059	0.24	0.99
Image resolution*Activation function - structure	1	0.000064	0.000064	0.25	0.618
Image resolution*NMS	1	0.000014	0.000014	0.06	0.812
Image resolution*Data augmentation technique	1	0.00017	0.00017	0.68	0.417
Image resolution*Anchor dimensions	1	0.00004	0.00004	0.16	0.694
Activation function - structure*NMS	1	0.000058	0.000058	0.23	0.634
Activation function - structure*Data augmentation technique	1	0.000028	0.000028	0.11	0.741

Factorial Regression					
Analysis of Variance					
Activation function - structure*Anchor dimensions	1	0.000004	0.000004	0.01	0.905
NMS*Data augmentation technique	1	0.000167	0.000167	0.66	0.421
NMS*Anchor dimensions	1	0	0	0	0.966
Data augmentation technique*Anchor dimensions	1	0.000048	0.000048	0.19	0.667
3-Way Interactions	10	0.000916	0.000092	0.36	0.954
Image resolution*Activation function - structure*NMS	1	0.00011	0.00011	0.44	0.513
Image resolution*Activation function - structure*Data augmentation technique	1	0.000026	0.000026	0.1	0.749
Image resolution*Activation function - structure*Anchor dimensions	1	0.000043	0.000043	0.17	0.682
Image resolution*NMS*Data augmentation technique	1	0.000004	0.000004	0.02	0.897
Image resolution*NMS*Anchor dimensions	1	0.000127	0.000127	0.5	0.484
Image resolution*Data augmentation technique*Anchor dimensions	1	0.000027	0.000027	0.11	0.744
Activation function - structure*NMS*Data augmentation technique	1	0	0	0	0.997
Activation function - structure*NMS*Anchor dimensions	1	0.000013	0.000013	0.05	0.819
Activation function - structure*Data augmentation technique*Anchor dimensions	1	0.000024	0.000024	0.1	0.76
NMS*Data augmentation technique*Anchor dimensions	1	0.000541	0.000541	2.14	0.153
4-Way Interactions	5	0.001249	0.00025	0.99	0.439
Image resolution*Activation function - structure*NMS*Data augmentation technique	1	0.000012	0.000012	0.05	0.83
Image resolution*Activation function - structure*NMS*Anchor dimensions	1	0.000146	0.000146	0.58	0.452
Image resolution*Activation function - structure*Data augmentation technique*Anchor dimensions	1	0.000857	0.000857	3.4	0.074
Image resolution*NMS*Data augmentation technique*Anchor dimensions	1	0.000092	0.000092	0.37	0.549
Activation function - structure*NMS*Data augmentation technique*Anchor dimensions	1	0.000141	0.000141	0.56	0.459
5-Way Interactions	1	0.000438	0.000438	1.74	0.197

Factorial Regression					
Analysis of Variance					
Image resolution*Activation function - structure*NMS*Data augmentation technique*Anchor dimensions	1	0.000438	0.000438	1.74	0.197
Error	32	0.008066	0.000252		
Total	63	0.064938			

Pareto chart of standardized effects

In the Pareto chart of the standardized effects, shown in Figure 4.13, the labeling of the factors is consistent with the analysis of Subsection 4.4.2. Similarly to the first Pareto chart, the factors are labeled A through E, representing the following variables:

- A: Image resolution
- B: Activation function in the structure
- C: NMS (Non-Maximum Suppression)
- D: Data augmentation technique
- E: Anchor dimensions.

In this chart the red dashed line representing the standardized effect has a value of 2.04, indicating the significance threshold at α=0.05. Factors with bars extending beyond this red dashed line have a significant impact on mAP. Therefore, the significant factors for testing according to the Pareto chart are the following:

- Image resolution (A) has the greatest standardized effect on mAP
- Activation function in the structure (B)
- Anchor dimensions (E).

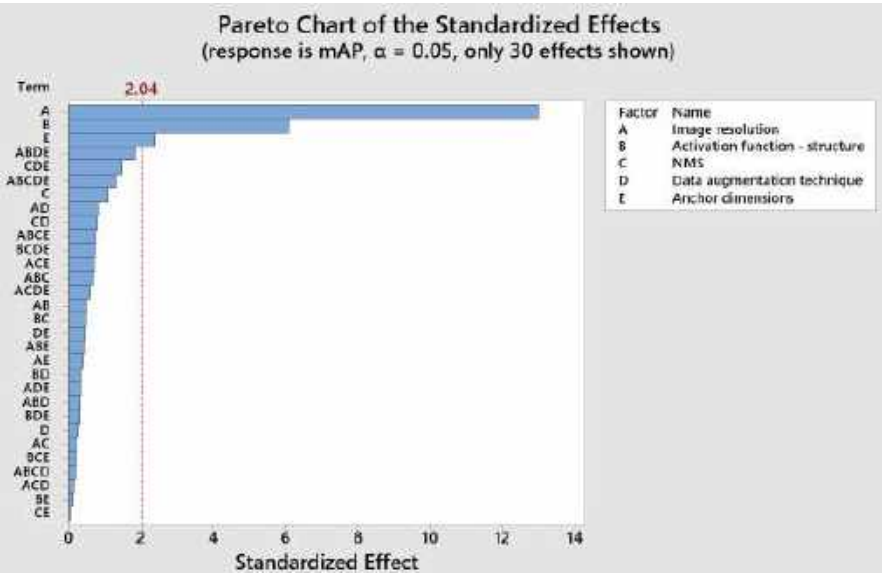

Figure 4.13 *Second Pareto chart of the standardized effects*

The remaining hyperparameters or their combinations do not have a significant effect on mAP, since their values are less than 2.04. Comparing the ANOVA results of training/validation vs. testing, we may conclude that

1. The effects of the three factors, image resolution (factor A), activation function in the structure (factor B) and anchor dimensions (factor E), are significant in both cases
2. The three-way interaction CDE is not significant in the testing ANOVA; however, it still has a relatively high standardized effect close to the threshold of Fig. 4.13

These conclusions are very important; they indicate that a model trained under optimal hyperparameter settings, has also superior performance when tested on data that the model has not been exposed to (although from the same dataset that all three, training, validation and testing sets have been created).

Main effects plot for mAP (testing)

Figure 4.14 presents a main effects plot for testing (Kim et al., 2007) that quantifies the effect in mAP for factors A, B, C, D and E, respectively. The following observations can be made regarding these five factors:

- Varying image resolution from 969x960 to 1280x1280 improves mAP by $|46.3\% - 51.4\%| = 5.1\%$
- Varying activation function from Swish to Mish improves mAP by $|47.6\% - 50.1\%| = 2.5\%$
- Varying anchor box sizes from the original set to new set improves mAP by $|48.4\% - 49.3\%| = 0.9\%$.

The effects of image resolution (A) and activation function (B) on mAP resulting from the testing experiments are very close to those resulting from the training/validation experiments. The effect of anchor size dimensions (E), although significant, has a lower value from the one observed from the latter experiments. This consistency between the two sets of experiments is quite encouraging, indicating that the effects of the hyperparameters on the performance of the trained model has been identified correctly and are significant.

The best options for our significant factors are:

- 1280x1280 for image resolution
- Mish function for the activation function in the structure
- New set of anchor box sizes for anchor dimensions.

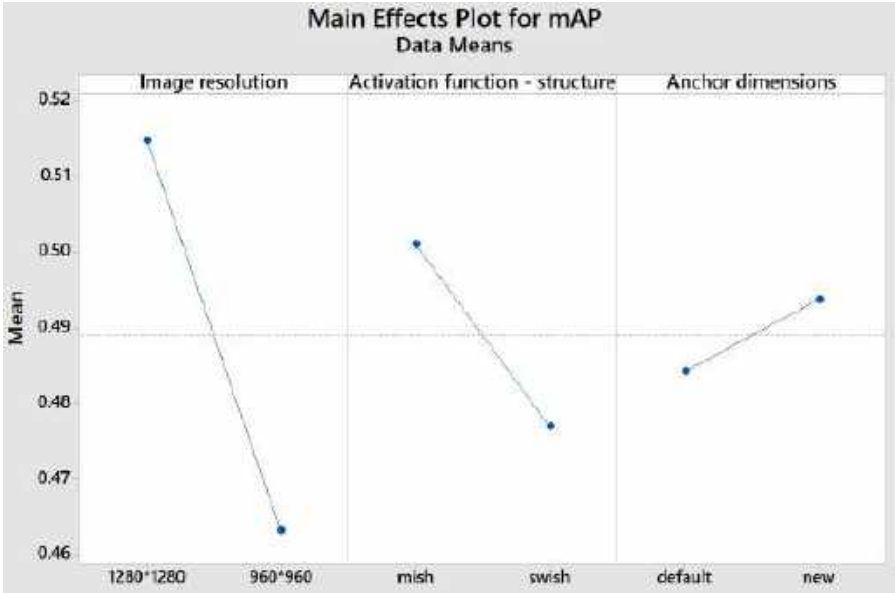

*Figure **4.14** Main effects plot for mAP (testing) of A, B and E factors*

4.5 Concluding remarks

After completing both ANOVA analyses, the results from training and testing runs agreed to a high degree.

From training, three hyperparameters (factors) and one hyperparameter three-way interaction have been found to affect mAP with statistical significance. The main effects, including image resolution (factor A), activation function in the structure (factor B), and anchor dimensions (factor E), are shown to affect mAP by 6.8%, 2.5%, and 2.1%, respectively. The significant hyperparameter CDE three-way interaction involves NMS, the data augmentation technique, and anchor dimensions. It positively impacts mAP when using the combination of DIoU-NMS, Mosaic + Mixup, and the new set of anchor box sizes for anchor dimensions.

From testing, the same three hyperparameters as in training have been found to affect mAP with statistical significance. The main effects, including image resolution (factor A), activation function in the structure (factor B), and anchor dimensions (factor E), are shown to affect mAP, increasing it by 5.1%, 2.5%, and 0.9%, respectively.

As mentioned in Section 4.4, based on the comparison of the ANOVA results for training/validation versus testing, the following findings can be reported:

1. **The same factors are significant**: Both the training/validation and testing ANOVAs reveal image resolution (factor A), activation function (factor B), and anchor dimensions (factor E) as significant factors
2. **The effects are similar**: The magnitude of the effects observed in the training/validation ANOVA are comparable to those found in the testing ANOVA, with a similar pattern of significance across the factors.

Note that the results achieved by a model in an application depends on the fit of the trained model to the application dataset. In Chapter 5, we use our trained models on our lab's dataset, referred to as the DeOPSys dataset, to assess the performance of the trained models on this independent UAV dataset that has quite different characteristics than the modified UAV dataset we used for model training.

Chapter 5 Testing the trained YOLOv4-p6 models on DeOPSys dataset

In this Chapter, we present the evaluation of the proposed hyperparameter optimization method using the best trained models (25th and 29th models) to detect objects from UAV images obtained by our lab (DeOPSys dataset). The objective of this exercise is to validate that the best performing models in the training/validation process are performing well in the DeOPSys dataset.

5.1 DeOPSys UAV dataset

The DeOPSys dataset (ENIRISST+, 2023) includes annotated images containing the following three classes:

- Person
- Car
- Boat

The images were captured by a drone in two locations on the island of Chios: the port/shipyard in the Tholos area (see example (a) in Figure 5.1) and a rural area near Lagada (see example (b) in Figure 5.1). These images were taken at different altitudes (15m, 30m, and 50m) and at different times of the day (morning and evening). Note that:

- Images taken from different heights offered varying levels of detail and viewpoints. Lower altitudes result in more detailed views of objects, but they cover a smaller area, while images taken from higher altitudes covered a wider area with less detail
- Daylight images generally had higher quality due to better natural light, while images taken in low light conditions were less clear, making it harder to decipher some object details. In addition, weather conditions, like cloudy or clear skies, also impacted the image quality.

Figure 5.1 *Sample images from the DeOPSys dataset featuring (a) the Tholos Port (daylight conditions) and (b) Lagada Rural Area (low light conditions)*

Furthermore,

- The objects of interest varied in size
- There were variations in the spatial placement of objects within the environment and the characteristics of the environment itself. For example, in some cases the dataset involved two persons and a car.

Finally, the dataset included 722 images with multiple annotated objects for detection, captured at various heights. Table 5.1 shows the objects in the images of the DeOPSys dataset's testing sets.

Table 5.1 *The labelled objects of DeOPSys dataset*

		Number of objects			
DeOPSys dataset	**Number of images**	**Number of persons**	**Number of cars**	**Number of boats**	**Total Number of objects**
Images 15m	241	244	229	0	473
Images 30m	240	256	117	62	435
Images 50m	241	257	123	476	856
Morning images	350	373	285	538	1196
Evening images	372	384	184	0	568
Total images	722	757	469	538	1764

The testing sets of Table 5.1 are the following:

1. Images taken from a drone at a height of 15 meters during the morning and evening hours
2. Images taken from a drone at a height of 30 meters during the morning and evening hours
3. Images taken from a drone at a height of 50 meters during the morning and evening hours
4. Images taken from a drone at heights of 15, 30 and 50 meters during the morning hours
5. Images taken from a drone at heights of 15, 30 and 50 meters during the evening hours
6. Images taken from a drone at heights of 15, 30 and 50 meters during the morning and evening hours.

5.2 Testing execution on DeOPSys dataset

Referring to Section 4.3, the 25th and 29th models showed the highest average mAP during training, achieving 52% mAP value. For this reason, we tested the selected models on the DeOPSys dataset. The execution of the testing process is similar to the one discussed in Section 4.2, where we kept the .cfg files unchanged and only modified the information included in the .data files. However, DeOPSys contains six different testing sets, as explained at the end of Section 5.1. This means that we have six .data files for each .cfg file, that is, we test the six testing sets with these models (see Figure 5.3). Furthermore, we executed the testing process in DeOPSys twice, just like the previous training and testing on the modified UAV dataset.

By way of example, the first two .data files contain the information shown in Figure 5.2.

.data file of the first testing set

```
1  classes= 4
2  valid  = /media/deopsys/Hard_Disk/drone_files/all_experiments_1.txt
3  names = /home/deopsys/Documents/darknet/data/yolov4_4_classes.names
```

.data file of the second testing set

```
1  classes= 4
2  valid  = /media/deopsys/Hard_Disk/drone_files/all_experiments_2.txt
3  names = /home/deopsys/Documents/darknet/data/yolov4_4_classes.names
```

Figure 5.2 *Data files of the first and second testing sets*

All six .data files retain the same "classes" and "names" parameters used in training or previous testing of the modified UAV dataset used for training. The "valid" parameter changes for every .data file, as presented in Figure 5.2 "valid = … /all_experiement_ [number 1 to 6].txt", to represent the testing set that is going to be used for testing the respective model (.cfg file). Similarly to our previous testing, the "train" and "backup" parameters were omitted in the .data files for testing, as they are only used for training.

To perform the testing process on DeOPSys, we created two folders, named Model_25 and Model_29, each containing one .cfg file with its six corresponding .data files used for testing the respective model. Figure 5.3 represents the folders and the first folder (Model_25), which includes the .cfg file of the 25th model and its six corresponding .data files.

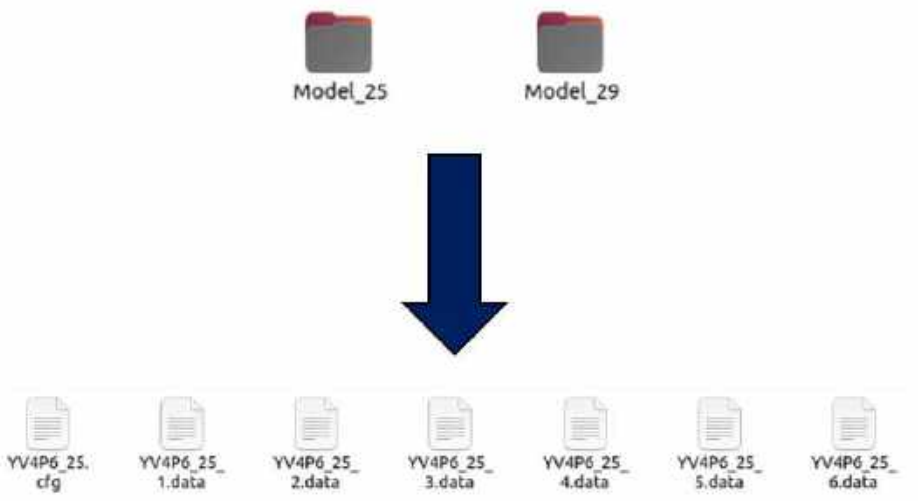

Figure 5.3 *Setup for testing the 25th and 29th models on DeOPSys dataset*

The names of the .cfg files are the same as the .data files, e.g., YV4P6_25. However, the .data files have an extra number at the end to represent the number of the testing set. For example, YV4P6_25_3 specifies the settings and paths for the 25th model's testing process, including the location of the third testing set. After this setup, we executed the command of Figure 5.4 to begin the testing process on DeOPSys dataset. The command in Figure 5.4 is different from the one used in Section 4.2, because this time we have six .data files for each .cfg file.

```
for i in 25 29; do > /media/deopsys/Hard_Disk/panos/testing/testing_${i}.txt; for j in {1..6}; do ./darknet
detector map cfg/cfg_panos/Testing/Model_${i}/YV4P6_${i}_${j}.data
cfg/cfg_panos/Testing/Model_${i}/YV4P6_${i}.cfg
/media/deopsys/Hard_Disk/panos/Testing_process/Weights/First_run/4V4P6_${i}_best.weights -points 101
-thresh 0.25 -iou_thresh 0.5 >> /media/deopsys/Hard_Disk/panos/testing/testing_${i}.txt; done; done
```

Figure 5.4 *Execution command for testing the experiments*

It contains a double loop: an outer loop explicitly targeting only folders Model_25 and Model_29 and an inner loop iterating through the six .data files for each folder. The outer loop selects the corresponding .cfg file and the best weights file created from the first training run of the selected folder. The inner loop then selects the first .data file and tests the respective .cfg file (included in the folder). Once testing of the model with the first testing set (.data file) is complete, it moves to the second .data file, continuing until the model is tested with all six .data files.

After testing the first folder, the process moves to the second folder with its corresponding best weights file, saving the terminal output for each .data file in a single line report under a unique file name to avoid overwriting the results of the previous model. This process continues until the two models have been tested using their corresponding best weights. Once completed, the same command is executed again using the best weight files from the second training run. As a result, from the testing process on the DeOPSys dataset, we obtain line reports similar to our previous testing, containing the mAP metric for each model of each run, which will be evaluated in the next Section.

In summary, to perform the DeOPSys testing process, we used the six testing sets, our trained models, and the best weights (which achieved the highest mAP in validation) generated from the training runs.

5.3 Testing results of DeOPSys dataset

For the DeOPSys dataset, we set up our models to detect only the person and small vehicle classes. Although the DeOPSys dataset includes the class small boat, the models have not been trained in this third class. Note that the ship class included in training has quite different characteristics than the small boat class.

Table 5.2 presents the mAP results of the DeOPSys tests using the 25th and 29th models (rows of the table). The columns correspond to: a) three testing sets (three columns) containing images photographed from 15, 30 and 50 m height, b) the next two columns correspond to images photographed from all heights (15,

30, 50 m) corresponding to morning and evening conditions, respectively. The last column corresponds to all images. Note that for each column, the average mAP of the two models corresponding to each of the 25 and 29 training runs are presented.

From Table 5.2 it is clear that both models yielded very encouraging mAP results for the images taken at heights of 15, 30, and 50 meters during the morning and evening hours (as mentioned in Section 5.1), achieving average mAP of 77.6% and 76.3%, respectively. Furthermore, the two models

- Achieved the highest average mAP on images captured at a height of 15 meters during the morning and evening hours, with the 25th model achieving 85.3% mAP and the 29th model achieving 82.8% mAP.
- Conversely, they achieved the lowest average mAP on images captured at a height of 30 meters during the morning and evening hours, with the 25th model achieving 70.8% mAP and the 29th model achieving 72.0% mAP.

In general:

- Both trained models displayed exceptional performance
- Morning and evening images taken at a lower height are the easier to detect by both models
- Morning and evening images taken at a 30m height are the harder to detect.

Table 5.2 *Average mAP testing results of 25thand 29thmodels.*

Model	Average mAP (%)					
	Images 15m	Images 30m	Images 50m	Morning images	Evening images	All images
25	85.3	70.8	75.4	82.4	74.3	77.6
29	82.8	72.0	74.8	80.4	74.5	76.3

Table 5.3 presents the average AP (Average Precision) values of the two runs for the person and small vehicle classes obtained from testing the 25th and 29th models on the six testing sets from DeOPSys. The rows of the table represent the testing sets (same as in Table 5.2), and the columns correspond to: a) the average AP values of the two runs for the person and small vehicle classes on each testing set separately (two columns), and b) the average mAP results of the models on each testing set (one column). As described in Section 2.6, the AP values are used to calculate the mAP metric. Likewise, utilizing the average AP values, we can calculate the average mAP results. For instance, the average mAP of the 25th model on the images 50m testing set is calculated as: $(51.3 + 99.6/2) = 75.4\%$.

Table 5.3 *AP testing results of 25th and 29th models.*

Model	Testing sets	Average AP values per class %		Average mAP %
		Person	Small vehicle	
Model 25	Images 15m	70.7	99.9	85.3
	Images 30m	49.5	92.1	70.8
	Images 50m	51.3	99.6	75.4
	Morning images	65.4	99.3	82.4

Model	Testing sets	Average AP values per class %		Average mAP %
		Person	Small vehicle	
	Evening images	53.1	95.5	74.3
	All images	57.4	97.7	77.6
Model 29	Images 15m	65.7	100.0	82.8
	Images 30m	47.7	96.3	72.0
	Images 50m	49.6	100.0	74.8
	Morning images	61.4	99.4	80.4
	Evening images	50.8	98.2	74.5
	All images	54.2	98.3	76.3

From Table 5.3, we conclude three primary findings:

- The person class confirms that the models have limited performance with images captured at a height of 30 meters, while they perform better using images captured at heights of 15 and 50 meters. The highest AP value for the person class is 70.7% (images 15m testing set), and the lowest is 47.7% (images 30m testing set)
- The small vehicle class produces great results even from images captured at a height of 30 meters. The highest AP value for the small vehicle class is 100% (images 15m testing set and images 50m testing set), and the lowest is 92.1% (images 30m testing set)
- The models achieved good average AP results on the testing set containing all images. Specifically, the 25th model produced average AP results of 57.4% for the person class and 97.7% for the small vehicle class, while the 29th model produced average AP results of 54.2% for the person class and 98.3% for the small vehicle class. These AP values of the person class are significantly lower than those of the small vehicle class. The reason is that the person images correspond to a much smaller number of pixels in each photo of the datasets as compared to the vehicle class and, thus, the identification/detection uncertainty is higher in the former class.

Chapter 6 Conclusions

This thesis proposes a new method to optimize training of models that may effectively detect and classify objects in real time from images taken at logistics facilities, including ports. These objects include "persons", "small vehicles", "large vehicles", and "ships". The proposed approach is based on YOLO, an advanced image detection system. We reviewed several versions of this system YOLOv1, YOLOv2, YOLOv3, YOLOv4, and Scaled YOLOv4 models, and compared their performance based on the Average Precision (AP) metric. Evidence indicates that YOLOv4-p6 achieves better results on the COCO dataset compared to its previous versions. Therefore, this study focused on optimizing training of the YOLOv4-p6 model to achieve the highest attainable mean AP (mAP) results. The mAP metric was selected as our primary evaluation metric for training and testing. This metric measures the model's performance across multiple classes.

For our study, we generated a training dataset used by combining the following publicly available UAV image datasets that included the selected classes: Aerial vehicle, DOTA, VisDrone-DET, Stanford drone, and DAC-SDC. Prior to combining the datasets, the following modifications were made: 1) annotations were converted into YOLO format, 2) unnecessary classes were removed, and 3) the classes under interest were adjusted in the following order: person, small vehicle, large vehicle, and ship. The resulting combined UAV dataset consists of 76,872 images with 876,388 annotations across the four selected classes. The dataset was divided into three different sets: training (80% of the dataset), validation (10%), and testing (10%), which correspond to 61,429 images with 665,506 annotations, 7,678 images with 105,998 annotations, and 7,680 images with 101,830 annotations, respectively.

Training optimization was studied by tuning the training hyperparameters of YOLO. Our goal was to determine the most favorable set of hyperparameters that maximize the mAP results. To do so, we

1. Split the available training hyperparameters of YOLOv4-p6 into two sets. The first set included hyperparameters adjusted based on the characteristics of the training set; i.e., the number of classes, max batches, steps, and filters (of the convolutional layer before each detection head). We maintained the values/levels of these hyperparameters fixed throughout the experiments. The hyperparameters of the second set were tuned through the experimental work. The hyperparameters in this second set were varied at two levels (1^{st} level: default hyperparameter values of YOLOv4-p6 and 2^{nd} level: new hyperparameter values)
2. Utilized the second set of hyperparameters to generate our experiments design using a Full-Factorial approach. The five hyperparameters under study and tuning resulted in thirty-two (2^5) different factor combinations.

Model training was performed under these 32 hyperparameter combinations, thus generating 32 trained models. (Actually, since for analysis purposes, we performed two training rounds for each combination, the total number of models were 32x2=64.) Note that in each training session, every 100 training iterations validation is conducted. At the end of the session, testing of the trained model is performed.

The outputs of each of the 64 experiments were the highest mAP achieved in validation, and the mAP value resulting from training. The outputs of the experiments were analyzed using ANOVA. This analysis

of the validation results revealed that the hyperparameters impacting mAP in a statistically significant way are: image resolution, activation function, and anchor box sizes. Non-Maximum Suppression (NMS) methods and data augmentation techniques did not have a significant effect on mAP. Similar outcomes were obtained from analyzing the results obtained from the testing process. Furthermore, the analysis indicated a single significant three-way interaction (only in training) between NMS, data augmentation techniques and anchor dimensions. The quantitative effects on mAP of the three significant hyperparameters are as follows:

- Varying image resolution from 969x960 to 1280x1280 improves mAP by 6.8% in training and 5.1% in testing
- Varying activation function from Swish to Mish improves mAP by 2.5% in training and testing
- Varying anchor box sizes from the original set to new set improves mAP by 2.1% in training and 0.9% in testing.

The best trained models (corresponding to the 25th and 29th hyperparameter combinations achieved in validation a 52% average mAP and 53.3% in testing.

The considerable effects of the hyperparameters on model effectiveness support strongly our thesis that careful tuning of the hyperparameters during training may yield to major improvements in model effectiveness. The results from testing the two best trained models on the independent DeOPSys dataset validate the above thesis.

During this completely independent testing process, which used images never employed during training, both models achieved superior object detection results. The highest average mAP values were achieved for images taken from a height of 15m: mAP of 85.3% for the 25th model and 82.8% for the 29th model. Even the lowest model performance, corresponding to images taken from the 30m height, was very good with mAP values of 70.8% for the 25th model and 72.0% for the 29th model.

This exercise showed that the proposed method for tuning the training hyperparameters of YOLO is successful.

- It illustrated the significance of appropriate hyperparameter tuning
- It revealed the effects of the various hyperparameters on the significant mAP output
- It paves the way towards a systematic method for successful hyperparameter tuning to optimize model training.

Future directions for research include:

- Consider the hyperparameters of newer versions of YOLO
- Utilize more efficient and accurate backbone and neck components from updated algorithms on Scaled YOLOv4 models
- Incorporate updated modules; e.g. SPFF (Spatial Pyramid Pooling Fast) module used in YOLOv11
- Develop efficient methods for hyperparameter tuning.

References

Afif, M., Ayachi, R., Pissaloux, E., Said, Y., Atri, M., 2020. Indoor objects detection and recognition for an ICT mobility assistance of visually impaired people. Multimedia Tools and Applications 79. https://doi.org/10.1007/s11042-020-09662-3

Anwar, A., 2022. What is Average Precision in Object Detection & Localization Algorithms and how to calculate it? [WWW Document]. Medium. URL https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b (accessed 2.26.24).

Archdeacon, T.J., 1994. Correlation and regression analysis: a historian's guide. University of Wisconsin Press, Madison, Wis.

Bochkovskiy, A., 2021a. Scaled YOLO v4 is the best neural network for object detection on MS COCO dataset. Medium. URL https://alexeyab84.medium.com/scaled-yolo-v4-is-the-best-neural-network-for-object-detection-on-ms-coco-dataset-39dfa22fa982 (accessed 7.10.24).

Bochkovskiy, A., 2021b. Scaled YOLO v4 is the best neural network for object detection on MS COCO dataset. Medium. URL https://alexeyab84.medium.com/scaled-yolo-v4-is-the-best-neural-network-for-object-detection-on-ms-coco-dataset-39dfa22fa982 (accessed 3.5.24).

Bochkovskiy, A., 2021c. Scaled YOLO v4 is the best neural network for object detection on MS COCO dataset. Medium. URL https://alexeyab84.medium.com/scaled-yolo-v4-is-the-best-neural-network-for-object-detection-on-ms-coco-dataset-39dfa22fa982 (accessed 1.30.24).

Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y.M., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection.

Brownlee, J., 2019. A Gentle Introduction to 1x1 Convolutions to Manage Model Complexity. MachineLearningMastery.com. URL https://machinelearningmastery.com/introduction-to-1x1-convolutions-to-reduce-the-complexity-of-convolutional-neural-networks/ (accessed 2.20.24).

Chen, C., Zeng, W., Zhang, X., 2023. HFPNet: Super Feature Aggregation Pyramid Network for Maritime Remote Sensing Small Object Detection. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing PP, 1–17. https://doi.org/10.1109/JSTARS.2023.3286483

COCO, 2017. Papers with Code - UCSD Ped2 Dataset [WWW Document]. URL https://paperswithcode.com/dataset/ucsd (accessed 6.27.24).

Curti, N., 2020. Route Layer [WWW Document]. NumPyNet. URL https://nico-curti.github.io/NumPyNet/NumPyNet/layers/route_layer.html (accessed 2.20.24).

DataRobot, 2018. Introduction to Loss Functions [WWW Document]. DataRobot AI Platform. URL https://www.datarobot.com/blog/introduction-to-loss-functions/ (accessed 2.16.24).

Devansh, 2023. How does Batch Size impact your model learning. Geek Culture. URL https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa (accessed 7.18.24).

Du, X., Lin, T.-Y., Jin, P., Ghiasi, G., Tan, M., Cui, Y., Le, Q.V., Song, X., 2020. SpineNet: Learning Scale-Permuted Backbone for Recognition and Localization.

Dupont, M., 2023. Role of Tight Bounding Boxes in Enhancing Model Accuracy [WWW Document]. Labelvisor. URL https://www.labelvisor.com/role-of-tight-bounding-boxes-in-enhancing-model-accuracy/ (accessed 4.13.24).

El Aidouni, M., 2019. Understanding YOLO and YOLOv2 [WWW Document]. Manal El Aidouni. URL https://manalelaidouni.github.io/manalelaidouni.github.io/Understanding%20YOLO%20and%20YOLOv2.html (accessed 1.30.24).

ENIRISST+, 2023. Dataset - DeOPSys Dataset - DKAN Demo [WWW Document]. URL https://open-data.enirisstplus.uop.gr/dataset/d8d4163e-fac6-4369-87b8-c795d76fc274 (accessed 5.26.24).

Everingham, M., Van Gool, L., Williams, C., Winn, J., Zisserman, A., 2010. The Pascal Visual Object Classes (VOC) challenge. International Journal of Computer Vision 88, 303–338. https://doi.org/10.1007/s11263-009-0275-4

Gad, A.F., 2020. Accuracy, Precision, and Recall in Deep Learning [WWW Document]. Paperspace Blog. URL https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/ (accessed 2.26.24).

GeeksforGeeks, 2023. Epoch in Machine Learning [WWW Document]. GeeksforGeeks. URL https://www.geeksforgeeks.org/epoch-in-machine-learning/ (accessed 8.22.24).

GeeksforGeeks, 2021. Spatial Resolution (down sampling and up sampling) in image processing. GeeksforGeeks. URL https://www.geeksforgeeks.org/spatial-resolution-down-sampling-and-up-sampling-in-image-processing/ (accessed 3.10.24).

GeeksforGeeks, 2019. CNN | Introduction to Padding [WWW Document]. GeeksforGeeks. URL https://www.geeksforgeeks.org/cnn-introduction-to-padding/ (accessed 7.4.24).

Ghiasi, G., Lin, T.-Y., Le, Q.V., 2018. DropBlock: A regularization method for convolutional networks.

Ghosh, M., Sk, O., Gherardini, F., Zdimalova, M., 2021. Classification of Geometric Forms in Mosaics Using Deep Neural Network. Journal of Imaging 7, 149. https://doi.org/10.3390/jimaging7080149

GitHub, 2008. GitHub: Let's build from here [WWW Document]. GitHub. URL https://github.com/ (accessed 4.25.24).

GitLab, 2011. The most-comprehensive AI-powered DevSecOps platform [WWW Document]. URL https://about.gitlab.com/ (accessed 4.25.24).

Google, 2022. Classification: True vs. False and Positive vs. Negative | Machine Learning [WWW Document]. Google for Developers. URL https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative (accessed 2.21.24).

He, K., Zhang, X., Ren, S., Sun, J., 2015a. Deep Residual Learning for Image Recognition.

He, K., Zhang, X., Ren, S., Sun, J., 2015b. Deep Residual Learning for Image Recognition.

He, K., Zhang, X., Ren, S., Sun, J., 2015c. Deep Residual Learning for Image Recognition.

He, K., Zhang, X., Ren, S., Sun, J., 2014. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. pp. 346–361. https://doi.org/10.1007/978-3-319-10578-9_23

Henderson, P., Ferrari, V., 2017. End-to-end training of object class detectors for mean average precision.

Hosang, J., Benenson, R., Schiele, B., 2017. Learning non-maximum suppression.

Hui, J., 2019. mAP (mean Average Precision) for Object Detection. Medium. URL https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173 (accessed 6.13.24).

Ibrahim, M., Muhamad, N., Bakar, A., Jamaludin, K., Ahmad, S., Nor, N., 2011. Optimization of Micro Metal Injection Molding SS 316L For The Highest Green Strength By Using Taguchi Method. Advanced Materials Research 264–265. https://doi.org/10.4028/www.scientific.net/AMR.264-265.135

iguazio, 2022. What is Recall [WWW Document]. Iguazio. URL https://www.iguazio.com/glossary/recall/ (accessed 2.21.24).

Ioffe, S., Szegedy, C., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

JMP, 2023. Full Factorial Designs [WWW Document]. URL https://www.jmp.com/support/help/en/17.2/index.shtml#page/jmp/full-factorial-designs.shtml# (accessed 2.25.24).

kaggle, 2010. Kaggle: Your Machine Learning and Data Science Community [WWW Document]. URL https://www.kaggle.com/ (accessed 4.25.24).

Kamal, A., 2021a. YOLO, YOLOv2 and YOLOv3: All You want to know. Medium. URL https://amrokamal-47691.medium.com/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899 (accessed 1.30.24).

Kamal, A., 2021b. YOLO, YOLOv2 and YOLOv3: All You want to know. Medium. URL https://amrokamal-47691.medium.com/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899 (accessed 3.5.24).

Keita, Z., 2022. YOLO Object Detection Explained: A Beginner's Guide [WWW Document]. URL https://www.datacamp.com/blog/yolo-object-detection-explained (accessed 6.27.24).

Kenton, W., 2024. Analysis of Variance (ANOVA) Explanation, Formula, and Applications [WWW Document]. Investopedia. URL https://www.investopedia.com/terms/a/anova.asp (accessed 3.21.24).

Kharuzhy, 2018. Commits · jekhor/aerial-cars-dataset [WWW Document]. GitHub. URL https://github.com/jekhor/aerial-cars-dataset (accessed 4.12.24).

Kim, ki-chan, Ahn, J., Won, S., Hong, J.P., Lee, ju, 2007. A Study on the Optimal Design of SynRM for the High Torque and Power Factor. Magnetics, IEEE Transactions on 43, 2543–2545. https://doi.org/10.1109/TMAG.2007.893302

krishnab, 2018. YOLO object detection: how does the algorithm predict bounding boxes larger than a grid cell? Stack Overflow.

Kundu, R., 2022. F1 Score in Machine Learning: Intro & Calculation [WWW Document]. URL https://www.v7labs.com/blog/f1-score-guide, https://www.v7labs.com/blog/f1-score-guide (accessed 2.26.24).

Kutner, M.H. (Ed.), 2005. Applied linear statistical models, 5th ed. ed, The McGraw-Hill/Irwin series operations and decision sciences. McGraw-Hill Irwin, Boston.

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324. https://doi.org/10.1109/5.726791

Li, H., lv, xin, Zhang, S., 2022. Multiobjective Deep Reinforcement Learning based Joint Beamforming and Power Allocation in UAV assisted Cellular Communication. https://doi.org/10.21203/rs.3.rs-1634741/v1

Liu, S., Huang, di, 2019. Adaptive NMS: Refining Pedestrian Detection in a Crowd.

Liu, S., Qi, L., Qin, H., Shi, J., Jia, J., 2018. Path Aggregation Network for Instance Segmentation.

Loshchilov, I., Hutter, F., 2017. SGDR: Stochastic Gradient Descent with Warm Restarts.

Mantripragada, M., 2020. Digging deep into YOLO V3 - A hands-on guide Part 1 | by Manogna Mantripragada | Towards Data Science [WWW Document]. URL https://towardsdatascience.com/digging-deep-into-yolo-v3-a-hands-on-guide-part-1-78681f2c7e29 (accessed 4.13.24).

Misra, D., 2020. Mish: A Self Regularized Non-Monotonic Activation Function.

Misra, D., 2019. Mish: A Self Regularized Non-Monotonic Neural Activation Function.

MIT Lincoln Laboratory 1951, 1998. Datasets | MIT Lincoln Laboratory [WWW Document]. URL https://www.ll.mit.edu/r-d/datasets (accessed 7.3.24).

Müller, R., Kornblith, S., Hinton, G., 2020. When Does Label Smoothing Help?

Nagpal, M., 2023. The Ultimate Guide to YOLO3 Architecture [WWW Document]. ProjectPro. URL https://www.projectpro.io/article/yolov3-architecture/836 (accessed 1.31.24).

Navarro Tuch, S., López-Aguilar, A., Bustamante-Bello, R., Molina, A., Izquierdo-Reyes, J., Curiel-Ramirez, L., 2019. Emotional domotics: a system and experimental model development for UX implementations. International Journal on Interactive Design and Manufacturing (IJIDeM) 13. https://doi.org/10.1007/s12008-019-00598-z

Nguyen, M., Lam, H., Le, T., 2022. A Real-Time Application for Waste Detection and Classification. IJARCCE 11. https://doi.org/10.17148/IJARCCE.2022.11503

Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S., 2018. Activation Functions: Comparison of trends in Practice and Research for Deep Learning.

O'Shea, K., Nash, R., 2015. An Introduction to Convolutional Neural Networks.

Oti, E., Olusola, M., Eze, F., Enogwe, S., 2021. Comprehensive Review of K-Means Clustering Algorithms. International Journal of Advances in Scientific Research and Engineering 07, 64–69. https://doi.org/10.31695/IJASRE.2021.34050

Parico, A.I.B., Ahamed, T., 2021. Real Time Pear Fruit Detection and Counting Using YOLOv4 Models and Deep SORT. Sensors 21, 4803. https://doi.org/10.3390/s21144803

Parsania, P., Virparia, P., 2016. A Comparative Analysis of Image Interpolation Algorithms. IJARCCE 5, 29–34. https://doi.org/10.17148/IJARCCE.2016.5107

Patel, S., Patel, N., Deshpande, S., Siddiqui, A., 2021. Ship Intrusion Detection using Custom Object Detection System with YOLO Algorithm 08.

pawangfg, 2020. YOLO v2 - Object Detection. GeeksforGeeks. URL https://www.geeksforgeeks.org/yolo-v2-object-detection/ (accessed 1.30.24).

Pawar, R., 2022. labelImg: Convert annotations in XML to YOLO format [WWW Document]. Gist. URL https://gist.github.com/InputBlackBoxOutput/8ede7112531708b03a2e8e86ca2ff3d5 (accessed 2.24.24).

Pedro, J., 2023. Detailed Explanation of YOLOv8 Architecture — Part 1. Medium. URL https://medium.com/@juanpedro.bc22/detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e (accessed 7.4.24).

PyLessons, 2019. PyLessons [WWW Document]. URL https://pylessons.com/YOLOv3-introduction (accessed 4.13.24).

Ramachandran, P., Zoph, B., Le, Q., 2017. Swish: a Self-Gated Activation Function.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2015. You Only Look Once: Unified, Real-Time Object Detection.

Redmon, J., Farhadi, A., 2018. YOLOv3: An Incremental Improvement.

Redmon, J., Farhadi, A., 2016a. YOLO9000: Better, Faster, Stronger.

Redmon, J., Farhadi, A., 2016b. YOLO9000: Better, Faster, Stronger.

Riad, R., Teboul, O., Grangier, D., Zeghidour, N., 2022. Learning strides in convolutional neural networks.

ringringyi, 2023. ringringyi/DOTA_YOLOv2.

Robicquet, A., Sadeghian, A., Alahi, A., Savarese, S. (Eds.), 2016. Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes. Computer Vision – ECCV 2016, Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-319-46484-8_33

Seltman, H.J., 2018. Experimental Design and Analysis.

Shah, D., 2022. Mean Average Precision (mAP) Explained: Everything You Need to Know [WWW Document]. URL https://www.v7labs.com/blog/mean-average-precision, https://www.v7labs.com/blog/mean-average-precision (accessed 7.8.24).

Shaikh, S., Chopade, J., Kharate, G., 2023. Object Classification and Tracking Using Scaled P8 YOLOv4 Lite Model. Period. Polytech. Elec. Eng. Comp. Sci. 67, 102–111. https://doi.org/10.3311/PPee.20685

Shatravin, V., Shashev, D., Shidlovskiy, S., 2022. Sigmoid Activation Implementation for Neural Networks Hardware Accelerators Based on Reconfigurable Computing Environments for Low-Power Intelligent Systems. Applied Sciences 12, 5216. https://doi.org/10.3390/app12105216

Simonyan, K., Zisserman, A., 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition.

Simplilearn, 2023. What is XML: Overview on Comments, Attributes, Tags & Syntax | Simplilearn [WWW Document]. Simplilearn.com. URL https://www.simplilearn.com/tutorials/programming-tutorial/what-is-xml (accessed 3.10.24).

Singh, A., 2024. Selecting the Right Bounding Box Using Non-Max Suppression [WWW Document]. URL https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/ (accessed 2.28.24).

Singh, S., 2020. Swish as an Activation Function in Neural Network. Deep Learning University. URL https://deeplearninguniversity.com/swish-as-an-activation-function-in-neural-network/ (accessed 2.20.24).

Solawetz, J., 2020a. Data Augmentation in YOLOv4 [WWW Document]. Roboflow Blog. URL https://blog.roboflow.com/yolov4-data-augmentation/ (accessed 7.17.24).

Solawetz, J., 2020b. Data Augmentation in YOLOv4 [WWW Document]. Roboflow Blog. URL https://blog.roboflow.com/yolov4-data-augmentation/ (accessed 1.30.24).

Solawetz, J., Nelson, J., Sahoo, S., 2020. How to Train YOLOv4 on a Custom Dataset [WWW Document]. Roboflow Blog. URL https://blog.roboflow.com/training-yolov4-on-a-custom-dataset/ (accessed 4.25.24).

Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M., 2015. Striving for Simplicity: The All Convolutional Net.

Stanford University Computational Vision and Geometry Lab, 2009. Stanford Computational Vision and Geometry Lab [WWW Document]. URL https://cvgl.stanford.edu/resources.html (accessed 7.3.24).

Ševuliáková, P., Hurtik, P., 2023. Intersection over Union with smoothing for bounding box regression.

Subramanyam, V.S., 2021. Basics of Bounding Boxes. Analytics Vidhya. URL https://medium.com/analytics-vidhya/basics-of-bounding-boxes-94e583b5e16c (accessed 4.13.24).

SuperAnnotate, 2023. Mean average precision (mAP) in object detection | SuperAnnotate [WWW Document]. URL https://www.superannotate.com/blog/mean-average-precision-and-its-uses-in-object-detection (accessed 7.4.24).

Świeżewski, J., 2020. YOLO Algorithm and YOLO Object Detection - Machine Learning [WWW Document]. URL https://appsilon.com/object-detection-yolo-algorithm/ (accessed 2.25.24).

Synced, 2020. YOLO Creator Joseph Redmon Stopped CV Research Due to Ethical Concerns. SyncedReview. URL https://medium.com/syncedreview/yolo-creator-says-he-stopped-cv-research-due-to-ethical-concerns-b55a291ebb29 (accessed 5.26.24).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2014. Going Deeper with Convolutions. https://doi.org/10.48550/arXiv.1409.4842

Tan, M., Le, Q.V., 2020. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.

Tandon, A., 2024. adityatandon/VisDrone2YOLO.

Tepteris, G., Mamasis, K., Minis, I., 2023. State of the art object detection and recognition methods(draft) | DeOPSys Lab [WWW Document]. URL https://deopsys.aegean.gr/node/280 (accessed 12.12.24).

Tsang, S.-H., 2022. Review — YOLOv4: Optimal Speed and Accuracy of Object Detection. Medium. URL https://sh-tsang.medium.com/review-yolov4-optimal-speed-and-accuracy-of-object-detection-8198e5b37883 (accessed 6.27.24).

Ultralytics, 2023. YOLO Performance Metrics [WWW Document]. URL https://docs.ultralytics.com/guides/yolo-performance-metrics (accessed 2.28.24).

Vakili, M., Ghamsari, M., Rezaei, M., 2020. Performance Analysis and Comparison of Machine and Deep Learning Algorithms for IoT Data Classification.

VisDrone, 2024. VisDrone/VisDrone2018-DET-toolkit.

VisDrone, 2023. VisDrone/VisDrone-Dataset.

Wang, C.-Y., Bochkovskiy, A., Liao, H.-Y.M., 2020. Scaled-YOLOv4: Scaling Cross Stage Partial Network.

Wang, C.-Y., Liao, H.-Y.M., Yeh, I.-H., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., 2019. CSPNet: A New Backbone that can Enhance Learning Capability of CNN.

Wang, X., Lv, F., Li, L., Yi, Z., Jiang, Q., 2022. A novel optimized tiny YOLOv3 algorithm for the identification of objects in the lawn environment. Scientific Reports 12, 15124. https://doi.org/10.1038/s41598-022-19519-4

WikiDocs, 2023. C_2. Yolo V1 - EN - Deep Learning Bible - 4. Object Detection - Eng. [WWW Document]. URL https://wikidocs.net/167699 (accessed 5.26.24).

Woo, S., Park, J., Lee, J.-Y., Kweon, I.S., 2018. CBAM: Convolutional Block Attention Module.

Wu, L., Ma, J., Zhao, Y., Liu, H., 2021. Apple Detection in Complex Scene Using the Improved YOLOv4 Model. Agronomy 11, 476. https://doi.org/10.3390/agronomy11030476

Xia, G.-S., Ding, J., Xue, N., Bai, X., Yang, W., Yang, M.Y., Belongie, S., Luo, J., Datcu, M., Pelillo, P., Zhang, L., 2021. DOTA [WWW Document]. URL https://captain-whu.github.io/DOTA/dataset.html (accessed 2.9.24).

Xu, P., Li, Q., Zhang, B., Wu, F., Zhao, K., Du, X., Yang, C., Zhong, R., 2021. On-Board Real-Time Ship Detection in HISEA-1 SAR Images Based on CFAR and Lightweight Deep Learning. Remote Sensing 13, 1995. https://doi.org/10.3390/rs13101995

Xu, X., Zhang, X., Yu, B., Hu, X.S., Rowen, C., Hu, J., Shi, Y., 2018. DAC-SDC Low Power Object Detection Challenge for UAV Applications.

Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y., 2019a. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features.

Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y., 2019b. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features.

Zhang, F., Wang, S., Cui, X., Wang, X., Cao, W., Yu, H., Fu, S., Pan, X., 2022. Goat-Face Recognition in Natural Environments Using the Improved YOLOv4 Algorithm. Agriculture 12, 1668. https://doi.org/10.3390/agriculture12101668

Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D., 2018. mixup: Beyond Empirical Risk Minimization.

Zhang, Z., 2020. YOLO2 Walkthrough with Examples [WWW Document]. Medium. URL https://towardsdatascience.com/yolo2-walkthrough-with-examples-e40452ca265f (accessed 1.30.24).

Zhao, W., Alwidian, S., Mahmoud, Q.H., 2022. Adversarial Training Methods for Deep Learning: A Systematic Review. Algorithms 15, 283. https://doi.org/10.3390/a15080283

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., Ren, D., 2019a. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression.

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., Ren, D., 2019b. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression.

Zheng, Z., Wang, P., Ren, D., Liu, W., Ye, R., Hu, Q., Zuo, W., 2021. Enhancing Geometric Factors in Model Learning and Inference for Object Detection and Instance Segmentation.

Zhu, P., Wen, L., Du, D., Bian, X., Fan, H., Hu, Q., Ling, H., 2021. Detection and Tracking Meet Drones Challenge.

Zulkifli, H., 2018. Understanding Learning Rates and How It Improves Performance in Deep Learning [WWW Document]. Medium. URL https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10 (accessed 7.18.24).

Appendix A. IoU and CIoU losses

A.1 Intersection over Union (IoU)

The Intersection over Union (IoU) is used during the training process of the YOLO algorithm, where it facilitates the determination of an anchor box that seeks optimal matching with a specified object (referred to as the ground truth bounding box). IoU represents the ratio of the intersecting area between two bounding boxes (ground truth and anchor) to their combined area.

To better understand the IoU, we will explain the following concepts:

- Anchor boxes: are predetermined bounding boxes of different sizes and aspect ratios
- Ground truth bounding box: is the manually annotated box that precisely encloses an object and provides its respective label.

The mathematical representation for IoU is delineated as follows (Ševuliáková and Hurtik, 2023):

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ union} \tag{A.1}$$

The IoU values ranges from 0 to 1 (Ultralytics, 2023), where:

- 0 indicates that the two boxes do not overlap, showing a complete misfit between the two
- 1 signifies that the predicted box aligns perfectly with the ground truth box, thus indicating a perfect detection
- 0.5 or 0.75 IoU thresholds are widely used as benchmarks for the training process of a model.

A.2 Complete Intersection over Union (CIoU)

Complete Intersection over Union (CIoU) is an updated version of IoU, which optimizes the performance of the model by suppressing differences between the predicted bounding boxes and the ground truth bounding boxes (Zheng et al., 2021).

Figure A.1 shows a predicted bounding box (green box) and a ground truth bounding box (black box) with their centers marked by a green and black circle, respectively. The purple line represents the Euclidean distance ρ between the centers, and the gold line f is the diagonal of the smallest enclosing box that can contain both the predicted bounding box and the ground truth bounding box. Therefore, CIoU combines IoU, the center distance ρ, and the aspect ratio matching. From the above factors, CIoU tries to improve the accuracy of bounding box predictions in respect of location, size, and shape, resulting in better detection performance.

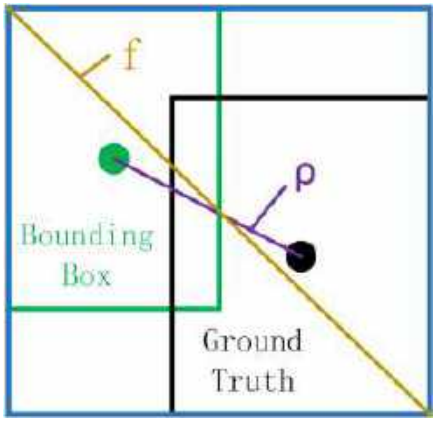

Figure A.1 *Complete Intersection over Union (CIoU) (Wang et al., 2022)*

The equation of CIoU is described below:

$$R_{CIoU} = \frac{\rho^2(b, b^{gt})}{c^2} + \alpha u \tag{A.2}$$

where,

- b and b^{gt}: represent the central points of the predicted bounding box (B) and the ground truth bounding box (B^{gt}), respectively
- ρ: is the Euclidean distance between the centroids of the predicted and ground truth bounding boxes
- c: is the length of the diagonal line that covers both the predicted and ground truth bounding boxes when they are enclosed in the smallest possible area
- α: can be adjusted to balance the importance of the distance between objects and their size differences when calculating CIoU
- u: functions as a standardizing factor, accommodating the difference in aspect ratio between the predicted and ground truth bounding boxes.

In Equation A.2, "α" and "u" are calculated as follows:

$$a = \frac{u}{(1 - IoU) + u'} \tag{A.3}$$

$$u = \frac{4}{\pi^2}\left(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h}\right)^2 \tag{A.4}$$

where,

- w and h: are the width and the height of the predicted bounding box b
- w^{gt} and h^{gt}: are the width and the height of the ground truth bounding box b^{gt}
- IoU: is the Intersection over Union.

The YOLOv4 and YOLOv4-p6 contain the CIoU in all their detection heads, as shown in Figure A.2.

```
[yolo]
mask = 0,1,2
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
scale_x_y = 1.2
iou_thresh=0.213
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
nms_kind=greedynms
beta_nms=0.6
max_delta=5
```

Figure A.2 *Position of the Complete Intersection over Union (CIoU) in YOLOv4 configuration file*

Appendix B. Functionality of bounding boxes

Both YOLOv4 and YOLOv4-p6 algorithms use the head component of their networks to detect objects. This component includes the detection heads, which use the bounding boxes to predict and detect the desired objects during the validation and testing processes.

The following sections of this appendix describe the use of ground truth bounding boxes, anchor bounding boxes and bounding boxes.

B.1 Ground truth bounding boxes

Ground truth bounding boxes are manually created and their spatial information is provided as input data at the beginning of training. Specifically, we fed the algorithm with a UAV dataset containing images with annotations that specify the coordinates of all objects the model has been (or will be) trained on. These coordinates form a rectangular box which is referred to as the ground truth bounding box (see Figure B.1). Consequently, these boxes are labelled rectangular boxes that indicate the position of an object within the image and its corresponding class. Each detected object in the image has a single ground truth bounding box (Subramanyam, 2021).

Figure B.1 *Ground truth bounding box containing a dog (SuperAnnotate, 2023)*

B.2 Anchor boxes

Anchor boxes are predetermined bounding boxes of different sizes and aspect ratios used to capture the selected object classes during detection. Instead of applying bounding boxes instantly, the model refines the provided anchor boxes during the training process to create accurate bounding boxes. The predetermined aspect ratios of the anchor boxes are established using K-means clustering (Redmon and Farhadi, 2016b). This algorithm finds the anchor box parameters that result in optimal Intersection over Union (IoU) results, in order to optimize the fit of the anchor boxes to the training data by determining the most suitable dimensions. Consequently, the selected distance metric is mathematically represented as follows:

$$d(box, centroid) = 1 - IoU(box, centroid) \quad \text{(B.1)}$$

where,

- box: represents the ground truth bounding box
- $centroid$: indicates the center point of the anchor bounding box
- $d(box, centroid)$: returns the percentage of the non-intersecting area between the ground truth bounding box and the anchor bounding box.

Throughout the training process, each grid cell is associated with a set of anchor boxes (Mantripragada, 2020). Their purpose is to include objects of different scales and shapes, functioning as predefined reference boxes. In YOLOv4, each cell the grid includes three anchor boxes. These are categorized into small, medium and large detection scales, resulting in a total of nine anchor boxes across all detection heads. In contrast, YOLOv4-p6 has four anchor boxes because the network is bigger and has four detection heads. Consequently, it has a total of sixteen anchor boxes across all detection heads (4 anchor boxes for each detection head).

The objective of the training process is to refine the initial anchor boxes during training to create better bounding boxes that match better the ground truth bounding boxes. Specifically, during the training iterations, the model adjusts the sizes and positions of anchor boxes based on the ground truth boxes. This process involves refining the initial dimensions and central positions of the anchor box that has the highest IoU with the ground truth box, ensuring that the anchor box with the most significant overlap is selected for adjustment. After adjustment, each selected anchor box produces a single bounding box that predicts the presence of an object of a particular class with a confidence score at that specific location.

YOLOv4 and YOLOv4-p6 are configured with anchor box dimensions originally based on the COCO (Microsoft Common Objects in Context) dataset (COCO, 2017), which includes 328,000 images of persons and various common objects. Since we are using a different dataset, we need to readjust our anchor box sizes to match with both the selected image resolutions of $1280x1280$ and $960x960$. One method is with the application of k-means algorithm, as described below:

./darknet detector calc_anchors cfg/"NAME_OF_THE_DATA_FILE".data -num_of_clusters "NUMBER_OF_CLUSTERS" -width "NUMBER_OF_IMAGE_WIDTH" -height "NUMBER_OF_IMAGE_HEIGHT"
An example by applying the code: ./darknet detector calc_anchors cfg/traffic_lights.data -num_of_clusters 16 -width 1280 -height 1280

Figure B.2 *Command for applying the k-means algorithm in Darknet*

where,

- **./darknet**: is the executable file of Darknet
- **detector calc_anchors**: instructs Darknet to calculate the anchor boxes corresponding to the object detection model
- **cfg/"NAME_OF_THE_DATA_FILE".data**: specifies the path to the data file, as shown in Figure B.3. The placeholder "NAME_OF_THE_DATA_FILE" needs to be replaced with the name of the data file that contains information about the paths to the training and validation or testing sets, the

number of classes and the backup path. The backup path saves the weights of the model (every 1,000 iterations) during the training process

```
classes= 4
train  = /media/deopsys/Hard_Disk/panos/Final_dataset(swapped_classes)/Paths/Train.txt
valid  = /media/deopsys/Hard_Disk/panos/Final_dataset(swapped_classes)/Paths/Val.txt
names = /home/deopsys/Documents/darknet/data/yolov4_4_classes.names
backup = /media/deopsys/Hard_Disk/panos/Experiments/cfg1
```

Figure B.3 *Representation of a data file*

-**num_of_clusters "NUMBER_OF_CLUSTERS":** specifies the number of clusters into which the anchor boxes should be grouped. This number depends on the model being trained. Therefore, "NUMBER_OF_CLUSTERS" should be replaced with the actual number of clusters specified in the corresponding configuration file. For example, as shown in Figure B.4, the YOLOv4-p6 model uses sixteen pairs of anchor box sizes in the “anchors” hyperparameter, indicating that there are sixteen clusters

```
[yolo]
mask = 12,13,14,15
anchors = 13,17,  31,25,  24,51,  61,45,  61,45,  48,102,  119,96,  97,189,  97,189,  217,184,  171,384,  324,451,  324,451,  545,357,  616,618,  1024,1024
classes=4
```

Figure B.4 *Position of anchors in YOLOv4-p6 configuration file*

- -**width "NUMBER_OF_IMAGE_WIDTH"**: specifies the input image width that is included in the configuration file
- -**height "NUMBER_OF_IMAGE_HEIGHT"**: specifies the input image height that is included in the configuration file.

By providing better dimensions for our initial anchor boxes, we improve the detection performance of our model, as the model can create more accurate bounding boxes.

B.3 Bounding boxes

The model does not directly use bounding boxes to predict the desired objects. Instead, it uses refined anchors to assist in generating bounding boxes during training. These bounding boxes learn and predict spatial offsets (vertical and horizontal) and sizes (height and width) from the refined anchors. As a result, the final feature map represents different offset and size predictions for each anchor defined in the output feature map, resulting in object detection for each class (Tepteris et al., 2023). Subsequently, the generated bounding boxes are evaluated during the validation process by measuring their overlap with the ground truth bounding boxes using IoU (YOLOv4 and YOLOv4-p6 use an updated version of IoU known as Complete Intersection over Union (CIoU), as discussed previously).

As illustrated in Figure B.5, the light green rectangular boxes represent bounding boxes showing the spatial locations within the image where cars are located (Subramanyam, 2021).

Figure B.5 *Displaying bounding boxes for the car class within light green boundaries (Dupont, 2023)*

As mentioned above, YOLOv4 uses three different grid sizes to detect objects at different scales. Specifically, the model applies a 19 by 19 grid generating 1,083 $(19x19x3)$ bounding boxes for the detection of large objects, a 38 by 38 grid generating 4,332 $(38x38x3)$ bounding boxes for the detection of medium objects and a 76 by 76 grid generating 17,328 $(76x76x3)$ bounding boxes for the detection of small objects. Consequently, YOLOv4 generates a total number of 22,743 bounding boxes, which are then reduced using Non-Maximum Suppression (NMS).

Similarly, YOLOv4-p6 model creates a total of 102,000 bounding boxes, and specifically:

- For very small object detection: $(160x160x3) = 76{,}800$
- For small object detection: $(80x80x3) = 19{,}200$
- For medium object detection: $(40x40x3) = 4{,}800$
- For large object detection: $(20x20x3) = 1{,}200$

In the detection heads of YOLOv4, the grid responsible for detecting large objects is defined with dimensions of $19x19x255$. Each cell within this grid contains three anchor boxes, which are tasked with generating three predicted bounding boxes possessing the following characteristics:

- The variables t_x and t_y represent the coordinates of the center
- t_w and t_h are the predicted width and height, respectively
- P_0 signifies the objectness score
- P_{c_i} represents an array of confidence scores corresponding to each class category within the predicted bounding box.

For each of the three predicted bounding boxes in a cell, the objectness score is calculated by multiplying the $Pc(object)$ (representing the probability that the box contains an object) by the Intersection over Union (IoU) between the predicted and ground truth bounding boxes. Then, the resultant objectness score is multiplied by the predicted class probability for each class in that box to calculate the confidence score

for each class. After calculating the confidence scores in the bounding box, the highest score across all classes is selected as the predicted object (class) for that bounding box (Redmon and Farhadi, 2018).

In the detection process, the objectness score (Redmon and Farhadi, 2018) within the central cell of the ground truth bounding box is indicated as 1, while the objectness scores within other cells included in the ground truth bounding box vary between 0 and 1 in relation to their distance from the center of the ground truth bounding box. Conversely, cells outside the ground truth bounding box are assigned an objectness score of 0. Each cell within the SxS grid contains three anchors, and for each anchor, a bounding box is proposed, determined by its center coordinates and dimensions (b_x, b_y, b_h, b_w). As shown in Figure B.6, the bounding box with the highest overlap with the ground truth box is considered as the bounding box responsible for object prediction, while the remaining two are ignored.

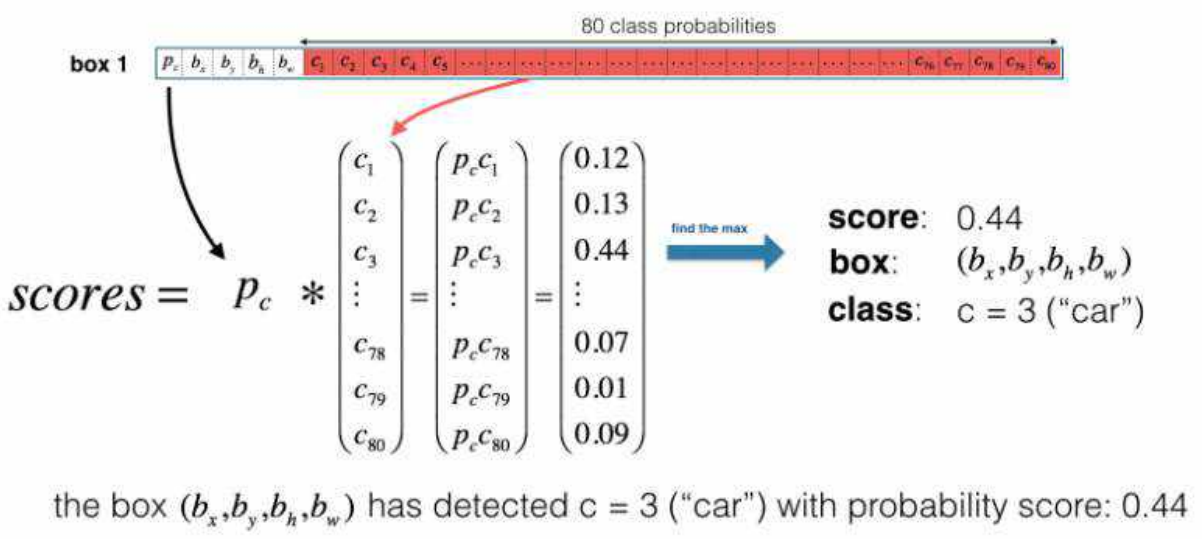

Figure B.6 *Calculation of the probability for an anchor to include a specific class (PyLessons, 2019)*

where,

- P_o represents the objectness score, which is assigned a value of 1 in the central cell of each ground truth bounding box. Cells surrounding the central cell have values less than 1, such as 0.95 and 0.9, while cells distant from the central cell have values close to zero
- P_{c_i} represents the class probability for class i, derived from the YOLOv4 algorithm during the training process.

For instance, Figure B.7 represents a 19x19 grid (large detection head). The red bounding boxes classified the objects as "car" because they had the highest overlap with their respective ground truth box. Therefore, the detected objects labelled as "car" are assumed to be the most appropriate result for detection.

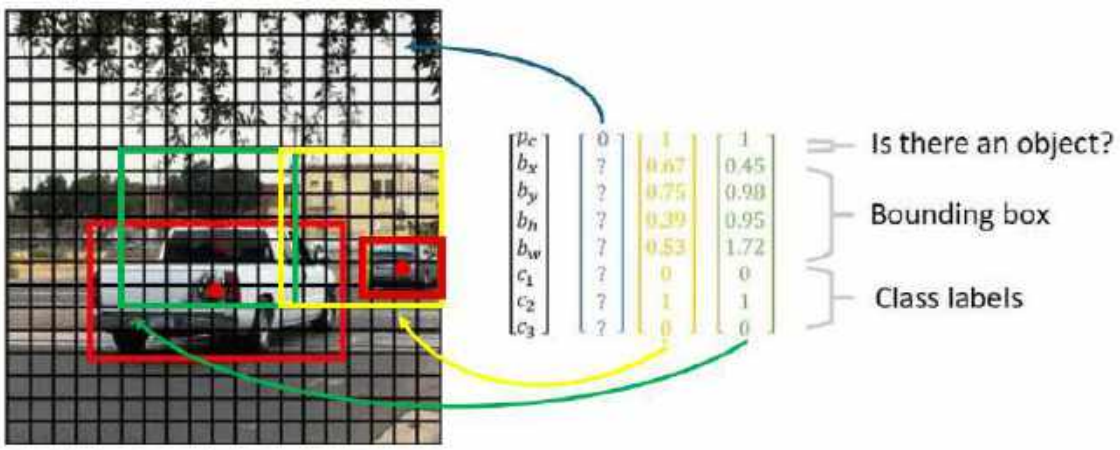

Figure B.7 *Two bounding boxes representing the class "car" (krishnab, 2018)*

B.4 Calculations regarding the resultant bounding boxes

In accordance with the methodology introduced in YOLO9000 and YOLOv3 algorithms, YOLOv4 and YOLOv4-p6 use dimension clusters as anchor boxes for the prediction of bounding boxes. The model generates four coordinates for each bounding box, namely t_x, t_y, t_w and t_h. If the cell is moved from the image's top-left corner by coordinates (c_x, c_y), and the previous bounding box has a width and height represented by p_w and p_h respectively, the predicted values are related as follows (Bochkovskiy, 2021a):

$$b_x = \sigma(t_x) * 1.1 - 0.05 + c_x \quad \text{(B.2)}$$

$$b_y = \sigma(t_y) * 1.1 - 0.05 + c_y \quad \text{(B.3)}$$

$$b_w = p_w \, e^{t_w} \quad \text{(B.4)}$$

$$b_h = p_h \, e^{t_h} \quad \text{(B.5)}$$

In the above equations, the terms are defined as follows: b_x, b_y, b_w and b_h. They represent the parameters of the bounding box, where b_x and b_y are the coordinates of the center of the predicted bounding box, while b_w and b_h indicate its width and height, respectively. p_w and p_h are the width and height of the anchor boxes, where p_w is calculated as the ratio of the width of the anchor box (w_{anchor}) to the width of the image (w_{image}), and p_h is calculated similarly using the height dimensions. c_x and c_y refer to the shift of the coordinates of the top-left corner of the cell containing the center of the predicted object from the top-left corner of the image. The function σ represents the sigmoid function, while t_x and t_y represent the shifts of the anchor from the top-left coordinates of the grid cell to which the anchor belongs, with the cell also including the center of the predicted object and the ground truth box. t_w and t_h are parameters that adjust the width and height of the anchor to match those of the ground truth box.

Appendix C. Convolution operations and network architectures

C.1 Stride operation

The stride operation refers to the size of the step performed by the filters of convolutional layers during the processes of feature extraction, downsampling and pooling. In that way, it controls the filter's movement across input images. The amount of movement the filter makes at each step, either horizontally, vertically, or both, depends on its configuration (Riad et al., 2022). For instance, the following table displays a 4x4 image:

Image			
1	2	0	1
4	3	1	2
2	0	0	4
1	1	0	2

Figure C.1 *Image size of 4x4*

We are going to perform a 2x2 convolutional operation with a stride of 2, which involves moving a 2x2 filter with a stride of 2 across the image:

Filter	
2	1
0	1

Figure C.2 *Filter size of 2x2*

We start by calculating the top-left element of the feature map, which involves performing the following operation:

$$(1 * 2) + (2 * 0) + (4 * 1) + (3 * 1) = 8 \quad \text{(C.1)}$$

Then, the filter is shifted to the right by a distance equivalent to 2 pixels, as demanded by the stride, to perform a similar operation:

$$(0 * 2) + (1 * 0) + (1 * 1) + (2 * 1) = 3 \quad \text{(C.2)}$$

Following the computation of the top row within the feature map, the filter is shifted downward by 2 pixels and repeats the same process, till there are no more rows below the table:

$$(2 * 2) + (0 * 0) + (1 * 1) + (1 * 1) = 6 \quad \text{(C.3)}$$

$$(0 * 2) + (4 * 0) + (0 * 1) + (2 * 1) = 2 \quad \text{(C.4)}$$

After completing the 2x2 convolutional operation with a step of 2, the resulting feature map is:

Feature map	
8	3
6	2

Figure C.3 *The output feature map*

The output feature map is 2x2, which is smaller than the input size of 4x4.

Even if neural networks may adopt larger input images and more complicated filters, the concept of stride remains the same. For instance, the YOLOv4 and YOLOv4-p6 algorithms apply the stride operation in their algorithms as follows:

```
# Downsample

[convolutional]
batch_normalize=1
filters=128
size=3
stride=2
pad=1
activation=mish

# Split

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish
```

Figure C.4 *Convolutional layer with stride in YOLOv4-p6 configuration file*

C.2 Feature map

During the forward pass of the YOLOv4 and Scaled YOLOv4 algorithms, a series of convolutions are applied to the input image using different filters. This results in the creation of multiple feature maps, which initially capture basic features like edges and corners. As these feature maps progress through the networks, they start to represent more complex features such as shapes and textures (Lecun et al., 1998). Consequently, at the end of these networks (in the heads component), these refined feature maps are applied with bounding boxes to detect and classify objects.

The use of many feature maps helps detection models to identify more detailed features, but also increases the computational requirements and introduces a potential risk of overfitting.

The following is an example of a $3x3$ filter applied to a $5x5$ image with stride of 2, creating a $2x2$ feature map:

$$\begin{bmatrix} 1 & 0 & 1 & 2 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 2 & 1 & 3 & 0 & 3 \\ 1 & 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 15 & 7 \\ 7 & 10 \end{bmatrix}$$

To begin with, we position the filter at the top left corner of the input matrix to calculate the first value of the feature map as follows:

$$(1 * 1) + (0 * 0) + (1 * 1) + (0 * 0) + (2 * 2) + (1 * 0) + (2 * 1) + (1 * 0) + (3 * 1) = 15 \quad \text{(C.5)}$$

After we complete the previous step, we shift the filter horizontally 2 pixels to the right to calculate the next value in the feature map:

$$(1 * 1) + (2 * 0) + (0 * 1) + (1 * 0) + (0 * 2) + (0 * 0) + (3 * 1) + (0 * 0) + (3 * 1) = 7 \quad \text{(C.6)}$$

Next, starting again from the top left corner, we shift the filter vertically downwards by 2 pixels to calculate the next value in the feature map:

$$(2 * 1) + (1 * 0) + (3 * 1) + (1 * 0) + (0 * 2) + (2 * 0) + (0 * 1) + (0 * 0) + (2 * 1) = 7 \quad \text{(C.7)}$$

Finally, we move the filter horizontally 2 pixels to the right of its previous position to calculate the last value in the feature map:

$$(3 * 1) + (0 * 0) + (3 * 1) + (2 * 0) + (1 * 2) + (0 * 0) + (2 * 1) + (0 * 0) + (1 * 0) = 10 \quad \text{(C.8)}$$

C.3 Convolutional layers

In the convolutional layer, a series of changeable filters that are commonly referred to as kernels are used to process the input image, resulting in feature extraction. These filters are numeric matrices whose dimensions are set during initialization and remain constant. During training, each filter moves across the input image, conducting element-wise cross-correlation operations (see Equation C.9) on its adjustable parameters and specific sections of the input, resulting in the formation of single values within the produced feature map (O'Shea and Nash, 2015). Throughout the same process, the filter weights are continually adjusted to minimize the loss function.

The mathematical expression used for this operation is applied to each colour channel separately, as shown in Equation C.9 (Ian Goodfellow, 2016):

$$F(i,j) = (I * K)(i,j) \sum_m \sum_n \big(I(i+m, j+n) K(m,n)\big) \quad \text{(C.9)}$$

where,

- I: represents the input image
- K: symbolizes the kernel
- F: represents the output feature map
- i and j: represent the locations of the pixels to be processed by convolution
- m: is the width of the kernel
- n: is the height of the kernel.

As illustrated in Figure C.5, a 6x6 colour image with three input colour channels is convolutionally processed with the respective filters. Each filter slides with a stride of 2 (moving two cells at a time) within its channel. Consequently, the convolution operations result in a feature map matrix of size 2x2.

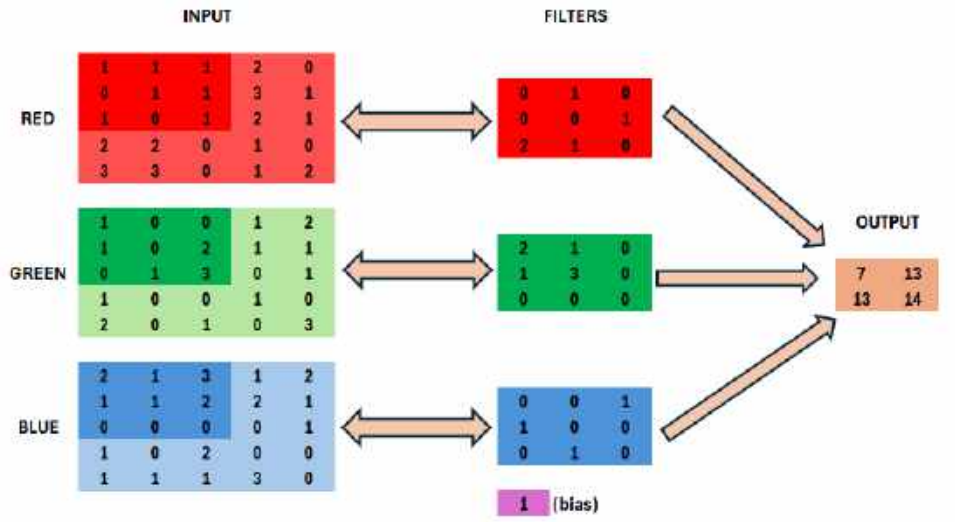

Figure C.5 *Filter size of 3x3 (for each channel) moves across the input to produce the output*

The first operation of the red channel with the respective filter is:

$$(1 * 0) + (1 * 0) + (1 * 2) + (0 * 1) + (1 * 0) + (1 * 1) + (1 * 0) + (0 * 1) + (1 * 0) = 3 \quad \text{(C.10)}$$

The first operation of the green channel with the respective filter is:

$$(1 * 2) + (0 * 1) + (0 * 0) + (1 * 1) + (0 * 3) + (2 * 0) + (0 * 0) + (0 * 1) + (0 * 3) = 3 \quad \text{(C.11)}$$

The first operation of the blue channel with the respective filter is:

$$(2 * 0) + (1 * 1) + (3 * 0) + (1 * 0) + (1 * 0) + (2 * 1) + (0 * 1) + (0 * 0) + (0 * 0) = 3 \quad \text{(C.12)}$$

By summing the values of 3, 3, and 3, and then adding the bias value of 1, the resulting output is 7, which represents the value located at the top-left position of the output feature map matrix. To calculate the values of the remaining three cells in the feature map matrix, this process is repeated, moving the kernel horizontally and vertically by 2 pixels in each operation (Tepteris et al., 2023).

With the use of convolutional layers, the network is capable to transfer features (as feature maps) across the entire network. In addition, there are two types of convolutional operations that help with this process, and these are:

- **Convolutional layers with 3x3 filters size:** The 3x3 convolution operation uses a 3x3 filter to analyze input images or feature maps, allowing the identification of localized patterns and features

within the input data. By applying multiple 3x3 convolutions, CNN designs build deeper networks capable of learning multi-level representations of the input data (Springenberg et al., 2015)

- **Convolutional layers with 1x1 filters size:** The convolution 1x1 operation uses a 1x1 filter to analyze input images or feature maps. The 1x1 convolutional layer is used for the following reasons (Szegedy et al., 2014):
  - A 1×1 filter possesses a single parameter or weight for each channel within the input, similar to the application of any filter that results in a single output value. This configuration allows the 1×1 filter to function similarly to a single neuron, by combining values from the same position across all feature maps in the input. The application of this single neuron, with a stride of 1, passes the input left-to-right and top-to-bottom, eliminating the need for padding and resulting in a feature map that represents the width and height of the input
  - It functions as a linear weighting or projection of the input, as it does not involve neighboring pixels in the input, preventing its classification as a traditional convolutional operation. Since traditional convolutional operations involve looking at groups of neighboring pixels to capture spatial information. Even though the convolution 1x1 operation is a linear operation, if we add a non-linear activation function (like Mish activation function), it can perform better computations on the input features
  - The 1×1 filter is used to perform dimensionality reduction and feature transformation within the network. With the application of multiple 1×1 filters, the network can adjust the number of channels in the feature maps, thus controlling the depth of the feature maps as needed (He et al., 2015b).

A network is capable to adjust the number of feature maps at any desired location by using convolutional layers with 1x1 filters size. This operation is called projection layer or feature map pooling. (Brownlee, 2019).

C.4 Padding

Padding in convolutional neural networks (such as YOLO algorithms) involves adding extra pixels around the edges of the input feature map, typically filled with zeros, before applying convolution operations. This technique ensures that information at the edges of the feature map is processed in the same way as information in the central regions during convolution.

There are two types of padding used (GeeksforGeeks, 2019):

- Valid Padding: In valid padding, no additional pixels are added to the input feature map, resulting in an output feature map that is smaller in size compared to the input. Specifically, it is effective if we want to decrease the spatial dimensions of the feature maps
- Same Padding: In same padding, additional pixels (often zeros) are added to the input feature map to ensure the size of the output feature map matches that of the input feature map (see Figure C.6). Specifically, it is effective if we want to preserve the spatial dimensions of the feature maps.

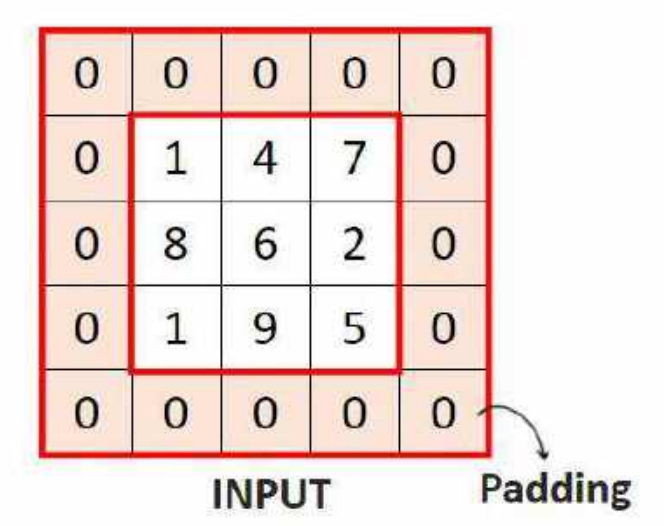

Figure C.6 *Example of same padding (Pedro, 2023)*

C.5 Batch normalization

Batch normalization takes place between a convolution operation and an activation function to reduce the effect of "internal covariate shift" effect. This effect results from the randomness in both the parameter initialization and the input data. Specifically, it can distort network training, but batch normalization (BN) handles it by standardizing network activations using the mean and variance calculated across instances within each mini-batch at each iteration. In addition, it improves model training by allowing the use of higher learning rates and reducing vulnerability to changes in input weights. This method is applied after the activation function within a convolutional layer and before the successive layers in the network architecture (Ioffe and Szegedy, 2015).

During the training process, batch normalization is applied to transform the input data as described below:

$$\mu_t(\theta_t) = \frac{1}{m}\sum_{i=1}^{m} x_{t,i}(\theta_t) \tag{C.13}$$

$$\sigma_t(\theta_t) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_{t,i}(\theta_t) - \mu_t(\theta_t))^2} \tag{C.14}$$

$$\hat{x}_{t,i}(\theta_t) = \frac{x_{t,i}(\theta_t) - \mu_t(\theta_t)}{\sqrt{\sigma_t(\theta_t)^2 + \varepsilon}} \tag{C.15}$$

$$y_{t,i}(\theta_t) = \gamma * \hat{x}_{t,i}(\theta_t) + \beta \tag{C.16}$$

Equations C.13 and C.14 are applied to calculate the mean and variance of the activation values throughout the batch. Subsequently by using Equation C.15, the activation vector $\hat{x}_{t,i}(\theta_t)$ is normalized to ensure consistency of each output to a standardized normal distribution within the batch, with the

inclusion of a constant ε to maintain numerical stability throughout this procedure. Following this, the batch normalization process calculates the layer's output, referred as $y_{t,i}(\theta_t)$, through a linear transformation involving two adjustable parameters, γ and β (as indicated in Equation C.16). The adjustment of the values of γ and β allows the model to effectively control the bias and the standard variation, respectively.

C.6 Downsampling operation

The downsampling operation reduces the size (number of pixels) of the input image during the training process of the model. Specifically, downsampling operation is achieved through the stride operation in YOLOv4 and YOLOv4-p6 algorithms (see Figure C.4 and C.7). The purpose of this operation is to:

- Reduce computational load and memory requirements
- Prevent overfitting by reducing the resolution of the input images, which allows the model to focus on more detailed features.

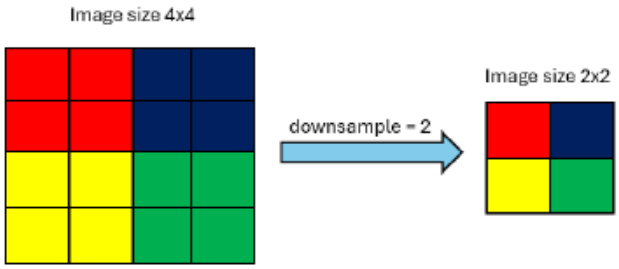

Figure C.7 *Downsampling operation*

C.7 Upsampling operation

With the application of upsampling operation, the number of pixels within the downsampled images can be increased, which improves both the resolution and dimensions of the input images. In the YOLOv4 and YOLOv4-p6 algorithms (see Figure C.8), the method applied for upsampling is Nearest Neighbor interpolation (GeeksforGeeks, 2021). This method determines the value of a target pixel by identifying the nearest pixel in the input image. It does this by rounding the coordinates of the desired interpolation point to find the closest pixel. As depicted in Figure C.8, this technique matches each pixel with its closest counterpart, therefore enlarging the image (Parsania and Virparia, 2016).

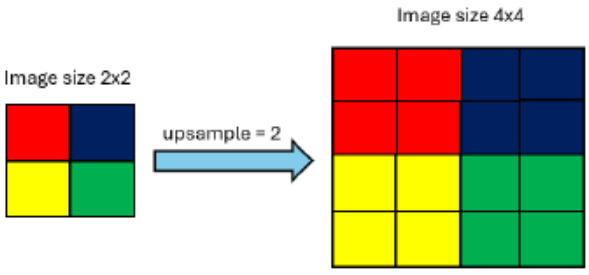

Figure C.8 *Upsampling operation*

In YOLOv4 and YOLOv4-p6 algorithms, the upsampling operation is used in the configuration file as follows:

```
[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish

[upsample]
stride=2

[route]
layers = 158

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=mish
```

Figure C.9 *Upsample layer in YOLOv4-p6 configuration file*

C.8 Residual blocks

Residual blocks were introduced to address the issue of the vanishing gradient problem during the training process of a network. The fundamental idea in the creation of the residual blocks is the use of skip connections, also called shortcut connections. These connections allow information to pass directly through one or more layers, making it easier to transfer directly to deeper layers in the network. Furthermore, the structure of a residual block can be expanded to include additional convolutional layers (He et al., 2015c), as shown in Figure C.10.

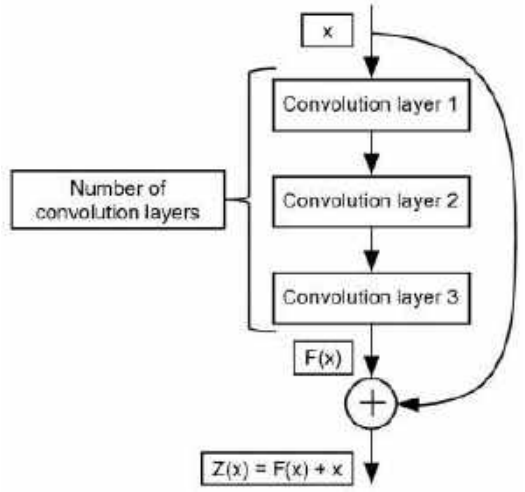

Figure C.10 *Residual block architecture*

As depicted in Figure C.10, x represents the block's input image, $Z(x)$ represents the result of the block, and the number of convolution layers indicates the number of convolutional layers along with their corresponding activations within the layer. In the residual block, the skip connections involve combining the input with the output of the convolutional layers, which improves the ability of the network to capture

more features. This occurs in the "+" layer, which acts as a shortcut layer. These layers are also used in YOLOv4 and YOLOv4-p6, as shown in Figure C.11:

```
# Residual Block

[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=mish

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=mish

[shortcut]
from=-3
activation=linear

# Transition first

[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=mish
```

Figure C.11 *Residual block in YOLOv4-p6 configuration file*

C.9 Route layers

In YOLO models, the route layer passes outputs from earlier layers to successive ones without any intermediate processing. This allows for the extraction of detailed features from previous network stages and the combination of outputs from different layers, as long as they are dimensionally compatible. Similar to the residual block in networks, which functions with the use of convolutional and shortcut layers, the route layer improves the integration of features from previous layers (Curti, 2020). In the architecture of YOLOv4-p6 model, multiple route layers specified by the "layers" parameter allow the combination of feature maps from selected layers, as detailed below:

```
[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=mish

# Merge [-1, -(3k+4)]

[route]
layers = -1,-7

# Transition last

# 10 (previous+7+3k)
[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=mish
```

Figure C.12 *Route layer in YOLOv4-p6 configuration file*

In Figure C.12, the route layer collects feature maps from both the seventh-to-last layer and the last layer. Subsequently, it merges these feature maps through concatenation operation, which combines the depths of two feature maps to capture low-level features and forwards the combined output to the next layer in the network.

C.10 Cross-stage partial connections (CSP connections)

Cross-stage partial connections (CSP) allow information to flow not only in a forward direction through the layers of the network, but also across different stages or blocks within the network (Wang et al., 2019). The output of a layer (or group of layers) is added back to the output of a previous layer, essentially skipping certain intermediary levels. This way, the network can learn to focus on the difference between the two outputs, instead of having to learn to match the required mapping.

As depicted in Figure C.13, the basic concept is to divide the output feature map into two different paths (Bochkovskiy, 2021b):

- A **primary route** (fifty percent of the features pass) improves semantic information generation through an extended receptive field, which helps the network capture and process more semantic information
- A **secondary bypass route** (the remaining fifty percent of the features are directed) allows the preservation of spatial information by way of a more restricted perceptual field, which helps preserve spatial details by focusing on local information.

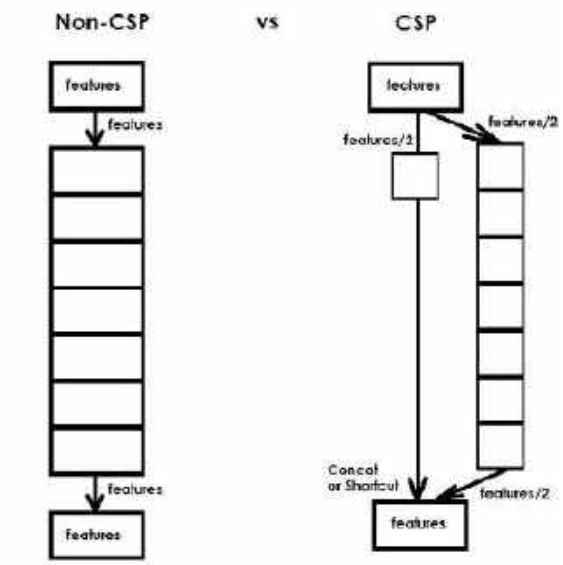

Figure C.13 *Conventional network (on the left) and CSP-Enhanced network (on the right) (Bochkovskiy, 2021c)*

In the YOLOv4 model, the CSP connections are present in the backbone of the architecture. With the application of CSPDarknet-53, the model captures and processes features from images, resulting in better accuracy and efficiency in identifying objects.

In the YOLOv4-p6 model, the CSP connections are used in the entire structure of the network.

C.11 Spatial Pyramid Pooling

The YOLOv4 algorithm uses the Spatial Pyramid Pooling (SPP) to resolve the difficulty of handling images with different sizes and resolutions (He et al., 2014). SPP was designed to allow Convolutional Neural Networks (CNNs) to process input images of varying sizes and produce fixed-length feature vectors, regardless of the image dimensions (see Figure C.14). Since traditional CNNs require fixed-size inputs, this is especially helpful when working with pictures of varied sizes throughout the training and testing procedures.

The Spatial Pyramid Pooling layer operates as follows:

1. **Input Image**: Consider an input image with varying dimensions, such as height H and width W
2. **Feature Extraction**: The image progresses through the early stages of CNN, going through convolution and pooling operations, resulting in feature maps
3. **Spatial Pyramid Pooling**: In this step, multiple subregions are created with different spatial levels. For each spatial level, the feature maps are pooled separately in a way that the output is a fixed-size vector for each subregion, regardless of the input image's size
    - For level 0, the entire feature map is pooled into a single value (global pooling)
    - Level 1 (gray square in Figure C.14) splits the feature map into $1x1$ segments and processes each segment independently by pooling
    - Similarly, level 2 (light green square in Figure C.14) splits the feature map into $2x2$ segments, pooling each segment independently
    - Similarly, level 3 (blue square in Figure C.14) splits the feature map into $4x4$ segments and processes each segment independently by pooling
4. **Concatenation**: The pooled feature vectors from all spatial levels, including global pooling, are combined to generate the overall feature vector of a fixed length.

Consequently, without scaling or cropping, CNN can handle pictures of various sizes by using spatial pyramid pooling.

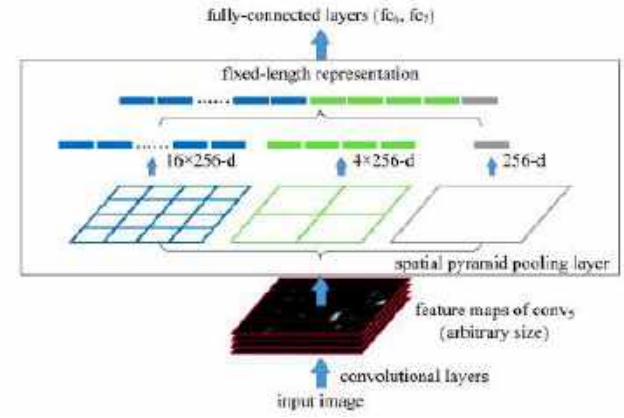

Figure C.14 *Spatial Pyramid Pooling (SPP) applied to network configuration ($conv_5$ represents the terminal convolutional layer with a filter count of 256) (He et al., 2014)*

The YOLOv4 and YOLOv4-p6 networks include the Spatial Pyramid Pooling (SPP) module following the backbone component. This module comprises three Maxpool layers of different sizes (5x5, 9x9, and 13x13) and three route layers for both networks.

C.12 Path Aggregation Network (PANet)

The application of the Path Aggregation Network (PANet) in YOLOv4 and YOLOv4-p6 improves the preservation and use of spatial information for more accurate object detection (Liu et al., 2018). This method integrates parameters from different levels of the backbone network to address different detection stages. It achieves this by feeding information from lower layers upwards (bottom-up) and through adaptive feature pooling. These two features are further outlined below (Parico and Ahamed, 2021):

- Bottom-up path augmentation reduces the length of the information path and enhances the feature pyramid by allowing top layers (the classifiers) to access detailed information from lower layers using route layers
- Adaptive feature pooling restores the distorted flow of information between each segment and all feature levels by merging information from different convolutional layers using element-wise max operation that extracts the maximum value from each corresponding pair of features.

There is also a slight modification to PANet that is applied in the YOLOv4 and YOLOv4-p6 models. As illustrated in Figure C.15, the process of Bottom-up path augmentation involves the application of a concatenation operation rather than the addition of neighboring layers, resulting in an enhancement in the accuracy of the predicted results (Bochkovskiy et al., 2020).

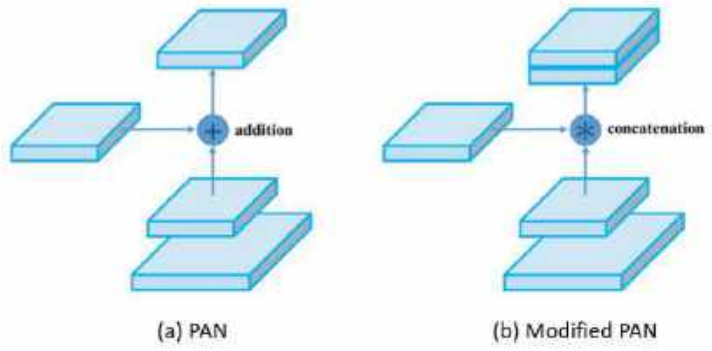

Figure C.15 *Visualization of (a) PAN and (b) modified PAN(Bochkovskiy et al., 2020)*

Appendix D. Combination of UAV datasets

D.1 YOLO format

YOLOv4 and YOLOv4-p6, like all versions of YOLO, requires the dataset to follow a specific format to use the data. The standard method of using the YOLO format involves creating separate text files for each image in the dataset. These files include the class number and spatial information about the objects present in each image. In addition, the annotations in these files are also scaled to be within the range of [0, 1], which makes it easier to handle the data even when the images are resized or modified. Each text file has rows of data, each with the following information:

(<object-class>, <x-center>, <y-center>, <width>, <height>)

Figure D.1 *YOLO format*

where,

- The "object-class" parameter indicates an integer that represents the object's class. The classes are numbered starting from 0 and increase by 1 for each different class in the dataset
- "x-center" and "y-center" parameters refer to the normalized coordinates of the ground truth bounding box center, which are normalized by the image's width and height, with values ranging from 0 to 1. The x and y coordinates are normalized according to the image size:

$$x = \frac{\frac{x_{max+}\, x_{min}}{2}}{image_{width}} \tag{D.1}$$

And

$$y = \frac{\frac{y_{max+}\, y_{min}}{2}}{image_{height}} \tag{D.2}$$

- "width" and "height" parameters represent the normalized width and height of the ground truth bounding box, which surrounds the object, with values also ranging from 0 to 1. These values are normalized according to the image size:

$$widht = \frac{boundingbox_{width}}{image_{width}} \tag{D.3}$$

And

$$height = \frac{boundingbox_{height}}{image_{height}} \tag{D.4}$$

D.2 Modifications for each UAV dataset

This appendix describes the steps for making necessary changes in each UAV dataset individually.

1. **Aerial cars dataset**
   The Aerial Cars dataset includes five different classes: car, truck, bus, minibus, and cyclist, with annotations in YOLO format. Within this dataset, the classes “car”, “truck”, “bus”, and “minibus” are retained, while the “cyclist” class was removed. Following this, the retained classes are converted by using the convert.py script. Specifically, the class "car" was replaced with number 1 to represent small vehicles, and the classes "truck", "bus", and "minibus" were replaced with number 2 to represent large vehicles.
2. **DOTA dataset**
   Each object in the dataset has a shape description called an Oriented Bounding Box (OBB), represented as $(x_1, x_2, x_3, x_4, x_5)$, where (x_i, y_i) is the i-th corner of the OBB. These corners are placed in a clockwise order. Besides the OBB, each object instance is categorized and given a difficulty flag, indicating if it is challenging to detect (1 for difficult, 0 for not difficult). The annotations for each image are stored in a text file with the same name as the image. Each line in the text file corresponds to one object instance. Figure D.2 is an illustration of an image annotation (Xia et al., 2021):

   Example: 558 523 563 546 501 554 501 529 large-vehicle 0

   Figure D.2 *DOTA dataset's annotation format*

   Figure D.2 represents an annotated Oriented Bounding Box defined by points (558, 523), (563, 546), (501, 554), and (501, 529), outlining an OBB that captures a large vehicle in the image. The label assigned to this object instance is "large-vehicle," which specifies its type. Furthermore, the difficulty indicator associated with the detection of this instance is set to 0, showing that it is relatively easy to identify the object in the image.
   To begin with, we converted the unique characteristics of the annotation structure in the DOTA dataset to YOLO format. The code provided in the cited GitHub repository-(ringringyi, 2023) transformed the DOTA into YOLO format. Although we combined DOTA versions 1.5 and 2.0 into a single dataset, each includes different classes and can operate separately. Specifically, DOTA version 1.5 includes sixteen distinct classes while DOTA version 2.0 includes nineteen. Consequently, the conversion process involves retaining only the classes "small vehicle," "large vehicle," and "ship" from both versions. These retained classes were converted using the convert.py script, where the class "small vehicle" was replaced with number 1 to represent small vehicles, the class “large vehicle” was replaced with number 2 to represent large vehicles, and the class “ship” was replaced with number 3 to represent ships.
3. **Visdrone-DET dataset**
   As depicted in Figure D.3, the annotation format used in the Visdrone-DET dataset includes various parameters (VisDrone, 2024). These parameters consist of “bbox_left”, which indicates the x-coordinate of the top-left corner of the predicted bounding box, and “bbox_top”, which shows the y-coordinate of the same corner. Additionally, “bbox_width” represents the width of the predicted bounding box in pixels, while “bbox_height” represents its height in pixels. The “score” parameter

in the "detection" file shows the confidence level associated with the predicted bounding box that contains an object instance. In contrast, in the "groundtruth" file, “score” is binary: 1 indicates inclusion in evaluation, and 0 indicates exclusion. Furthermore, “object_category” specifies one of the ten classes in the VisDrone-DET dataset, as mentioned in Section 3.1. In the "detection" result file, “truncation” is uniformly set to -1, where in the "groundtruth" file, it indicates the degree to which object parts extend beyond the image, with 0 meaning no truncation and 1 indicating partial truncation. Similarly, “occlusion” in the "detection" file is always set to -1, while in the "groundtruth" file, it represents the degree of object occlusion, with 0 indicating no occlusion, 1 indicating partial occlusion, and 2 indicating heavy occlusion.

Example: 755, 468, 16, 50, 1, 1, 0, 0

<bbox_left>,<bbox_top>,<bbox_width>,<bbox_height>,<score>,<object_category>,<truncation>,<occlusion>

Figure D.3 *Visdrone dataset's annotation format*

Figure D.3 shows the Visdrone format, which we converted into the YOLO format by using code the cited GitHub repository-(Tandon, 2024). We kept the classes: "pedestrian", "persons", "car", "van", "truck" and "bus". Then, we used the convert.py script to convert them. In this process, the classes "pedestrian" and "persons" were replaced with number 0 to represent persons, the class “car” was replaced with class number 1 to represent small vehicles and the classes "van", "truck", and "bus" were replaced with number 2 to represent large vehicles.

4. **Stanford dataset**

   Similar to the aerial cars dataset, the annotations in this dataset follow the YOLO format, which eliminates the need for a transformer code. The classes: "pedestrian", "car" and "bus" were included, while "biker", "skater" and "cart" were removed. The retained classes were converted using the convert.py script, where the class "pedestrian" was replaced with number 0 to represent persons, the class "car" was replaced with number 1 to represent small vehicles, and the class "bus" was replaced with class number 2 to represent large vehicles.

5. **DAC dataset**

   The DAC dataset uses the XML format for its annotation structure, because it provides guidelines for representing different types of data. However, unlike regular programming languages, XML does not have built-in functions for computational tasks (Simplilearn, 2023). In the DAC dataset, XML is used to organize and manage data with the help of programming languages or software systems. It uses markup symbols called tags, as shown in Figure D.4, where tags such as “annotation”, “size”, “object”, and “bndbox” can be used to encode information for the following annotation:

<annotation>
<filename>000002</filename>
<size>
<width>640</width>
<height>360</height>
</size>
<object>

<name>boat1</name>
<bndbox>
<xmax>273</xmax>
<xmin>196</xmin>
<ymax>323</ymax>
<ymin>164</ymin>
</bndbox>
</object>
</annotation>

Figure D.4 *DAC dataset's annotation format*

Figure D.4 provides detailed information about the image with file name “000002”. Specifically, it includes the image’s dimensions (640x360), the identification of the object (labelled as boat1), and specific coordinates of its ground truth bounding box (xmax: 273, xmin: 196, ymax: 323, ymin: 164).

After illustrating the application of XML in processing the DAC dataset, we proceed to convert it into YOLO format by using the code from the cited GitHub repository-(Pawar, 2022). We kept the classes: "person", "car", and "boat", while removing all the others. Next, we convert the retained classes using the convert.py script, where the class “person” was replaced with number 0 to represent persons, the class “car” was replaced with number 1 to represent cars and the class “boat” was replaced with number 3 to represent ships.

D.3 Python code for converting the classes in annotation files

The Python script in Figure D.5 is used to change the order of the existing classes in the UAV datasets used in our experiments. Since the five datasets contain different classes, we use this script to convert them as follows: “0” for the person class, “1” for the small vehicle class, “2” for the large vehicle class and “3” for the ship class.

```
#Name: convert.py
import os

# Define the mapping of original classes to new classes
class_mapping = {
 #add classes here, e.g.
   #'0': '0',
   #'1': '0',
   # ...
}

# Define the path to the main folder containing the subfolders
main_folder_path = 'write the folder path here'

# Iterate over each subfolder and file in the main folder
for root, dirs, files in os.walk(main_folder_path):
```

```
for filename in files:
    if filename.endswith('.txt'):
        file_path = os.path.join(root, filename)

        # Read the contents of the file
        with open(file_path, 'r') as file:
            lines = file.readlines()

        # Filter and modify the lines
        modified_lines = []
        for line in lines:
            parts = line.split()

            # Check if the line is empty or doesn't have enough elements
            if not parts or len(parts) < 2:
                continue

            class_label = parts[0]

            # Check if the class should be ignored
            if class_label not in class_mapping or class_mapping[class_label] is None:
                continue

            # Modify the first number in the line
            new_class_label = class_mapping[class_label]
            parts[0] = str(new_class_label)

            # Create a new line with the modified class label
            modified_line = ' '.join(parts) + '\n'
            modified_lines.append(modified_line)

        # Write the modified lines back to the file
        with open(file_path, 'w') as file:
            file.writelines(modified_lines)
```

Figure D.5 *Convert.py script*

Appendix E. Analysis of Variance (ANOVA)

ANOVA includes various statistical methodologies that divide the total variation observed in a dataset into separable systematic and random components. Specifically, systematic factors show significant statistical effects on the dataset being studied, as opposed to random factors, which do not have such effects. In addition, analysts use the ANOVA test to measure the effect that the independent variables have on the dependent variable in regression analysis (Kenton, 2024).

ANOVA is classified into various categories based on the experimental design and the number of independent variables being studied. The following instances are different types of ANOVA:

- One-way ANOVA
- Two-way ANOVA
- Factorial ANOVA
- Repeated Measures ANOVA
- Mixed ANOVA

In this thesis, only one-way Analysis of Variance will be used to analyze the experimental results.

The one-way Analysis of Variance (ANOVA) is used to compare the means of three or more groups. Specifically, it examines if differences in the levels of a single independent variable (referred to as factor) or the interactions between multiple factors have an effect on a dependent variable. Consequently, one-way ANOVA is applicable when there is only one factor and one dependent variable being studied. As it helps to determine if there are significant differences in means between the groups, although it does not specify which specific pairs of groups show these differences (Seltman, 2018).

As mentioned above, one-way ANOVA is a statistical methodology used to determine if there is a difference between the means of three or more groups. It tests the null hypothesis (H_0) that the means are equal against the alternative hypothesis (H_1) that at least one of the means is different. In statistical notation, where "k" is the number of means, the hypotheses can be written as follows:

$$H_0: \mu_1 = \mu_2 = \mu_3 = \ldots = \mu_k \quad \text{(E.1)}$$

$$H_1: not\ all\ means\ demonstrate\ equivalence \quad \text{(E.2)}$$

where, μ_i is the mean value corresponding to the i-th level of the factor.

Since random samples may not accurately reflect entire populations, there is a risk that the means obtained from these samples may not represent the true means of the populations. For this reason, hypothesis testing uses a statistical measure known as the p-value. The p-value measures the probability of observing differences in sample means that are as significant as those observed, assuming that there is no real difference in population means (null hypothesis). If the p-value is less than 0.05, this is considered sufficient evidence to reject the null hypothesis, indicating that there is at least one different mean within the population.