



University of the Aegean

School of Engineering

Department of Financial Management and Engineering

**FINE TUNING OF YOLOv3 TRAINING FOR OBJECT DETECTION IN IMAGES
RECORDED BY A UAV**

Tsiflitzi Anna

Supervisor: Prof. Georgios Dounias

Committee Members: Associate Prof. Vasileios Zeimpekis

Associate Prof. Vasileios Koutras

Chios, January 2025

To my family...

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Professor Ioannis Minis and my supervisor, Professor Georgios Dounias, for their invaluable guidance, inspiration, and support throughout this academic endeavor. Their insights and expertise have been a source of inspiration throughout this journey. Their continuous and constructive feedback on my analysis enabled me to optimize my thesis, while their insightful advice helped me grow personally and pushed me towards my academic excellence.

I am deeply thankful to George Teptaris for his invaluable support through every step of my analysis, experiments, and knowledge sharing on the field. His dedication, technical expertise and willingness to assist have been important in overcoming challenges and achieving meaningful results.

Another special thank you is extended to my colleague, Panos Lapsanis, for his exceptional teamwork, productive collaboration, and fruitful academic discussions. The knowledge exchange and cooperative spirit we shared have not only enriched the outcomes of my thesis but also made this knowledge journey more fulfilling.

Finally, I am deeply grateful to my family and friends for their unwavering support, encouragement, and belief in me throughout this journey of learning and growth. To my parents, who have always been by my side in every step in my life, being my constant source of strength and showing me the value of hard work and patience. Their love and sacrifices have been the foundation of all my achievements. To my sibling, whose cheerfulness and continuous support helped me overcome any challenges, your constant encouragement has meant more than I can express. To my friends, whose encouragement, understanding and patience in my progress kept me motivated. Thank you for being by my side.

Abstract

This thesis focuses on optimizing training of the YOLOv3 (You Only Look Once) model for object detection using UAV (Unmanned Aerial Vehicle)-captured imagery. It showcases the importance of hyperparameter selection in training effectiveness. Specifically, through extensive analysis, we identified important hyperparameters that influence the trained model's performance. By adjusting these hyperparameters we fine-tuned training of the model to achieve higher precision in detecting objects.

The study utilized annotated UAV datasets that were preprocessed to align with YOLOv3's requirements. These datasets were publicly available and comprised of the UA Vehicle Detection Dataset, Stanford Dataset and VisDrone2019DET dataset.

Training optimization was approached by classifying hyperparameters into two categories. The first included hyperparameters that were set according to the characteristics of the training dataset and were kept invariant throughout the analysis. These included max batches, number of classes, filters, and steps. The second category contained the hyperparameters we selected to adjust; i.e., image resolution, backbone network, anchor box dimensions, dilated convolution, box loss and data augmentation techniques.

A Full-Factorial experimental design was employed to generate 96 ($2^5 \times 3$) distinct combinations of these key hyperparameters. The training process was executed twice for each combination of the selected hyperparameters, resulting in a total of 192 trained models. During training, validation was performed every 100 iterations. Finally, after training, we conducted the testing process to evaluate model performance.

Each of the 192 experiments produced outputs consisting of the highest mAP achieved during validation and testing. The results of these experiments were analyzed using ANOVA, which revealed that all hyperparameters significantly influence model performance. Among them, the most impactful hyperparameters are the backbone network, data augmentation and image resolution. Additionally, two significant two-way interactions were observed: a) between the backbone network and data augmentation, and b) between the backbone network and dilated convolution.

The best-performing model achieved mAP values of 60.99% during training/validation and 52.51% during testing. The model that achieved this performance corresponds to the following hyperparameter combination:

- Image Resolution: 832x832
- Dilated Convolution: No
- Box Loss: DIoU
- Anchor Dimensions: Default
- Backbone: Darknet-53
- Data Augmentation: Mosaic

On the other hand, the performance of the worst performing models was very low, indicating that hyperparameter selection and tuning plays an important role and could lead to significant improvements in YOLOv3's detection performance.

The study demonstrates the important role of tailored training processes, dataset preparation, hyperparameter selection and tuning in enhancing YOLOv3's effectiveness for object detection.

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στη βελτιστοποίηση της διαδικασίας εκπαίδευσης του Once) για την ανίχνευση και ταξινόμηση αντικειμένων από εικόνες που καταγράφονται από UAV (Μη Επανδρωμένα Εναέρια Οχήματα) και συγκεκριμένα ανθρώπων, αυτοκινήτων, ποδηλάτων και μεγάλων οχημάτων. Η επίτευξη της βελτιστοποίησης πραγματοποιήθηκε μέσω της ρύθμισης κατάλληλων υπερπαραμέτρων του YOLOv3 μοντέλου, με στόχο τη μεγιστοποίηση της μέσης τιμής της μέσης ακρίβειας ή mAP). Συγκεκριμένα, μέσα από εκτενή ανάλυση, εντοπίστηκαν οι υπερπαραμέτροι που επηρεάζουν την απόδοση του YOLOv3 μοντέλου και μας ενδιαφέρουν για την δική μας έρευνα. Με την κατάλληλη προσαρμογή αυτών των υπερπαραμέτρων, έγινε η σωστή ρύθμιση της εκπαίδευσης του μοντέλου, και αποδείχθηκε ότι ορισμένοι υπερπαραμέτροι επέφεραν υψηλότερα αποτελέσματα mAP συγκριτικά με τις προεπιλεγμένες ρυθμίσεις του YOLOv3.

Στην έρευνα μας χρησιμοποιήσαμε δημόσια διαθέσιμα σύνολα δεδομένων εικόνων από UAV, όπως τα Dataset, το Stanford Dataset και το VisDrone2019DET dataset. Τα συγκεκριμένα αυτά σύνολα δεδομένων εικόνων τροποποιήθηκαν προκειμένου να περιλαμβάνουν μόνο τις κατηγορίες αντικειμένων που μας ενδιέφεραν για την έρευνά μας.

Η βελτιστοποίηση της εκπαίδευσης του YOLOv3 πραγματοποιήθηκε με την ταξινόμηση των υπερπαραμέτρων σε δύο κατηγορίες. Η πρώτη περιλάμβανε υπερπαραμέτρους που καθορίστηκαν σύμφωνα με τα χαρακτηριστικά του συνόλου δεδομένων εκπαίδευσης και παρέμειναν αμετάβλητες καθ' όλη τη διάρκεια της ανάλυσης. Η δεύτερη κατηγορία περιλάμβανε τις υπερπαραμέτρους που ρυθμίστηκαν για τη βελτιστοποίηση της απόδοσης εκπαίδευσης του YOLOv3.

Για τον σχεδιασμό πειραμάτων χρησιμοποιήθηκε η μέθοδος πλήρους παραγοντικού σχεδιασμού (Full-design), δημιουργώντας 96 ($2^5 \times 3$) διακριτούς συνδυασμούς των επιλεγμένων για την έρευνα υπερπαραμέτρων. Η διαδικασία εκπαίδευσης εκτελέστηκε δύο φορές για κάθε συνδυασμό των επιλεγμένων υπερπαραμέτρων, με αποτέλεσμα να δημιουργηθούν συνολικά 192 εκπαιδευμένα μοντέλα. Έπειτα, μετά την εκπαίδευση των μοντέλων, πραγματοποιήθηκε η διαδικασία δοκιμής απόδοσης του μοντέλου (testing) για την αξιολόγηση της απόδοσης του μοντέλου. Σκοπός της εκπαίδευσης και της διαδικασίας δοκιμής απόδοσης είναι να πραγματοποιηθεί η ανάλυση των αποτελεσμάτων μέσω της Ανάλυσης Διακύμανσης (ANOVA).

Η ανάλυση των τιμών του mAP που προέκυψαν από την διαδικασία εκπαίδευσης και δοκιμών μέσω ANOVA αποκάλυψε ότι όλες οι υπερπαραμέτροι επηρεάζουν σημαντικά την απόδοση του μοντέλου. Ωστόσο, οι υπερπαραμέτροι με τη μεγαλύτερη επίδραση ήταν ο κορμός δικτύου (backbone network), η αύξηση δεδομένων (data augmentation) και η ανάλυση εικόνας (image resolution). Επιπλέον, παρατηρήθηκαν δύο σημαντικές αλληλεπιδράσεις μεταξύ παραγόντων: α) μεταξύ του κορμού δικτύου (backbone network) και της αύξησης δεδομένων (data augmentation), και β) μεταξύ του κορμού δικτύου (backbone network) και της διατεταμένης συνελκτικής (dilated convolution).

Το καλύτερο εκπαιδευμένο μοντέλο πέτυχε τιμές mAP 60,99% κατά την εκπαίδευση/επικύρωση και 52,51% κατά τη δοκιμή απόδοσης. Το μοντέλο που πέτυχε αυτήν την απόδοση αντιστοιχεί στον ακόλουθο συνδυασμό υπερπαραμέτρων:

- Ανάλυση εικόνας (Image Resolution): 832x832
- Διατεταμένη Συνελικτική (Dilated Convolution): Όχι
- Απώλεια Κουτιού (Box Loss): DIoU
- Διαστάσεις των περιγραμμάτων (Anchor Dimensions): Default
- Κορμός Δικτύου (Backbone Network): Darknet-53
- Αύξηση Δεδομένων (Data Augmentation): Mosaic

Από την άλλη πλευρά, η σημαντικά χαμηλή απόδοση ορισμένων μοντέλων, υποδεικνύει ότι η επιλογή και ρύθμιση των υπερπαραμέτρων παίζει σημαντικό ρόλο και μπορεί να οδηγήσει σε σημαντικές βελτιώσεις στην αποτελεσματικότητα του YOLOv3 για την ανίχνευση και ταξινόμηση αντικειμένων.

Table of Contents

Chapter 1 Introduction	13
Chapter 2 Understanding object detection and YOLO	15
2.1 Fundamentals of object detection	15
2.1.1 Overview of computer vision	15
2.1.2 Definition and purpose of object detection	15
2.1.3 Key components of object detection systems	16
2.1.4 Object detection approaches (R-CNN, SSD, YOLO)	18
2.2 Evolution of YOLO algorithms and key features	21
2.2.1 Introduction to YOLO (You Only Look Once) algorithms	21
2.2.2 Overview of YOLOv1 and YOLOv2	22
2.2.3 Innovations in YOLOv3 and its significance	23
2.2.4 Concluding remarks	24
Chapter 3 Deep Dive into YOLOv3	25
3.1 YOLOv3 architecture overview	25
3.2 The YOLOv3 architecture	27
3.2.1 Darknet-53 backbone and feature extraction	27
3.2.2 Feature Pyramid Network (FPN)	30
3.2.3 Detection heads	33
3.3 Training, validation and testing	35
3.3.1 Training process of YOLOv3	36
3.3.2 Validation process of YOLOv3	40
3.3.3 Testing process of YOLOv3	41
3.4: Operation of YOLOv3	45
Chapter 4 Data preparation and parameter selection for training the YOLOv3 algorithm	47
4.1 Data collection and annotation	47
4.1.1 Data collection	47
4.1.2 Annotation and consolidation	49
4.2 Experimental set up	50
4.3 YOLOv3 hyperparameters	51
4.3.1 Hyperparameters determined by the characteristics of the dataset	52
4.3.2 Hyperparameters associated with the YOLOv3 architecture and functionality	56
4.3.3 Hyperparameters selection for our research	60

Chapter 5 Experimental Analysis	73
5.1 Full Factorial Design	73
5.2 Experimental set-up and execution	76
5.3 Experimental results and analysis	81
5.4 Hyperparameter effects on mean Average Precision (mAP)	88
5.4.1 ANOVA analysis on best mAP results from the training/validation process	89
5.4.2 ANOVA analysis for the testing process	106
5.4.3 Similarities and differences in the analysis results of validation vs. testing	114
5.4.4 Evaluation Metrics for YOLOv3: Performance	115
Chapter 6 Conclusions	118

Table of Figures

Figure 2.1 Object detection in visual recognition (Pathak et al., 2018)	16
Figure 2.2 Use of Convolutional Neural Network for object detection (Pathak et al., 2018)	17
Figure 2.3 Architecture of SSD algorithm (Rohan et al., 2019)	19
Figure 2.4 Two-stage object detectors R-CNN (Diwan et al., 2023)	20
Figure 2.5 Two-stage object detectors Faster R-CNN (Diwan et al., 2023)	20
Figure 2.6 Architecture of YOLOv1 algorithm (Redmon et al., 2016).....	21
Figure 3.1 YOLOv3 network architecture (Palma, 2020)	26
Figure 3.2 Darknet-53 architecture (Ma et al., 2020)	27
Figure 3.3 A 3x3 kernel (per channel) slides over the input to generate the output(Tepteris et al., 2023).....	28
Figure 3.4 YOLOv3 Residual block structure (Xu & Wu, 2020)	30
Figure 3.5 Network architecture of feature pyramid network (FPN) (Zhang et al., 2021).....	31
Figure 3.6 Upsampling layer (Tepteris et al., 2023).....	32
Figure 3.7 Concatenation of two inputs (Tepteris et al., 2023).....	32
Figure 3.8 YOLOv3 Output vector per anchor in each cell (Tepteris et al., 2023)	34
Figure 3.9 Training process of YOLOv3 (Tepteris et al., 2023)	36
Figure 3.10 Computing Intersection over Union (IoU) (Padilla, Netto, & Silva, 2020)	37
Figure 3.11 IoU of bounding boxes (Kamal, 2019)	37
Figure 3.12 YOLOv3 detection with class score (Shivaprasad, 2019).	39
Figure 3.13 YOLOv3 prediction example (Gilbert, 2020).....	39
Figure 3.14 Validation process of YOLOv3.....	40
Figure 3.15 Testing process of YOLOv3 (Tepteris et al., 2023)	41
Figure 3.16 Operation process of YOLOv3 (Tepteris et al., 2023)	46
Figure 3.17 Image after the applying of YOLOv3 object detection algorithm (Cruz Martinez, 2021).....	46
Figure 4.1 The number of training, validation and testing images included in the consolidated dataset..	50
Figure 4.2 Hardware Configuration of the system	50
Figure 4.3 Software components of the system	51
Figure 4.4 Illustration of "classes" adjustment in the configuration file	53
Figure 4.5 Illustration of "max_batches" in the configuration file	54
Figure 4.6 Illustration of "filters" in the configuration file	55
Figure 4.7 Illustration of "steps" in the configuration file.....	56
Figure 4.8 Illustration of the three different image resolution options in the configuration file.....	61
Figure 4.9 Illustration of the default anchor dimensions for the YOLO head responsible for detecting small objects	61
Figure 4.10 Illustration of the default anchor dimensions for the YOLO head responsible for detecting medium objects	62
Figure 4.11 Illustration of the default anchor dimensions for the YOLO head responsible for detecting large objects.....	62
Figure 4.12 Illustration of the updated anchor dimensions in the configuration file	63
Figure 4.13 Illustration of mosaic augmentation (Alexey, 2020).....	65
Figure 4.14 Illustration of "mosaic" in the configuration file.....	66
Figure 4.15 Dilated convolution filters with dilation rates $D = 1$, $D = 2$, $D = 3$ respectively (Heffels and Vanschoren, 2020)	67

Figure 4.16 Illustration of the DIOU in the configuration file	70
Figure 5.1 Configuration files (.cfg) along with the corresponding data files (.data).....	76
Figure 5.2 Variables from the first .data file corresponding to the first .cfg file in our experiments.....	76
Figure 5.3 Txt file including the object classes.....	76
Figure 5.4 Execution command for conducting the experiments	77
Figure 5.5 Information included in all .data files used for testing	79
Figure 5.6 Execution command for the testing process of our experiments	79
Figure 5.7 Training progress chart of the 90th experiment	87
Figure 5.8 Training progress chart of the 74th experiment	87
Figure 5.9 Pareto chart of the standardized effects during training process.....	96
Figure 5.10 Main effects plot for mAP during training process	98
Figure 5.11 Interaction plot for mAP during training process.....	98
Figure 5.12 Pareto chart of the standardized effects during testing process	112
Figure 5.13 Main Effects Plot for mAP during testing process	114

List of Tables

Table 2.1 Pooling layers used for object detection (Pathak et al., 2018)	17
Table 2.2 Evolution of YOLO Algorithms (Alexey, 2024)	24
Table 4.1 The number of training, validation and testing objects in the consolidated dataset.....	49
Table 4.3 Hyperparameters in the backbone network related to architecture	56
Table 4.4 Hyperparameters in the backbone network related to training	57
Table 4.5 Hyperparameters in the YOLO heads related to architecture	57
Table 4.6 Hyperparameters in the YOLO heads related to architecture related to training	58
Table 4.7 Default and updated values of hyperparameters selected for our research.....	60
Table 4.8 Representation of the use of the k-means algorithm.....	62
Table 4.9 Anchor boxes for Darknet-53.....	63
Table 4.10 Anchor boxes for Resnet-152.....	64
Table 4.11 Illustration of 'dilation' in the configuration file.....	68
Table 4.12 Hyperparameter values for Darknet-53.....	71
Table 4.13 Hyperparameter values for Resnet-152.....	71
Table 5.1 Multilevel factorial design of our study.....	73
Table 5.2 Key class metrics during validation.....	80
Table 5.3 Best mAP and the average best mAP in the 1 st and 2 nd run of the training and testing processes	82
Table 5.4 Number of objects in the training, validation and testing datasets	84
Table 5.6 Analysis of Variance during training process	91
Table 5.7 The best and worst hyperparameters' interactions on the mAP.....	99
Table 5.8 Analysis of Variance during testing process.....	107
Table 5.9 Comparison of the effects of the main factors on mAP between validation and testing. The values indicate the difference between High and Low	114
Table 5.10 Evaluation metrics of YOLOv3 90 th experiment	115
Hyperparameters.....	52

Chapter 1 Introduction

In the broad domain of computer vision, object recognition and object detection are two significant and distinct areas. Object recognition is the process of identifying and categorizing predefined objects, in the classes of interest, within an image based on their visual features. It determines the classes of objects present in an image without providing detailed information about their location. On the other hand, object detection recognizes the objects in the image (or video stream) and encloses each within an appropriate bounding box. Thus, in addition to classifying the objects in the image, it also determines the position and size of each object (Voulodimos et al., 2018).

In this thesis, we analyze the YOLOv3 algorithm within the specific context of UAV-captured images, aiming to better understand the related training process and improve it through available training parameters that may be tuned. YOLOv3 (You Only Look Once) is a single-pass object detection algorithm recognized for its real-time processing capabilities and high accuracy (Kamal, 2021). The scope of this research involves a detailed exploration of the YOLOv3 architecture, and its distinguishing features compared to its predecessors, and an examination of how various training hyper-parameters influence its performance in the recognition and localization of objects in UAV imagery.

To achieve these objectives, we begin by explaining how YOLOv3 differentiates itself from earlier one-stage and two-stage object detectors. We then present an in-depth analysis of YOLOv3's architecture, focusing on its convolutional layers, anchor boxes, and detection techniques. The study preparation phase involved selecting appropriate UAV datasets and setting up the necessary hardware and software for our experiments. We introduced appropriate classes in the selected datasets, modified and organized the raw and unprocessed data appropriately to support our research needs.

For the hyperparameter study, we performed necessary modifications of the YOLOv3 configuration files to be able to adjust systematically the hyperparameters during training. The first step of this study included training and validation experiments using UAV datasets that were purposely selected. Subsequent, we systematically varied key training hyperparameters such as image resolution, activation functions, anchor dimensions, backbone architecture, data augmentation strategies, and the incorporation of dilated convolutions. These parameters were selected from a larger set based on a thorough examination of YOLOv3 characteristics, architecture and training possibilities. After the execution of 96 training tests, the best value of mean Average Precision (mAP) achieved was 60.89%. By using a new UAV dataset (deopsys dataset), that was created by our lab, we tested the best weights of the trained YOLOv3 model. The highest tests results provided a mean Average Precision (mAP) of 84.59%. This specific mAP percentage was noted for a height of 15m, and various lighting conditions.

The results of this thesis highlight the impact of key training hyperparameters on the YOLOv3 algorithm's performance. Through detailed analysis, we identified how adjustments to image resolution, activation functions, anchor dimensions, and other factors affect the algorithm's accuracy and speed in detecting objects in UAV-captured images. These findings offer guidelines for enhancing the YOLOv3 training process, ultimately advancing its effectiveness in UAV image recognition and localization tasks.

The structure of the remainder of this thesis is as follows: In Chapter 2, we will delve into how YOLOv3 distinguishes itself from its predecessors by comparing one-stage and two-stage detectors. In Chapter 3, we provide an in-depth analysis of the YOLOv3 architecture, examining its key components and mechanisms. In Chapter 4, we detail the preparation of UAV datasets and the experimental set up, including the dataset selection process, hardware, and software setup, as well as the procedures we followed for data preparation. This chapter also covers a detailed analysis of the parameters that were selected for our research. Chapter 5 focuses on analyzing the impact of key hyperparameter on YOLOv3's training and performance in recognizing and localizing objects in UAV imagery. Finally, Chapter 6 presents the conclusions of this work and provides insights into possible future research directions.

Chapter 2 Understanding object detection and YOLO

In this chapter, we will go through the fundamental elements of object detection as well as the evolution of YOLO algorithms over time. Beginning with an overview over computer vision and an analysis of the purpose and the key components of object detection. Following up, we will explore the object detection approaches, including R-CNN (Region-based Convolutional Neural Network), SSD (Single Shot Detection), and YOLO (You Only Look Once). In the next subsection, we will focus on the evolution of YOLO algorithms and their key features. We will go through a detailed overview of the initial versions, YOLOv1 and YOLOv2, highlighting their key features, innovations, and the improvements that each version brought to address existing limitations, concluding to YOLOv3 and its significance over its predecessors.

2.1 Fundamentals of object detection

In this Section, we aim to gain insights into the evolution and current state-of-the-art techniques in the field of object detection. We will define the concept of object detection, highlighting its purpose and its importance. Additionally, we will explore the key components of object detection systems, clarifying the processes involved in identifying and localizing objects within images. Moreover, we will discuss object detection approaches, including R-CNN (Region-based Convolutional Neural Network), SSD (Single Shot Detection), and YOLO (You Only Look Once), focusing on the respective methodologies and contributions to advancing object detection technology.

2.1.1 Overview of computer vision

Computer Vision (CV) is a cross-disciplinary field that combines computer science and image processing. More particularly computer vision provides computing systems with the ability to extract high level understanding from digital images and video streams. It tries to give machines the ability to understand visual data in a way similar to humans. Over the decades, CV has undergone a revolutionary transition from early image processing algorithms and manually constructed features to the current deep learning approaches, particularly Convolutional Neural Networks (CNNs). This radical change enables computers to autonomously learn hierarchical representations directly from annotated raw data, leading to significant breakthroughs in tasks such as object detection, image classification, and semantic segmentation. Despite considerable progress, there are still many challenges including maintaining model robustness, resolving ethical concerns, and improving clarity. Current research aims to make deep learning models more adaptable to real-world circumstances, and able to address more complex problems (Bi et al., 2023; Voulodimos et al., 2018).

2.1.2 Definition and purpose of object detection

Object detection is a key computer vision task that detects and marks semantic objects of defined classes (such as humans, cars, or birds) in digital images and videos. It is different from image classification, which only assigns labels to the image. Object detection also defines the exact boundaries of each object, using bounding boxes. So, it not only identifies the class instance of any object within an image, but also encloses it in a bounding box, thus determining the object's location within the image and its size. When the object detection task searches for a single class instance in an image, it is called single class object detection.

When it searches for all defined class instances of the objects in an image, it is called multi class object detection.

The primary purpose of object detection is to provide machines with the capability of understanding visual scenes mirroring human cognitive processes. For example, object detection enables the identification of pedestrians, vehicles, and other objects, thereby facilitating real-time decision-making to complex problems, such as real life-like safe navigation. Object detection is essential in applications that need accurate object recognition and localization, such as autonomous cars, surveillance systems, medical imaging, and augmented reality. The overarching goal is to improve the capabilities of intelligent systems in interpreting and understanding the visual world (Pathak et al., 2018).

2.1.3 Key components of object detection systems

Object detection systems use various techniques and components to achieve accurate and efficient detection. However, there are basic key components which are similar in every object detection system, and they are important parts of its framework.

The procedure begins with obtaining input data, typically comprising images or video frames. Preprocessing methods such as resizing, normalization, and data augmentation are employed to optimize quality and facilitate subsequent analysis. Deep neural networks, particularly Convolutional Neural Networks (CNNs), are utilized for feature extraction. More specifically CNN extracts the features of the image into a “feature map”, which is the outcome of applying a filter in the output of the previous layer. After passing from a number of layers, the result of the process is to obtain several sets of extracted “feature maps” of different sizes. The framework also involves object classification and bounding box regression. Object classification requires assigning a class label to each object. Simultaneously, bounding box regression fine-tunes the spatial coordinates of proposed bounding boxes, improving the accuracy of object localization. To reduce redundancy and eliminate overlapping predictions, a non-maximum suppression (NMS) step is introduced, which makes it easier to maintain the most confident and non-overlapping object detections.

The comprehensive training of the framework requires labeled datasets, where each object is annotated with a class label and a bounding box. The training process involves adjusting the model’s parameters to optimize them with the use of a loss function that measures the difference between predicted outputs and ground truth labels (Papageorgiou et al., 1998). A step-by-step procedure of the object detection process is overviewed in Figure 2.1.

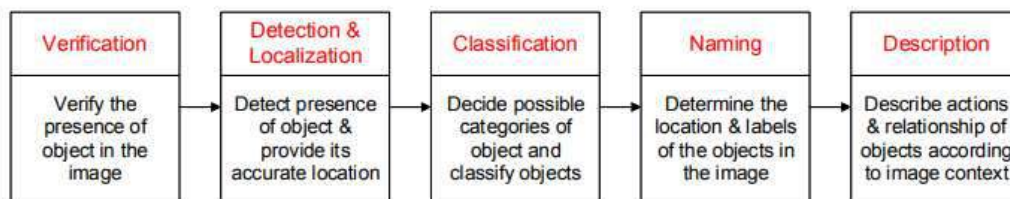


Figure 2.1 Object detection in visual recognition (Pathak et al., 2018)

As already mentioned, CNNs play an important role in all CV systems: they are responsible for extracting the features of an image into a “feature map”. A “feature map” is basically the outcome of applying a filter in the output of the previous layer. After repeating this process multiple times, the result is several sets of extracted “feature maps” in different sizes. As illustrated in Figure 2.2, the layered architecture of CNNs for object detection involves input images with activation functions to generate feature maps. Subsequently, pooling layers are applied to abstract these feature maps and reduce spatial complexity. This process is iterated across multiple filters to create diverse feature maps. Finally, fully connected layers process these feature maps to produce output images with confidence scores for predicted class labels (Pathak et al., 2018).

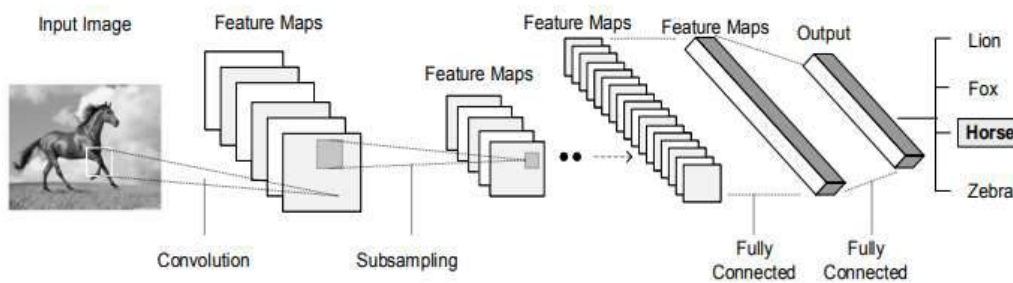


Figure 2.2 Use of Convolutional Neural Network for object detection (Pathak et al., 2018)

To mitigate network complexity and reduce the number of parameters, CNN employs various types of pooling layers, as described in the table (Table 2.1) below.

Table 2.1 Pooling layers used for object detection (Pathak et al., 2018)

Pooling layer	Description
Max pooling	It is a widely used pooling mechanism in CNNs. Max pooling selects the maximum value from the result of the convolution operation which is applied to the input feature map. After a convolutional layer processes an image and produces a feature map, the max pooling layer scans over small regions within this feature map and keeps only the highest value from each region, resulting in detecting the most important feature in the area.
Average pooling	Average pooling calculates the average value within each region, as it considers all values in the region rather than focusing on the most important one. This results in a more generalized feature representation of the input features.
Spatial pyramid pooling (He et al., 2015)	This pooling mechanism performs down-sampling of the image and produces a feature vector with a fixed length. This feature vector can be used for object detection without making any deformations on the original image.
Scale dependent pooling (Yang et al., 2016)	This pooling mechanism handles scale variation in object detection and helps to improve the accuracy of detection.

2.1.4 Object detection approaches (R-CNN, SSD, YOLO)

The development of object detection algorithms has attracted intense interest during recent years. The most known algorithms include Single Shot Detection (SSD), Regional CNN (R-CNN), Faster R-CNN and the You Only Look Once (YOLO) algorithms. These algorithms specify the coordinates of bounding boxes around the objects and provide at the same time the exact location of the object regarding the bounds of the image, with the intention of classifying the objects inside the image. The YOLO family and SSD are representative of one-stage detectors, while the R-CNN family is an example of two-stage detectors.

Starting with the analysis of the one-stage detectors, Single Shot Detection (SSD) is an object detection technique developed by Google and introduced by Wei Liu in 2016 (Liu et al., 2016). The SSD model is a single-stage object detection network, as it executes object detection in a single “pass” through the network, that enhances both detection speed and accuracy. It uses a Convolutional Neural Network (CNN) to process images and produce a feature map, which is a simplified version of an image that highlights important details. The SSD model consists of three main components:

- 1) the backbone network, which extracts key features from the image
- 2) the bounding box creation, which generates potential boxes around objects
- 3) the convolutional prediction, in which the model generates potential boxes around objects based on the extracted features. (Shuai and Wu, 2020).

A distinctive characteristic of SSD is its capability to predict bounding boxes at multiple stages within the network. To achieve this, a series of convolutional layers with a small kernel size 3x3 are applied. These convolutional layers are designed to focus on objects of various sizes, including small, medium, and large ones. As the network progresses through these convolutional layers, it gradually reduces the image's resolution, effectively enabling the extraction of all the image details (Liu et al., 2016). This multi-scale feature extraction is crucial for identifying objects of different sizes.

In the later stages of convolutional operations, SSD accomplishes two critical tasks simultaneously. First, it generates classification probabilities for each detected object, determining the type of object present in the image. Second, it computes the coordinates of bounding boxes, localizing the detected objects within the image.

To optimize object detection results, SSD incorporates a non-maximum suppression step. This step ensures that only the most suitable bounding boxes are retained for each detected object, reducing redundancy and overlapping bounding boxes. In summary, SSD integrates feature extraction, object classification, and bounding box prediction within a single detection through a CNN. It is applied in different scales to adapt on objects of varying sizes and employs non-maximum suppression to produce precise and reliable object detection outcomes (Shuai and Wu, 2020).

In Figure 2.3 is illustrated the SSD architecture. Starting at the left side with the input image, this image is processed by the base network, which is shown as a large block next to it, which is a pre-trained CNN named VGG-16. This network extracts feature maps from the image, capturing different levels of complexity in an image, from basic to more complex objects. Next to the base network, we see several additional feature layers, each represented by blocks at various resolutions. These layers process the feature maps further to detect objects at different scales. Each feature layer is connected to detection

heads, which are convolutional layers responsible for predicting bounding boxes and class scores for objects. In the right side of the image, the last step labeled as Non-Maximum Suppression (NMS) is responsible for filtering the overlapping boxes to output the most accurate detections.

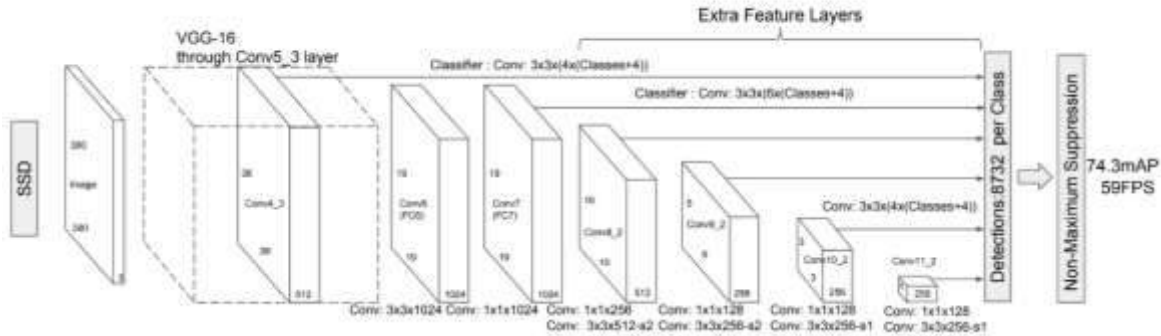


Figure 2.3 Architecture of SSD algorithm (Rohan et al., 2019)

YOLO, which stands for "You Only Look Once," is a technology used for detecting objects in images. Unlike older methods that involve multiple steps (first finding possible object areas and then classifying them), YOLO processes the entire image in one single step. This approach allows YOLO to quickly identify and locate objects like people, cars, animals etc. with great speed and accuracy. The key to YOLO's performance lies in its architecture. YOLO uses a single convolutional neural network (CNN) to divide the image into a grid and simultaneously predict bounding boxes and class probabilities for each grid cell. This means that YOLO doesn't need separate stages for object detection and classification, which speeds up the whole process. The network is designed to predict multiple bounding boxes and object classes in one forward pass, making it ideal for real-time applications. Additionally, YOLO excels in detecting small objects, which can be challenging for other models. It achieves this through its detailed grid-based approach, which helps it focus on smaller details within each part of the image. YOLO's design also includes features like anchor boxes and advanced techniques for managing overlapping objects, enhancing its accuracy and efficiency. managing overlapping objects effectively. (Jiang et al., 2022). A more detailed understanding of the YOLO algorithm will be given in the next chapter.

R-CNN is an object recognition model that follows a multi-step process (Girshick et al., 2014) and it is an example of a two-staged detector (see Figure 2.4.) Initially, it estimates potential object positions within an image, and then, it performs object classification. The estimation of object positions relies on a selective search algorithm which generates approximately 2000 region proposals, each representing a potential object location within the image (Uijlings et al., 2013). These region proposals are then inserted into a Convolutional Neural Network (CNN) to extract image features, resulting in 4,096-dimensional feature vectors for each proposal. A feature vector, whose dimensions came up after experiments and tuning during the development of the model, because it provides greater accuracy in object detection tasks. The extracted features from the CNN are further processed by a Support Vector Machine (SVM) algorithm, primarily used for classification tasks. The SVM's goal is to distinguish different object categories (Teptaris et al., 2023).

R-CNN has some limitations, including being slow and computational heavy. It processes each region proposal separately, which takes a lot of time and resources. Additionally, R-CNN faces difficulties in adapting to different image patterns effectively due to the fact that inherits patterns slower. This slowness comes from the need to process the 2,000 region proposals, since the feature extractor must repeatedly perform the same task for each of these regions. Another issue is the dependency on a fixed algorithm during the selective search, preventing the network from learning patterns within the image. This limitation stems from the fact that the algorithm combines similar regions into large ones, potentially resulting in the creation of inferior region proposals. To cope with these inefficiencies, an improved model was developed, known as Faster R-CNN (Ren et al., 2015). Unlike its predecessor, Faster R-CNN no longer employs the Selective Search method for generating region proposals. Instead, the model is trained to predict region proposals using a Convolutional Neural Network (CNN), as illustrated in Figure 2.5. These predicted region proposals are then fed into separate CNNs to determine the presence of objects of interest within these regions. Faster R-CNN outputs both the object class and its position within the image, marking a significant improvement in efficiency and accuracy compared to the original R-CNN model (Ren et al., 2015).

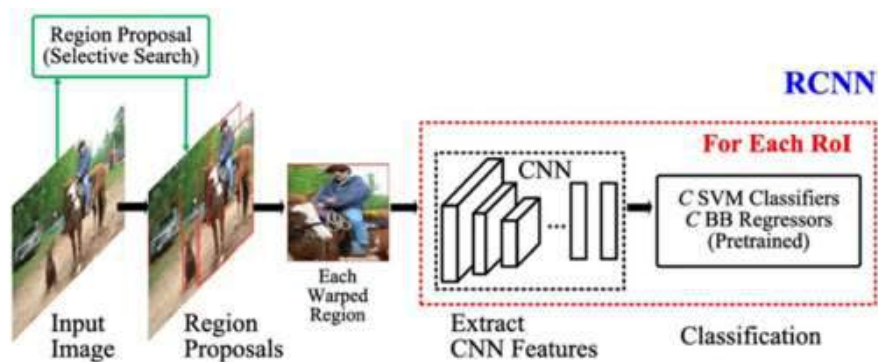


Figure 2.4 Two-stage object detectors R-CNN (Diwan et al., 2023)

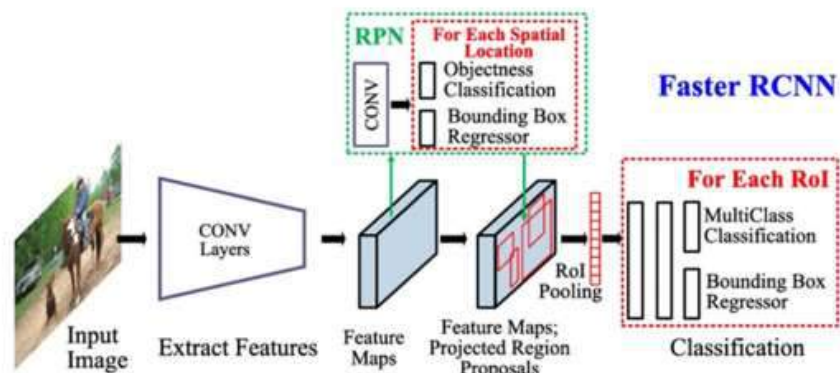


Figure 2.5 Two-stage object detectors Faster R-CNN (Diwan et al., 2023)

2.2 Evolution of YOLO algorithms and key features

The "You Only Look Once" (YOLO) algorithm has significantly impacted the field of object detection, setting new standards for speed and accuracy. This chapter is divided into two sections and attempts to provide an understanding of this technology. The first section introduces the foundational concepts behind the YOLO algorithm, explaining briefly how it transforms object detection into a single regression problem. The second section offers a detailed overview of the initial versions, YOLOv1 and YOLOv2, highlighting their key features, innovations, and the improvements that each version brought to address the limitations of its predecessor.

2.2.1 Introduction to YOLO (You Only Look Once) algorithms

The YOLO algorithm is an open-source object detection technique that employs convolutional neural networks (Redmon et al., 2016). Its core strength lies in its small model size, enabling fast calculations. YOLO directly outputs bounding box positions and categories through a single neural network, facilitating real-time detection, including video processing. This single-stage detection architecture (Figure 2.3) treats object detection as a regression problem applied on the whole image.

Firstly, the YOLO algorithm imposes to the input picture a grid of $S \times S$ cells. The size of this grid may differ. For example, grids of sizes 3×3 , 5×5 , 19×19 may be used. Each cell within a grid assesses independently the presence of an object, its size and class. The aim of these operations is the creation of bounding boxes. The generation of bounding boxes is followed by the creation of an estimation vector for each grid, which encapsulates significant metrics. These metrics are the confidence score, B_x (x coordinate of the object's midpoint), B_y (y coordinate of the midpoint of the object), B_w (w the width of the object), B_h (h the height of the object) and the dependent class probability (Atik et al., 2022). However, YOLO has some limitations. The first YOLO version can only detect 49 objects and if objects are small there are many possibilities of not been detected. Another issue is the inaccurate localization, in many cases the model faces difficulties in localizing precisely an object. To address these issues, newer versions improve the YOLO algorithm, both in quality and speed (Jiang et al., 2022).

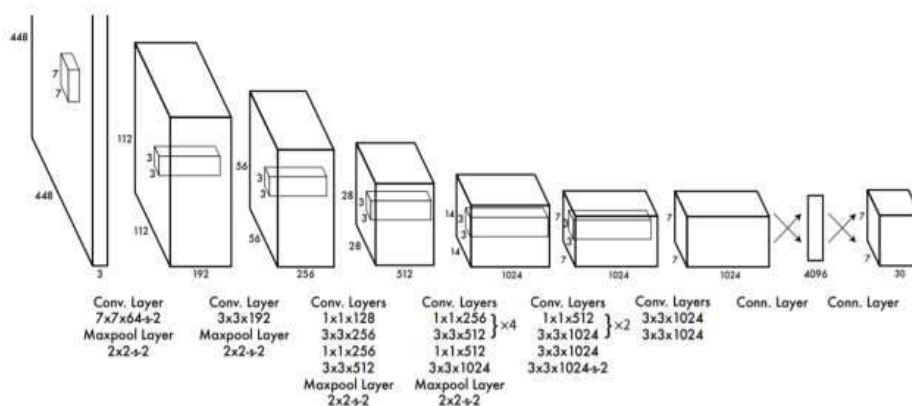


Figure 2.6 Architecture of YOLOv1 algorithm (Redmon et al., 2016)

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.

2.2.2 Overview of YOLOv1 and YOLOv2

Below is provided a brief description of the two YOLO versions. Readers that are interested to further details may refer to (Redmon et al., 2016; Redmon and Farhadi, 2017).

The first version of YOLO uses a 7x7 grid, with the restriction of any grid cell to be able to detect only one object. This is the reason why YOLOv1 can detect maximum 49 objects. At the same time, YOLOv1 was trained to detect only 20 different classes, so for any grid cell it will output 20 class probabilities, one for each class. Although each grid offers the option of two bounding boxes, the process continues only with the boxes that have higher confidence score. This architecture yields an output of $S \times S \times (B \times 5 + C)$, where B represents the number of bounding boxes that each grid cell predicts, and C represents the number of object classes the network can detect. Therefore, for $S=7$, $B=2$ and $C=20$, the final output of the network will be 7x7x30 tensor of predictions (Figure 2.6).

The model comprises 24 convolutional layers followed by 2 fully connected layers, employing ReLU activation function except for the final layer, which utilizes linear activation. Pre-training on the ImageNet dataset and fine-tuning on PASCAL VOC datasets enhanced the performance of YOLOv1, reducing localization errors compared to other methods. Nonetheless, YOLOv1 faces some limitations based on the number of detected objects, high localization error and inability of detecting smaller objects. That were some of the reasons for releasing the next version (Atik et al., 2022).

YOLOv2, an advanced version of YOLO, introduces improvements in localization and recall ability while preserving classification accuracy. YOLOv2 simplifies the architecture and employs non-max suppression to select bounding boxes with the highest Intersection Over Union (IOU) (Jiang et al., 2022). The algorithm focuses on better and faster detection, emphasizing the handling of large and small objects through the design of an effective loss function. YOLOv2 maintains its performance while improving the mean Average Precision (mAP). Through the introduction of new features, the YOLOv2 model adapts to different image sizes, offering a balance between speed and accuracy.

Significant enhancements include the integration of batch normalization for input data preprocessing, which is used in neural networks to standardize the inputs to each layer. The addition of batch normalization leads to an improvement in mAP by 2%. Another improvement refers to the adoption of a high-resolution classifier from 224x224 to 448x448 for detection, yielding a 4% rise in mAP. YOLOv2 introduces for the first time anchor boxes. Instead of predicting the coordinates of bounding boxes directly using fully connected layers on top of convolutional feature extractor, YOLOv2 uses Faster R-CNN, which predicts bounding boxes using hand-picked priors ((Ren et al., 2015). YOLOv2's architecture is based on the usage of a new network which works in a "network in network" concept, that has a new classification model as a backbone network, Darknet-19, and 5 max-pooling layers. It also utilizes fewer filters. YOLOv2 has 25 convolutional layers instead of 24 of the first YOLO version (Redmon and Farhadi, 2017).

The utilization of convolutional layers with anchor boxes contributes to a higher performance. Additionally, the introduction of multi-scale output offers to the network the ability to detect objects at different scales or resolutions. This means that YOLOv2 can detect small objects as well as larger ones in the same image (Jiang et al., 2022). All these advances make YOLOv2 a more accurate model for object detection in comparison with its predecessor.

2.2.3 Innovations in YOLOv3 and its significance

YOLOv3, the third version of the You Only Look Once (YOLO) algorithm, represents a notable evolution from its predecessor, YOLOv2, and introduces several key differences. A considerable improvement lies in the refined network architecture, using three distinct detection heads. The latter are added to YOLOv3's architecture offering the algorithm the ability to classify small, medium, and large objects respectively, improving accuracy for objects of varying sizes.

Furthermore, YOLOv3 uses a new backbone network with 53 convolutional layers, called Darknet-53, which is a hybrid approach between Darknet-19 (YOLOv2's network) and the residual network, providing more speed to the algorithm. Another significant improvement includes the adoption of a feature pyramid network (FPN), enabling the algorithm to capture object details at multiple resolutions.

YOLOv3 also integrates skip connections to access features from earlier layers, enhancing its contextual understanding (Jiang et al., 2022). By using independent classifiers, YOLOv3 adds the ability to classify the detected object in a bounding box to more than once classes. More specifically, during training, a binary cross-entropy loss function is used for class prediction.

Additionally, YOLOv3 embraces the use of multiple anchor boxes per grid cell, facilitating more precise object localization. Another important enhancement is the addition of a confidence score, determined through logistic regression, for each bounding box prediction. This score is expected to be 1 if the bounding box effectively covers a ground truth box than any other bounding box prior, based on the highest Intersection over Union (IOU). The system ensures that only one bounding box prior is assigned to each ground truth box. If the box does not have the highest IOU but does overlap a ground truth box by more than a threshold (0.5), the prediction is disregarded (Kamal, 2021). These innovations contribute to YOLOv3's better performance in real-time object detection tasks.

Table 2.2 below illustrates the differences between the three YOLO versions: The parameters that were changed or added in the algorithm's architecture, as well as those related to the training procedure.

Table 2.2 Evolution of YOLO Algorithms (Alexey, 2024)

Evolution of YOLO Algorithms and Key Features			
Parameters for Backbone Network	YOLOv1	YOLOv2	YOLOv3
Architecture			
Backbone Depth	24	19 (Darknet-19)	53 (Darknet-53)
Training			
Input Size	224x224	224x224	variable sizes
Input Normalization	No	Yes	Yes
Data Augmentation	No	Yes	Yes
Multi-scale Training	No	Yes	Yes
Parameters for YOLO Network			
Architecture			
Number of layers	24	25	53
Spatial Pyramid Pooling (SPP)	No	No	Yes
Multi-scale Output	No	Yes	Yes
Training			
Anchor Boxes per cell of the grid	0	5	9
Batch Normalization	No	Yes	Yes

2.2.4 Concluding remarks

When considering YOLOv3 over its predecessors, YOLOv1 and YOLOv2, several key advancements distinguish it as a superior choice in various object detection applications. YOLOv3 introduces significant improvements in detection accuracy, speed, and versatility compared to its predecessors. Through the adoption of a deeper network architecture, YOLOv3 achieves enhanced detection performance, particularly in detecting small objects and handling object occlusion. Additionally, YOLOv3 incorporates a feature pyramid network (FPN) and utilizes multiple scale detections, enabling the model to effectively capture objects at different scales and resolutions. Furthermore, YOLOv3 introduces the use of anchor boxes to improve bounding box predictions, offering greater flexibility and accuracy in object localization. Notably, YOLOv3 maintains a remarkable balance between detection accuracy and speed, making it well-suited for object detection. Based on all these advantages we proceed in the selection of the YOLOv3 algorithm for our experimental research (Redmon and Farhadi, 2018).

Chapter 3 Deep Dive into YOLOv3

In this Chapter, we overview the YOLOv3 algorithm. We analyze some of its components, explore the network layers and explain their functionalities and roles in the YOLOv3 architecture. Key areas of focus of this overview include a) Darknet-53, which is the backbone of YOLOv3 and is responsible for feature extraction; b) the feature pyramid network (FPN), the main component of the algorithm's neck, which employs additional convolutional layers with the purpose of enabling the network to detect objects at multiple scales; c) the YOLO heads, the final stage of the object detection process. In the last Section of this Chapter, we describe the process of training, testing and validation.

3.1 YOLOv3 architecture overview

The inputs of the YOLOv3 model are images or video streams. The default image resolutions that YOLOv3 typically accepts are 416x416 and 608x608 pixels. These input sizes are the most commonly used because they offer a good balance between detection accuracy and computational efficiency. However, YOLOv3 is a fully convolutional network, which means that it does not use any fully connected layers that require a fixed-size input. Thus, the network may process images of various sizes. It can technically accept images of any resolution, as long as the dimensions are divisible by 32.

The network architecture of YOLOv3 includes a series of convolutional layers, some of which have strides greater than 1. Note that a stride of 1 means that the filter of the convolution process moves one pixel at a time, resulting in a high-resolution output. On the other hand, a stride of 2 means that the filter moves two pixels at a time, resulting in a downsampled output with reduced spatial dimensions.

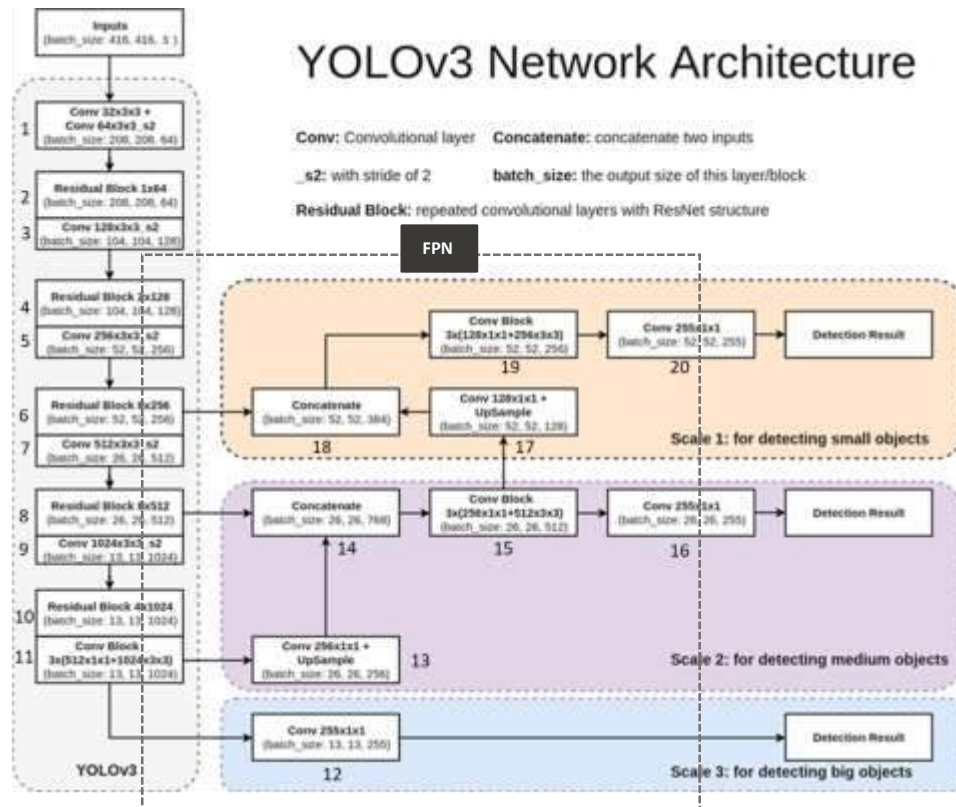


Figure 3.1 YOLOv3 network architecture (Palma, 2020)

The YOLOv3 network has three main parts, each highlighted in different colors and steps (Figure 3.1). First is the backbone which is responsible for feature extraction. The backbone is illustrated by the grey section and its functionality is to process the input image to extract features. It includes steps 1 to 11, where several layers reduce the image size while increasing the detail in the features detected.

The second major part of the architecture is the Feature Pyramid Network (FPN) or else the neck, which is highlighted by orange, purple, and blue in Figure 3.1. FPN is responsible for combining features from different stages of the backbone to help detect objects of various sizes. This involves upsampling and concatenation in steps 13 to 20 (will be analyzed in detail in Section 3.2.2), allowing the network to detect both medium and small objects. Step 12 is also part of FPN, but it is used for processing large objects without the need of going through upsampling and concatenation techniques.

Finally, the YOLO heads are where the actual object detection occurs. YOLO heads are the latter parts of the orange, purple, and blue colored sections of Figure 3.1. They use the combined features from the neck to identify objects at three different scales, small (steps 17-20, orange section), medium (steps 13-16, purple section), and large (steps 12, blue section). In the YOLO heads the network predicts the class probabilities for each detected object. This allows YOLOv3 to classify objects within the detected bounding boxes based on the features extracted and processed by the backbone and the neck of the network.

In the following Sections we focus on each of the key components of YOLOv3's architecture and present its functionalities.

3.2 The YOLOV3 architecture

In this Section we describe the three main components of YOLOv3 architecture, the Darknet-53 backbone, the Feature Pyramid Network (FPN), which comprises the network's neck, and the detection heads.

3.2.1 Darknet-53 backbone and feature extraction

The backbone of YOLOv3 serves as the feature extractor responsible for capturing semantic information from input images. Typically, this part consists of a deep convolutional neural network (CNN) pretrained on large-scale image classification datasets such as ImageNet. There are many choices for backbones in YOLOv3, including Darknet-53 and ResNet-152. Darknet-53 comprises 53 convolutional layers organized into convolution and residual blocks, providing robust feature extraction capabilities (see Fig. 3.2). It is characterized by its simplicity and effectiveness, making it a suitable backbone for YOLOv3 (Teptaris et al., 2023). On the other hand, Resnet-152 consists of 151 convolutional layers and 1 fully connected layer at the end of its network. It is a deep convolutional network which offers high accuracy in more complex patterns. It has increased computational requirements that make it slower to train, but it provides higher accuracy (Xu et al., 2019).

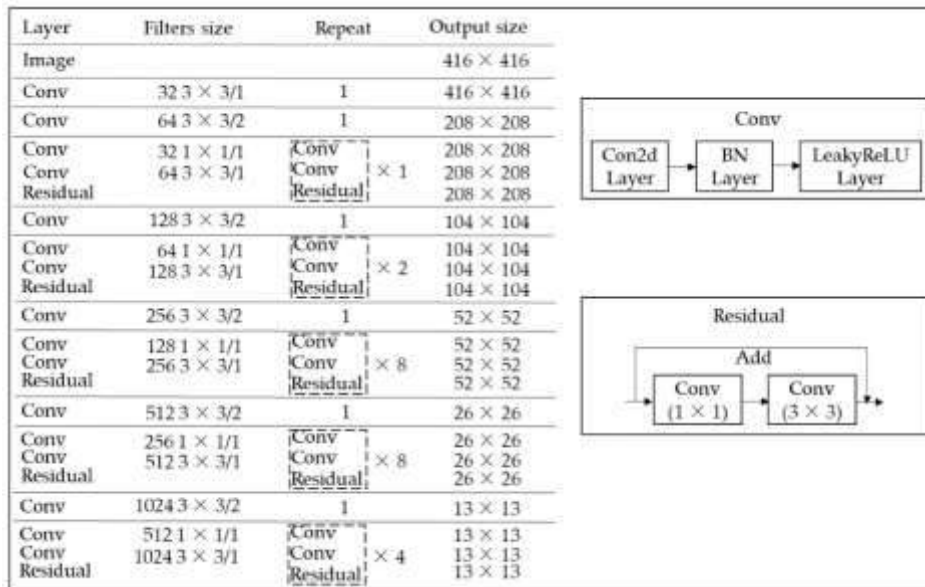


Figure 3.2 Darknet-53 architecture (Ma et al., 2020)

The Darknet-53 architecture (see Fig. 3.2) comprises convolutional blocks and residual blocks. Each convolutional block includes a 2d convolutional layer, a batch normalization layer and a LeakyReLU layer. These layers perform basic feature extraction and dimensionality reduction, enabling the network to capture low-level visual patterns. On the other hand, residual blocks contain a series of convolutional layers followed by a shortcut connection that skips these layers and adds the original input to the output. This shortcut is important because it helps the network learn more effectively and capture more complex patterns, enhancing network's overall performances (He et al., 2016).

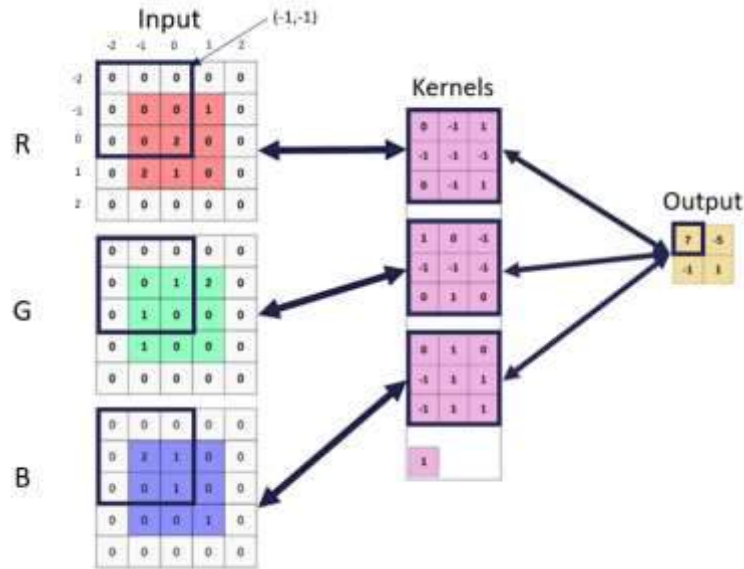


Figure 3.3 A 3x3 kernel (per channel) slides over the input to generate the output (Teptaris et al., 2023)

The convolution block

Convolution operation

The convolution operation in deep learning involves using convolutional filters, also known as kernels to extract features from input data. These filters are small numeric matrices with fixed dimensions. More precisely, a kernel (filter) is a small matrix of weights and is used to perform convolution operations on the input image and on the outputs of previous convolution blocks. The output of each convolution step is obtained by element-wise multiplication between the kernel matrix and a corresponding region of the color channel, concluding by summing the results. This process is repeated for all three-color channels, and the resulting numbers are summed together to form the elements of the output matrix (see also Fig. 3.3). The convolution operation progresses the kernel across the input image (at a specific stride) producing a new matrix called a feature map. Common kernel sizes are 1x1, 3x3, or 5x5. In the YOLOv3 case, the kernel sizes are 3x3 and 1x1. During training, the weights of the kernels are adjusted to improve the network's ability to extract relevant features from the input data.

In the architecture of Fig. 3.2, the first convolution block consists of 32 filters, each with a size of 3x3 and a stride value of 1. This block is repeated once, and the output size remains the same as the input size (e.g. 416 pixels x 416 pixels). This operation is repeated for all 32 kernels, generating a total of 32 feature maps (Teptaris et al., 2023).

Batch normalization

Batch normalization (BN) is a technique used in deep learning to make training more efficient and stable. It works by normalizing the inputs to each layer, making sure that the activations (the outputs of a layer) have a consistent distribution across the mini-batches used during training. This helps to address a problem called "internal covariate shift", which occurs when the distribution of inputs to a layer changes during training, making it harder for the model to adapt on the new changes. This process always runs

between a convolution operation and an activation function and allows successive layers of the network to learn more independently (Ioffe and Szegedy, 2015a).

More specifically, for each mini-batch, batch normalization calculates the mean and variance of the activation values (the outputs of the previous layer). It then normalizes the activations so that they have a mean of 0 and a variance of 1. This ensures that the input distribution remains consistent across layers during training. After normalization, two parameters, γ (*gamma*) and β (*beta*), are introduced. The γ parameter controls the scaling (how stretched or compressed the values are), and the β parameter controls the shifting (moving the values up or down). These parameters allow the model to "un-normalize" the data if needed, so that the network can adapt to a wider range of patterns. These two parameters are adjusted during training to optimize the network's performance. By keeping the input to each layer normalized, batch normalization makes the training process more stable and faster, reducing overfitting as it adapts more accurately to the new input layers. (Teptaris et al., 2023).

LeakyReLU

After each batch normalization process, the Leaky Rectified Linear Unit (LeakyReLU) is activated. This is an improved version of the ReLU activation function. LeakyReLU addresses the issue of negative values turning into zeros. By introducing a small slope for negative values, the LeakyReLU prevents this problem and ensures that the weights of neurons are still affected during training. This activation function is commonly used due to its simplicity and low computational requirements (Dubey and Jain, 2019).

The residual block

The residual block combines the output of the previous layer, denoted as x , with the output of the current layer, denoted as $f(x)$ (see Figure 3.4). Specifically, it adds the feature maps generated by the previous convolution block to the feature maps produced by the convolution layers in the residual block. To ensure that the dimensions of x match those of $f(x)$, a 1×1 convolutional layer is applied to x before it is added to the output of the residual block. This 1×1 convolution adjusts the number of channels and the spatial dimensions of x , making it compatible with $f(x)$. Within the residual block, the convolution layers typically include a 3×3 convolutional layer with stride 2, followed by Batch Normalization and a Leaky ReLU activation. A stride of 2 is used to downsample the feature maps, reducing their spatial dimensions by half. This choice of stride 2 is important for capturing higher-level, more abstract features, increasing the area that each neuron can "see" effectively and improve computational efficiency. The addition of x and $f(x)$ helps deeper networks to learn more efficiently (He, Zhang, Ren, & Sun, 2016).

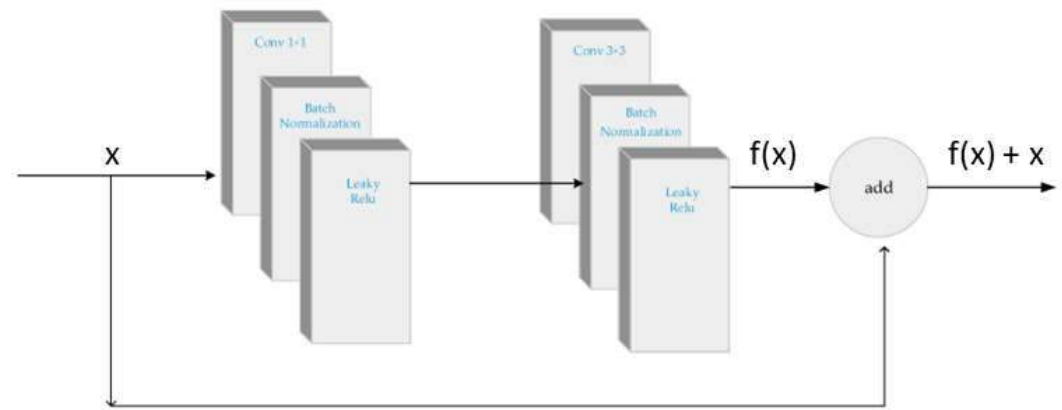


Figure 3.4 YOLOv3 Residual block structure (Xu & Wu, 2020)

3.2.2 Feature Pyramid Network (FPN)

The neck of YOLOv3 acts as an intermediary between the backbone and the YOLO heads. It improves the features extracted by the backbone, enhancing the capability of detecting objects at different scales. This component often applies additional convolutional layers and feature based techniques to integrate multi-scale information effectively.

The Feature Pyramid Network (FPN) is a widely used architecture for the neck in YOLOv3. Its architecture is presented in Fig. 3.5. It creates a multi-scale feature pyramid by combining features from different levels of abstraction within a convolutional neural network (CNN). In YOLOv3's architecture (Figure 3.1), steps 12, 13, 14, 17 and 18 are the ones that comprise the FPN architecture.

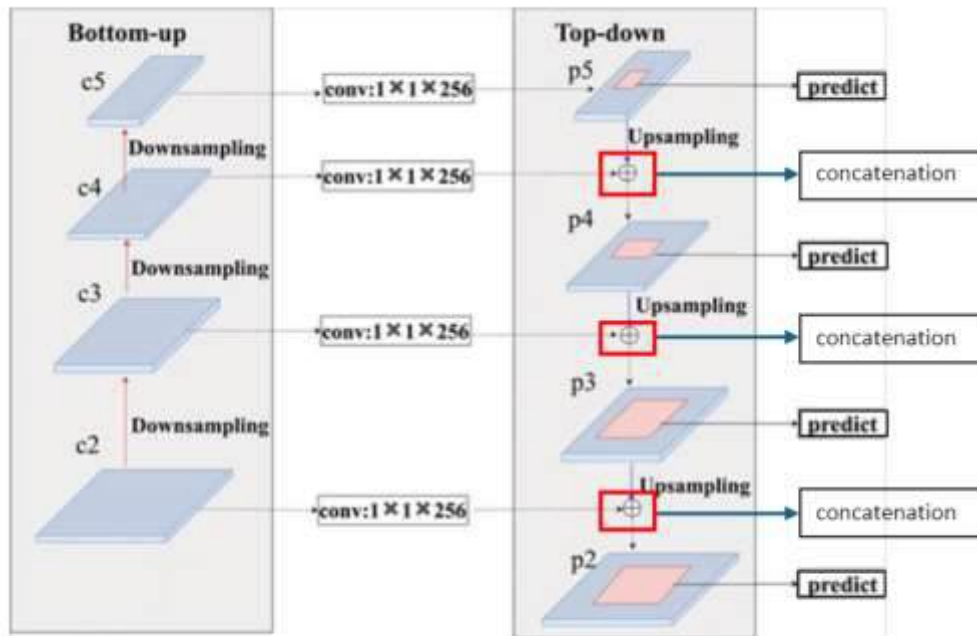


Figure 3.5 Network architecture of feature pyramid network (FPN) (Zhang et al., 2021).

As shown in Fig. 3.5, FPN has two main components: the bottom-up pathway, downsampling and the top-down pathway, upsampling. Downsampling reduces the spatial resolution of feature maps while increasing their depth, enabling the network to capture higher-level features. On the other hand, upsampling increases the spatial resolution of feature maps, allowing the network to reconstruct higher-resolution features from lower-resolution inputs.

Before we describe Fig. 3.5 in detail, please note that:

- The downsampling operation reduces the spatial dimensions (height and width) of the feature maps, allowing the network to capture more complicated and abstract features at higher layers. This is achieved through operations like convolution with stride > 1 . Convolution with a stride greater than 1 reduces the number of positions where the filter is applied across the input feature map. As a result, the output feature map has fewer spatial dimensions (height and width). Pooling layers are another method used to downsample feature maps in CNNs. Downsampling occurs in Step 1, Step 3, Step 5, Step 8, and Step 11 in the context of YOLOv3 (Figure 3.1). These steps involve convolutional layers with a stride of 2 (s_2), which reduces the spatial resolution of the feature maps progressively through the network (Xu et al., 2019).
- The upsampling operation increases the spatial resolution of feature maps, allowing lower-resolution feature maps to be scaled up and aligned with higher-resolution feature maps from earlier layers. The nearest neighbor method is used during upsampling. Specifically, an empty

initial upsample grid is generated. Subsequently, every pixel in the upsample grid is filled with the nearest pixel in the original image patch. This process is repeated until all pixels in the unsampled grid are filled with the image patch values (Figure 3.6) (Tepteris et al., 2023). In the context of YOLOv3, this can be seen in Step 13 (Conv 256x1x1 + UpSample) and Step 17 (Conv 128x1x1 + UpSample) (Figure 3.1).

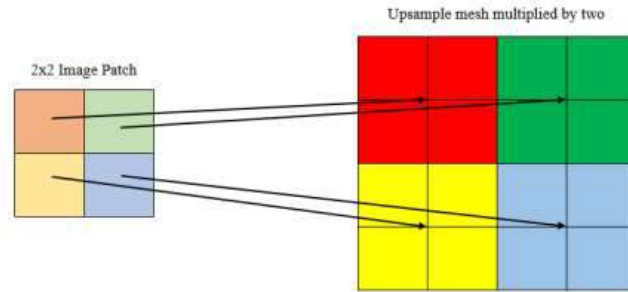


Figure 3.6 Upsampling layer (Tepteris et al., 2023)

- The concatenation operation combines these “upsampled” feature maps with feature maps from earlier layers, merging high-resolution with lower resolution. This operation combines the depth of two feature maps to capture low-level features and detect small objects (Tepteris et al., 2023). For instance, in the YOLOv3 model, the output of step 18 (Figure 3.1) receives the outputs of steps 6 (52x52x256) (Figure 3.1) and 17 (52x52x128) (Figure 3.1), which have the same width and height but different depth dimensions (Figure 3.7). So, the outcome of the concatenation operations is sized at 52x52x384. In the context of YOLOv3, this can be seen in Step 14 (Concatenate with batch size: 26, 26, 768) and Step 18 (Concatenate with batch size: 52, 52, 384) (Figure 3.1).

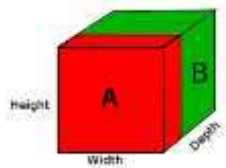


Figure 3.7 Concatenation of two inputs (Tepteris et al., 2023).

Now let’s turn to Fig. 3.5. In the Figure, the bottom-up pathway, feature maps labeled c2, c3, c4, and c5 are extracted from different layers of a convolutional neural network (CNN). As we progress from c2 to c5, the spatial resolution of the feature maps decreases due to downsampling, while the quality of information contained within a feature map increase. In a CNN, earlier layers (like the ones producing c2) typically learn to detect basic features such as edges, textures, or simple shapes. These features are fundamental and low-level. As we move deeper into the network, the layers (producing feature maps such as c3, c4, and c5) start combining these basic features to recognize more complex patterns, objects, or high-level concepts. By this process the network learns to increasingly detect more information about the input image. Specifically, c2 represents a feature map from an earlier layer of the CNN (high resolution,

low semantic information), c3 from a deeper layer (lower resolution, higher semantic information), c4 from an even deeper layer, and c5 from the deepest layer (lowest resolution, highest semantic information).

In the top-down pathway, feature maps labeled p5, p4, p3, and p2, where p is the prediction, are progressively upsampled and combined with corresponding bottom-up feature maps. Specifically, p5 is obtained from c5 using a 1×1 convolution to adjust the channel dimension, p4 is obtained by upsampling p5 and combining it with c4, p3 by upsampling p4 and combining it with c3, and p2 by upsampling p3 and combining it with c2. Each “upsampled” feature map is combined with a corresponding “downsampled” feature map, leveraging both high-level information from deeper layers and more detailed one from earlier layers giving the ability to the network to detect objects at various scales (Alexey, 2024).

By integrating upsampling, downsampling and concatenation operations, YOLOv3 creates a more detailed feature representation that improves the network's ability to detect objects of different sizes (Alexey, 2024).

3.2.3 Detection heads

The primary function of the YOLO heads is to predict bounding boxes for objects detected within the input image. YOLOv3 employs three separate detection heads, each responsible for detecting objects at different scales. These heads are associated with feature maps of different sizes. More specifically:

- The large-scale detection head is designed to capture large objects and operates on the output feature map from the final layer of the network
- The medium-scale detection head captures medium-sized objects
- The small-scale detection head is responsible for detecting small objects

Each detection head is associated with three anchor boxes. These anchor boxes are predefined and help in predicting the bounding box dimensions. Each anchor box has a fixed width and height, and these values are adjusted during training to better fit the objects in the dataset. (Redmon and Farhadi, 2018).

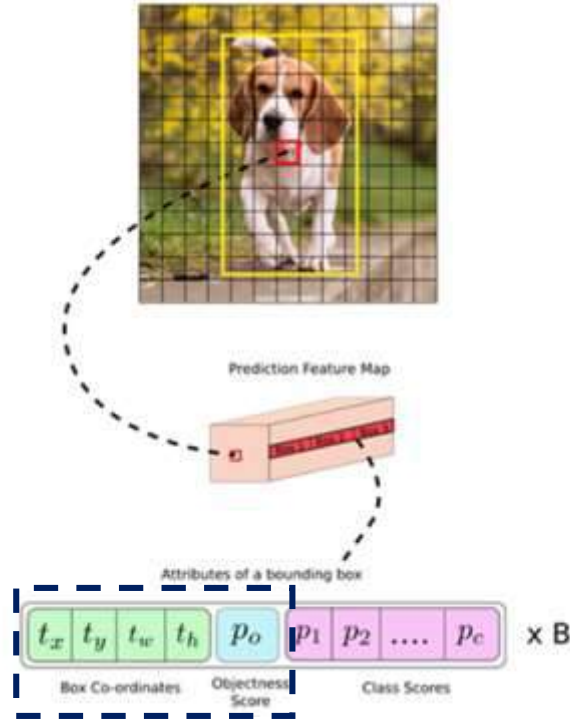


Figure 3.8 YOLOv3 Output vector per anchor in each cell (Teptaris et al., 2023)

Regarding the architectural perspective, each detection head is placed before a convolutional layer with a 1×1 kernel size which is responsible for the attributes of the anchor boxes (see Fig. 3.8). Therefore, the number of filters in this layer is determined by the number of anchor boxes and classes (see Eq. 3.1).

More specifically it is determined by the formula:

$$filters = num_{anchors} \times (4 + 1 + num_{classes}) \quad (3.1)$$

where,

$num_{anchors}$ = the number of anchor boxes,

$num_{classes}$ = the number of classes

Value 4 indicates the four anchor box coordinates which define the location (t_x, t_y) and the size (t_w, t_h) of the bounding box that contains the detected object (Figure 3.8).

Value 1 indicates the objectness score (p_o), that is the probability that an object is present in the bounding box and its value ranges between 0 to 1 (Figure 3.8). See also the black box in Figure 3.8.

For example, if there are 3 anchor boxes and 80 classes the number of filters in the convolutional layer of each detection head would be $3 \times (4 + 1 + 80) = 255$ filters. This equation is applied in each filter of convolution layer before the detection heads.

The output of each detection head has a dimension:

$(grid_{size}, grid_{size}, filters)$

In this case, different grid sizes are used to detect objects at various scales:

- Large grid (52x52): Downsampled by a factor of 8 ($416 / 8 = 52$), detecting small objects
- Medium grid (26x26): Downsampled by a factor of 16 ($416 / 16 = 26$), detecting medium-sized objects
- Small grid (13x13): Downsampled by a factor of 32 ($416 / 32 = 13$), detecting large objects.

In Figure 3.1, that describes the YOLOv3 architecture, the detection head for large objects is illustrated in steps 12 and 13. In step 12, a convolutional block with a 1×1 kernel size is applied directly to the feature map from the backbone network, preparing the feature map for detecting large objects. In step 13, a $Conv\ 255 \times 1 \times 1$ layer is applied with a 1×1 kernel size to reduce the number of channels in the feature map to the size required for the final detection. This step produces the final detection output for large objects which has a dimension of $(batch_{size}, 13, 13, 255)$.

The detection head for medium objects is illustrated in steps 15 and 16. In step 15, a convolutional block is applied to the feature map resulting from the previous upsampling and concatenation operation (step 14). In step 16, a $Conv\ 255 \times 1 \times 1$ layer with a 1×1 kernel size is used to reduce the number of channels to 255, similar to the detection head for large objects. This layer outputs a feature map of dimensions $(batch_{size}, 26, 26, 255)$, designed for detecting medium-sized objects.

The detection head for small objects is illustrated in steps 19 and 20. In step 19, a convolutional block is applied to the feature map from the upsampling and concatenation operation (step 18). In step 20, again a $Conv\ 255 \times 1 \times 1$ layer is applied with a 1×1 kernel size to produce the feature map for detecting small objects. This feature map, designed for detecting small objects, has dimensions of $(batch_{size}, 52, 52, 255)$ (Alexey, 2024).

3.3 Training, validation and testing

In this Section, we overview training, validation and testing processes of YOLOv3 algorithm.

Training is the process by which the YOLOv3 algorithm learns to detect and classify objects in images. During training, the model is fed with a large set of annotated images, where each image is paired with its corresponding ground truth labels (the coordinates of bounding boxes and the class of the objects). The algorithm adjusts its parameters repeatedly to minimize the difference between its anchor boxes and the ground truth bounding boxes. Also, each anchor box is assigned to the ground truth bounding box that has the highest IoU with it. The corresponding class label of the ground truth box is then used to train the network to predict the correct class. Therefore, training is used to enable the models to learn and understand the patterns in the data (Redmon and Farhadi, 2018).

Validation is the process used to evaluate the performance of the YOLOv3 model during training. For this process, a separate subset of the dataset, called the validation set, is used. After each round of training (an epoch), the model's performance is tested on this validation set, to detect any overfitting or

underfitting that might occur. Overfitting happens when a model learns too much detail from the training data, including irrelevant patterns. As a result, the model becomes very good at predicting the training data but fails to perform well on new data because it has "memorized" the training data instead of learning general patterns. Underfitting occurs when a model does not learn enough from the training data. It fails to capture more complex patterns in the data and therefore performs poorly both on the training and the new data. Thus, the validation process is used to tune the models hyperparameters and prevent overfitting, ensuring that the model generalizes well to new data (Redmon and Farhadi, 2018).

Testing is the process of evaluating the final performance of the trained YOLOv3 model on a separate set of images that were not used during training or validation. This dataset is called the test set. The model's evaluation metrics are calculated on the test set to measure how effectively it can detect and classify objects. Therefore, testing is used to evaluate the final model's performance and confirm its accuracy (Teptaris et al., 2023).

For all three processes, training, validation and testing, an annotated dataset of images is used. This dataset consists of images that are labeled with the ground truth boxes and class of objects. Note that the training set used for model learning comprises typically 80% of the dataset, the validation set 10%, and the testing set comprises also 10% of the dataset.

3.3.1 Training process of YOLOv3

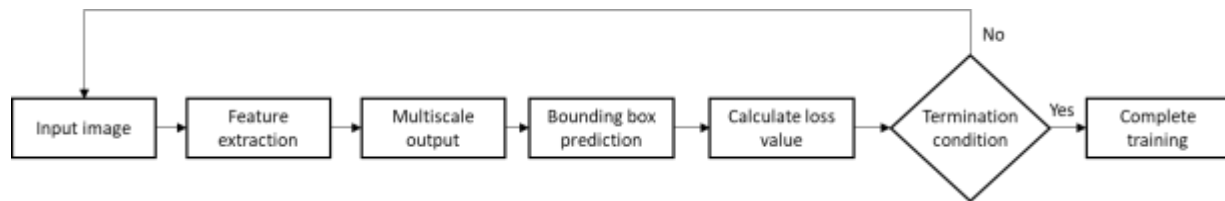


Figure 3.9 Training process of YOLOv3 (Teptaris et al., 2023)

Figure 3.9 overviews the step-by-step actions that take place in the training process. Initially, the training dataset's images are divided into smaller batches. This division helps speed up the training process by allowing the neural network's weights to be updated more frequently. Each batch contains images, such as color images with dimensions 416x416 pixels, represented as arrays of size (416, 416, 3). The batch size is set prior to training. For instance, if the batch size is 32, it means that 32 images are fed to the model in each iteration to update the weights of YOLOv3 model. On the other hand, when the entire dataset of images is passed through the YOLOv3 network once, this is one epoch. For example, consider a 4000 images dataset, divided in batches of 32 images. It will take 125 iterations to complete one epoch. More specifically, an iteration is when the model processes a small batch of images, and in this case, the batch size is 32. Since the dataset has 4000 images, it takes 125 (4000/32) iterations to go through all images once, which is one epoch.

In the second step of Fig. 3.9 the image batches are passed through a classification neural network, which extracts features from the images, such as object outlines. In the third step, the model predicts classes for

objects at three different scales, enabling the detection of small, medium, and large objects. Step four involves detecting objects within each of the three scales and enclosing them within bounding boxes. In step five, a loss value is calculated using the loss function. This value is a combination of how accurate the class predictions are and how well the bounding boxes fit the predicted objects.

In YOLOv3, the loss value is computed for batches of images rather than individual images. This means that the loss function is applied to a group of images, and the resulting outputs are combined to calculate a single loss value for that batch. The optimizer then uses this loss value to adjust the model's weights. This process is repeated for multiple batches until the model converges or a predetermined number of iterations is reached (Teptoris et al., 2023).

Key technical tools used in training

Intersection over Union (IoU)

Intersection over Union (IoU) is the region where the ground truth bounding box and predicted bounding box intersect over the region where they are united (Figure 3.10). According to the definition of IoU, an IoU can have a value of 0 or 1, and the objective of training is to choose the predicted box that most closely resembles the ground truth box in order to obtain an IoU as close to 1 as possible. An IoU threshold value is used during training to keep only the "good" predicted box and throw out the ones below the threshold. A typical threshold value is 0.5 (Kamal, 2021).

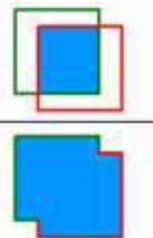
$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$


Figure 3.10 Computing Intersection over Union (IoU) (Padilla, Netto, & Silva, 2020)

The ground truth box and the predicted box are represented by the red and yellow boxes in Figure 3.11, respectively. Because of the tiny area of intersection in the left image, the IoU is low. In contrast, the right image's area of intersection between the two boxes is nearly equal to their union, meaning the IoU is near to 1 (Kamal, 2021).



Figure 3.11 IoU of bounding boxes (Kamal, 2019)

Objectness and Class Scores

The objectness score in YOLOv3 explains us how confident the model is that a bounding box contains an object, regardless of its class (e.g., car, person, bike) and how well that bounding box aligns with the actual object.

This score is a combination of two factors:

1. Probability of Object Presence ($P_c(object)$): This is the probability that the predicted bounding box contains any object, as predicted by the model.
2. Intersection Over Union (IoU): This measures how much the predicted bounding box overlaps with the actual object's ground truth bounding box.

Objectness score is calculated by multiplying these 2 values (Huang et al., 2022):

$$P_0 = P_c(object) * IoU \quad (3.2)$$

If $P_c(object)$ is high, close to 1, it means the model is highly confident that an object exists. If IoU has a value close to 1, it means that there is a significant overlap between ground truth box and the predicted box, suggesting a more accurate prediction. Therefore, for an accurate prediction, P_0 should ideally be greater than 0.5 and as close to 1 as possible. A high P_0 indicates both confidence in object presence and accurate localization of the object's bounding box, which is important for accurate object detection (Teptaris et al., 2023).

The class score indicates the probability that the detected object belongs to a certain class (e.g., car, person, bike), with the highest score indicating the predicted class. More specifically, for each anchor box, the model computes a probability over all predefined classes included in the dataset. For instance, if there are 80 classes, the model outputs 80 probabilities corresponding to these classes (Redmon et al., 2016).

For example, if the class scores are:

- **0.7** for "car"
- **0.2** for "person"
- **0.1** for "bike"
- **0** for all other classes.

Then, the model predicts that the object is a car (since 0.7 is the highest score).

As illustrated in Figure 3.12, the output of the YOLOv3 model consists of bounding boxes that enclose each detected object, along with a label displaying the objectness score and the name of the class to which the object belongs.



Figure 3.12 YOLOv3 detection with class score (Shivaprasad, 2019).

Non-Maximum Suppression (NMS)

In Figure 3.13, the class probabilities are shown in the left corner of each bounding box next to the class name of the detected object. As it can be seen in this Figure two bounding boxes are drawn around the same object (dog and person). To ensure accurate object detection, the bounding box that best encloses the object, with the highest confidence score (1.00 for the class = “dog”) should be retained, while the other inaccurate and redundant bounding boxes (0.31 for the class = “person”) should be ignored. This process is known as Non-Maximum Suppression (NMS) and its purpose is to reduce duplicate detections and keep only the most accurate prediction for each object. It basically works by comparing the confidence scores of all bounding boxes and keeping the ones with the highest scores while removing those with a high overlap with the chosen bounding box (Hosang et al., 2017).



Figure 3.13 YOLOv3 prediction example (Gilbert, 2020).

The NMS algorithm's threshold is a hyperparameter that can be adjusted to balance accuracy and recall (Teptaris et al., 2023).

Loss Function

The loss function is utilized to optimize the model's parameters throughout the training procedure. Training continues until the loss function reaches its minimum value, signifying that the model has successfully learned to detect objects in an image. YOLOv3's loss function is computed at the last output layer of YOLOv3.

The loss function in YOLOv3 consists of three parts (Huang, Lin, & Liu, 2022):

1. $Error_{coord}$: refers to the coordinate prediction error
2. $Error_{objectness}$: refers to an Intersection Over Union (IoU) error
3. $Error_{class}$: refers to the classification error

$$Loss = Error_{coord} + Error_{objectness} + Error_{class} \quad (3.3)$$

For further information on YOLOv3 loss function see (Tepteris et al., 2023).

3.3.2 Validation process of YOLOv3

The validation process evaluates the performance of the YOLOv3 model during training using a separate subset of the dataset known as the validation set. After each round of training (an epoch or in our study several iterations), the model's performance is tested on this validation set to detect any overfitting or underfitting occurs during the training process. Overfitting occurs when the model learns too much detail from the training data, including irrelevant patterns, resulting in high accuracy on the training data but poor performance on new data. Underfitting happens when the model does not learn enough from the training data, failing to capture complex patterns and thus performing poorly on both training and new data. During this process, the network uses a separate dataset, called the validation set, which is different from the training dataset. The validation process is crucial for tuning hyperparameters and preventing overfitting, ensuring that the model generalizes well to new data (Redmon and Farhadi, 2018).

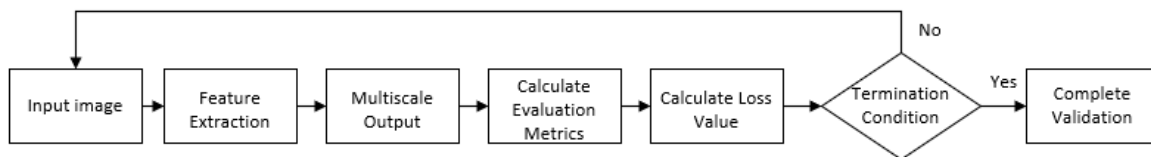


Figure 3.14 Validation process of YOLOv3

As illustrated in Fig., 3.14, in step 2 of the validation process the model continues with the feature extraction from the input image without updating the weights, as it does during training. In step 3, the model generates predicted outputs, including class labels and bounding boxes. In step 4, the model's performance is evaluated by comparing the predicted outputs with the actual labels using various evaluation metrics, such as precision, recall, Average Precision (AP) and mean Average Precision (mAP). In step 5, the validation loss is calculated to assess how well the model is performing on the new input image data. Then, in step 6, a decision is made, if the model's performance is acceptable, meaning the validation

loss is low and the evaluation metrics are satisfying, the process moves to Step 7, completing the validation. When the model's performance is not acceptable during validation, adjustments are made to improve it, typically by tuning hyperparameters. This process can happen by stopping training and starting over with new hyperparameters. After evaluating the model during validation, hyperparameters like learning rate, batch size, can be adjusted and the training process starts over again.

Satisfactory validation result

A good validation result occurs when the validation loss is low and close to the training loss, indicating that the model generalizes well to new data. Additionally, high values for evaluation metrics, such as precision, recall, Average Precision (AP), or mean Average Precision (mAP) reflect that the model is accurately making predictions across the predetermined classes and identifies true positives. Additionally, when the training and validation losses are similar and stable after multiple iterations (epochs), it means that the model is well-trained with no overfitting or underfitting issues.

Unsatisfactory validation result

On the other hand, an inferior validation result is characterized by a high validation loss compared to training loss. This indicates overfitting issues when the model performs well on training data and underfitting issues when the model fails to learn from new data. Low precision or recall values are also a sign that the model is missing information or is making incorrect predictions.

Nevertheless, if the validation results are not the desirable ones, there are several strategies we can follow to improve the model's performance. One of them is the hyperparameter tuning, where parameters, such as learning rate, batch size, or the number of epochs is adjusted. If the model is overfitting, techniques, such as reducing model's complexity (fewer layers) can help. In cases of underfitting, increasing model complexity or training for more epochs may be necessary. Data augmentation can be also used to expand the training data and prevent overfitting by generating new examples from existing data (e.g., rotating or flipping images). A technique which was used in our research too.

All in all, the validation process is important for making sure a machine learning model is reliable, accurate, and works well with new data. It helps us check how well the model performs and shows where adjustments are needed to avoid issues like overfitting or underfitting. By analyzing the performance metrics and making small changes to improve the model's performance, we can get the best possible results (Redmon et al., 2016).

3.3.3 Testing process of YOLOv3

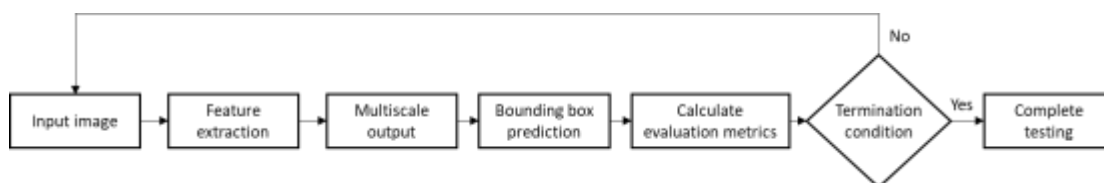


Figure 3.15 Testing process of YOLOv3 (Teptaris et al., 2023)

The testing process is essential because it evaluates how well the trained model adapts to new data. The performance metrics obtained during testing indicate how accurately the model detects objects and handles different conditions on which it was not trained on. Figure 3.15 overviews of the testing process. During this process each annotated image from the testing subset is fed into the trained network for object detection. The first four steps of the testing process are similar to those in the training process except that the weights are not updated during testing. The model uses the weights that were learned during training to make predictions on the test data, but it does not modify them. Testing is a separate process where the model's performance is evaluated on the data of the testing subset, with no further learning or weight updates. In step five, after obtaining the detection results, evaluation metrics (such as Precision, Recall, F1-Score, Mean Average Precision (mAP), etc.) are calculated to analyze the performance of the model on the testing subset (see below).

Step six involves checking if the termination conditions have been met. These conditions could be related to processing a certain number of images or completing all images in the test set. If the conditions are met, the testing process is completed. Otherwise, the next image is processed, and the steps are repeated (Teptaris et al., 2023).

Successful testing process

Successful testing means that the model has reached the expected thresholds for evaluation metrics, such as precision, recall, F1 score, average precision (AP) or mean average precision (mAP). For example, if the goal was to achieve 70% precision and recall on a specific dataset, the test is successful if these targets are met. It also shows us that the model performs with consistency across the different testing subsets; i.e. data subsets that might have different lighting conditions, object sizes and orientations than the training dataset. Testing can also indicate a low number of false positives (detecting an object that isn't there) and false negatives (missing an object that is present), which means the model reliably detects objects without significant errors.

Unsuccessful testing process

A non-successful test occurs when the model fails to meet the performance goals. This means that model's performance metrics fall below the acceptable thresholds. For instance, if the target precision is 70% and the model only achieves 50%, this would be considered unsuccessful. Also, when testing fails the model generates too many false positives or false negatives, concluding to an unreliable object detection process. If the model performs well on the training dataset but its performance is low on the testing dataset, it indicates overfitting. This means that the model is memorizing training data rather than learning general patterns. Finally, if the model performs poorly on both training and testing datasets, it means that it was not trained well, indicating underfitting. If a test is unsuccessful, the next steps typically involve making adjustments and analyzing the errors. For example, if the dataset faces specific difficulties, such as small objects or crowded scenes, we analyze what changes can be made to the algorithm to help the model improve detection under these conditions. If the weaknesses are identified, e.g. for the detection of small objects, we could proceed with modifying the model's architecture to help improve its performance. After making appropriate changes, the model should be trained, validated, and tested again to check if the issues are solved. This cycle continues until the desired performance is achieved (Zhang and Wallace, 2015).

Evaluation metrics

As discussed above, evaluation metrics are utilized to evaluate the performance of a model and they are calculated during the algorithm's validation and testing process (Fig. 3.14 and Fig. 3.15). Key evaluation metrics include recall, precision, F1-score, Average Precision (AP), and mean Average Precision (mAP). These metrics are derived from the model's classification and detection outcomes, which are identified as False Positives (FP), False Negatives (FN), True Positives (TP) and True Negatives (TN)

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

True positives (TP) can occur, when the both conditions below are met :

- 1) $IoU \geq IoU_{threshold}$, indicating that the predicted bounding box overlaps or matches the ground truth bounding box.
- 2) The class of the object within the ground truth bounding box is predicted correctly.

False positives (FP) can occur in four distinct cases when:

- 1) $IoU < IoU_{threshold}$, indicating that the predicted bounding box locates the object incorrectly
- 2) $IoU \geq IoU_{threshold}$, but the object is assigned to the wrong class
- 3) A predicted bounding box has appeared without a corresponding ground truth bounding box with the respective class.
- 4) The model generates multiple bounding boxes with with $IoU \geq IoU_{threshold}$. In this case, only the bounding box with the highest IoU will be considered a true positive. The remaining bounding boxes are classified as false positive.

False negatives (FN) can occur in three distinct cases:

- 1) $IoU \leq IoU_{threshold}$. FP and FN outcomes occur when the object is mislocated due to insufficient overlap between the predicted and ground truth boxes, leading to an unsuccessful detection. In FP case, the predicted bounding box locates the object incorrectly, thus in FN case the ground truth box fails to detect the object
- 2) $IoU \geq IoU_{threshold}$, but the model is not able to predict the class of the object correctly within the ground truth bounding box. This can result in both FP and FN outcomes. Nevertheless, the FP occurs because the predicted bounding box misclassifies the object, while the FN represents the model's failure to correctly identify the object in the ground truth bounding box
- 3) The model fails to detect an object although it is present in an image, because although a ground truth box exists, the model does not predict a bounding box for the corresponding object.

Finally, *True Negatives (TN)* are the scenarios when the model successfully identifies the non-existence of an object in the image (Xiong et al., 2024).

Precision

The precision metric is used to assess the accuracy of a model in identifying objects. It is calculated by (Vakili et al., 2020):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (3.4)$$

An increased precision can be observed, either when the model generates a larger number of accurate positive classifications, thereby maximizing the number of true positives, or when the model minimizes the occurrence of incorrect positive classifications, thereby reducing the number of false positives (Gad, 2020)

Recall

The recall metric measures the ability of a model to accurately locate objects within an image. It is calculated by dividing the number of correctly identified objects (true positives) by the total number of objects (true positives and false negatives) (Vakili et al., 2020).

Therefore, recall can be calculated using the formula:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3.5)$$

A high recall value suggests that the model is able to find most of the objects in the image, reducing the risk of missing any. On the other hand, a lower recall value indicates that the model is missing a significant number of objects, which can result in inaccurate object detection and lower performance (iguazio, 2022).

F1-score

Precision and recall evaluate different aspects of a model's performance. The F1-score is proportional to the harmonic mean of precision and recall. It is calculated by the formula:

$$\text{F1 - score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.6)$$

The value of F1-score ranges from 0 to 1 (0%-100%) and represents the balance between precision and recall, reflecting the model's overall performance (Vakili et al., 2020).

Average Precision (AP)

Average Precision (AP) is a metric that calculates the mean precision across the range of recall values generated by the model when applied to multiple images. In other words, AP measures the overall precision performance of the model across different levels of recall. It provides a single value that summarizes the model's ability to accurately identify true positives while minimizing the number of false positives across the range of recall values observed in the dataset.

Average Precision (AP) is calculated as the mean precision value across the entire range of recall values generated by the model. (Anwar, 2022).

Mathematically, it is evaluated by the formula:

$$AP_i = \int_{r=0}^1 p(r) dr \quad (3.7)$$

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.

Mean Average Precision (mAP)

The mean Average Precision (mAP) metric is a measure that summarizes the Average Precision (AP) of each individual class and calculates the average across all classes. Mathematically, mAP is represented as (Henderson and Ferrari, 2017):

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (3.8)$$

where,

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

3.4: Operation of YOLOv3

Post successful training, validation and testing, the model may be used for object detection, using the weights computed during the training phase. During operation, a raw input image (non-annotated) is fed to the trained model for object detection.

The model outputs:

- The spatial information of a bounding box that encapsulated each detected object in the image represented as (b_x, b_y, b_h, b_w) , \square where b_x , and b_y are the center coordinates of the bounding box, and b_h and b_w are its height and width, respectively.
- The objectness score, which represents the model's confidence that a bounding box contains an object. This score ranges from 0 to 1, where a higher value indicates a greater possibility that the bounding box contains an actual object
- The class confidence probabilities $p(c)$ for each object in the image, where $c = 1, 2, 3, \dots, c$ and represents the object classes. These probabilities indicate how likely it is that the detected object belongs to each class (Teptaris et al., 2023).

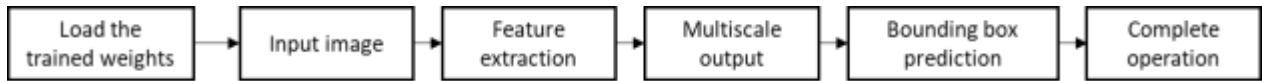


Figure 3.16 Operation process of YOLOv3 (Teptaris et al., 2023)

Figure 3.16 overviews model operation. In step one, the trained weights are loaded into the YOLOv3 network. These weights are the result of training the network and are responsible for detecting objects in real-time. In step two a raw input image, which can be a live video stream or a single image, is fed into the YOLOv3 network. Following, in step three, the input image is processed by the backbone of the YOLOv3 network to extract relevant features, which are essential for identifying objects in the image. In step four, during multiscale detection, the model identifies objects of different sizes by predicting bounding boxes at three different scales within the feature pyramid. Following in step five, the model refines these predictions, placing bounding boxes around the detected objects. Finally, in step six, the operation process is completed, providing the final outputs (bounding boxes, objectness score and class probabilities).

Figure 3.17 illustrates the output after the YOLOv3 algorithm has run. The output image showcases the detected objects with their corresponding bounding boxes, allowing for visual identification and localization of the objects within the image.

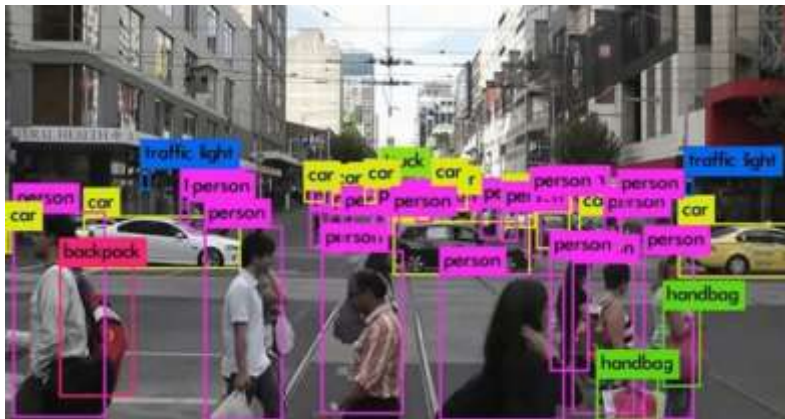


Figure 3.17 Image after the applying of YOLOv3 object detection algorithm (Cruz Martinez, 2021)

Chapter 4 Data preparation and parameter selection for training the YOLOv3 algorithm

In this chapter, we will focus on the process followed for preparing the dataset and selecting the right hyperparameters for training the YOLOv3 algorithm.

First, we focus on the training data. We describe how we collected the images from existing UAV datasets, which parameters we considered for the dataset selection and which datasets are finally chosen. The steps required to adjust the annotations to suit the characteristics of our UAV datasets, including adjustments such as aligning label names and numbering to ensure consistency across datasets, are presented. Subsequently, we describe the experimental setup, covering both the hardware and software used.

Secondly, we present the pre-determined hyperparameters of YOLOv3, as well as the hyperparameters that are associated with YOLOv3's architecture and functionality. After analyzing these hyperparameters, we describe the ones selected for our research and what modifications we made to the initial algorithm to include the new values of these hyperparameters.

4.1 Data collection and annotation

In this Section, we describe the process followed to identify the appropriate data for our research. We discuss the characteristics of each chosen dataset and analyze the modifications made to the final consolidated dataset to meet the requirements of our study. Additionally, we describe the training, validation, and testing subsets, as well as the presence of objects within each predefined class for our research.

4.1.1 Data collection

The success of training object detection and recognition models relies on the quantity and quality of data utilized in the process. Utilizing extensive and diverse datasets is essential to reduce errors, overfitting, and limit bias. Various resources, including cloud repositories, web platforms, resources from universities and research institutions provide annotated image collections for computer vision tasks. While some datasets are freely available, others may require payment or subscription for access.

For this thesis, the predetermined classes for training the object detection model are four. More specifically, the classes are:

- Person
- Car
- Long vehicle that refers to vehicles such as buses, trucks
- Bike

In this thesis, the selection of datasets that already contain the predetermined classes was a prerequisite as the image annotation is a time-consuming process that needs specific tools. The image datasets were sourced from open-source datasets containing images captured by drones. In the selection of the datasets, two parameters were kept in mind, the quantity and the quality of captured images. It is essential to have

a large dataset, capturing images from various perspectives. The qualitative characteristics are also important. The dataset should consist of multicolored images, with numerous objects, with variations in lighting conditions. Furthermore, images should be free from distortions such as blurs, ensuring clear object detection.

There are many publicly available open labelled datasets, including ImageNet (Yang et al., 2022), Common Objects in Context (COCO) (Lin et al., 2014) etc. Each one of them is a set of digital pictures that developers use to train and validate the performance of their algorithms.

Considering the above requirements and characteristics, the datasets selected for YOLOv3 algorithm training, in our research, are analyzed in detail below:

- The **"UA Vehicle Detection Dataset"** is downloaded from GitHub and is a dataset specifically selected for UAV (Unmanned Aerial Vehicle) vehicle detection tasks. It contains a collection of images captured by UAVs, focusing on scenes where vehicles are present. The dataset is annotated to include bounding boxes around vehicles, enabling the training and evaluation of vehicle detection algorithms. Additionally, it includes various environmental conditions, lighting scenarios, and vehicle types to ensure robustness of the trained models. Specifically, it includes 1,470 images that contain vehicles such as cars, long vehicles, and bikes, but any of the dataset images contain the object person (which is one of the classes of our study). The size of each image in this dataset is 224×224 pixels (Wang, 2024).
- The **"Stanford Drone Dataset"** is an extensive collection of aerial videos captured by a UAV platform. Hosted on the Stanford Computer Vision and Geometry Lab (CVGL) website, this dataset offers a diverse range of scenes and scenarios, including urban environments, campus settings, and outdoor landscapes. Furthermore, the dataset provides high-resolution images captured from different viewpoints, enabling the exploration of scale and perspective variations. The Stanford drone dataset contains images categorized into six different classes: pedestrians, bikers, skaters, carts, cars, and buses. It includes 6,748 annotated images, displaying diverse image resolutions ranging from 1322×1079 to 1640×1948 pixels. Within these UAV-captured images, all four classes, people, cars, long vehicles, and bikes, are represented. The numerous other labeled objects are excluded intentionally for the purposes of our study (Robicquet et al., 2016).
- **"VisDrone2019DET"** dataset contains a variety of high-resolution images captured by unmanned aerial vehicles (UAVs). It consists of images from various urban and natural environments, including the objects of interest that serve our study. Due to the great number of images that contain, across different classes and environmental conditions, this dataset provides a solid basis for training, validation, and testing purposes. It contains a total of 10,209 annotated images, which are obtained from urban and countryside landscapes, showcasing diverse resolutions ranging from 480×360 to 2,000×1,500 pixels. Additionally, the images within the training and validation sets contain ten distinct classes: pedestrian, person, bicycle, car, van, truck, tricycle, bus, and motorcycle. For the purpose of this thesis only the four predetermined classes are used and, therefore, the rest of them are excluded. More specifically, the class person which integrates both pedestrian and people, the class bike which contains the bicycle and tricycle categories, the class

long-vehicle which contains the categories van, truck and bus and the class car which remains the same (Zhu et al., 2021).

4.1.2 Annotation and consolidation

Before combining the three datasets, we standardized their annotations to be compatible with the YOLO model. This involves aligning label names and numbering to establish similarity across datasets. Specifically, the objective is to categorize annotations into four distinct classes: "person", "car", "long vehicle", and "bike", corresponding to classes zero to three in YOLO format. More information on image annotation in deep learning may be found in (Khan, 2023).

Table 4.1 The number of training, validation and testing objects in the consolidated dataset

Number of Objects				
	UA Vehicle Detection Dataset	Stanford Dataset	VisDrone2019DET	TOTAL NUMBER OF OBJECTS IN EACH CLASS
Number of total objects	26,218	203,722	372,300	
Number of persons	0	92,880	120,365	,
Number of Cars	20,647	26,332	158,914	,
Number of Long Vehicles	5,330	910	46,721	,
Number of Bikes	241	83,600	46,300	,
Total number of objects in all the selected datasets				

Once datasets are standardized, the datasets are merged to a single set that is split into training, validation and testing subsets. The training one comprises 80% of the initial dataset, the validation one 10%, used to evaluate model performance during training, and the testing one also 10%. Each set includes annotations for the specified classes.

The numbers of objects and images in each dataset and their distribution into training, validation and testing are presented in Table 4.1 and illustrated in Figure 4.1, respectively.

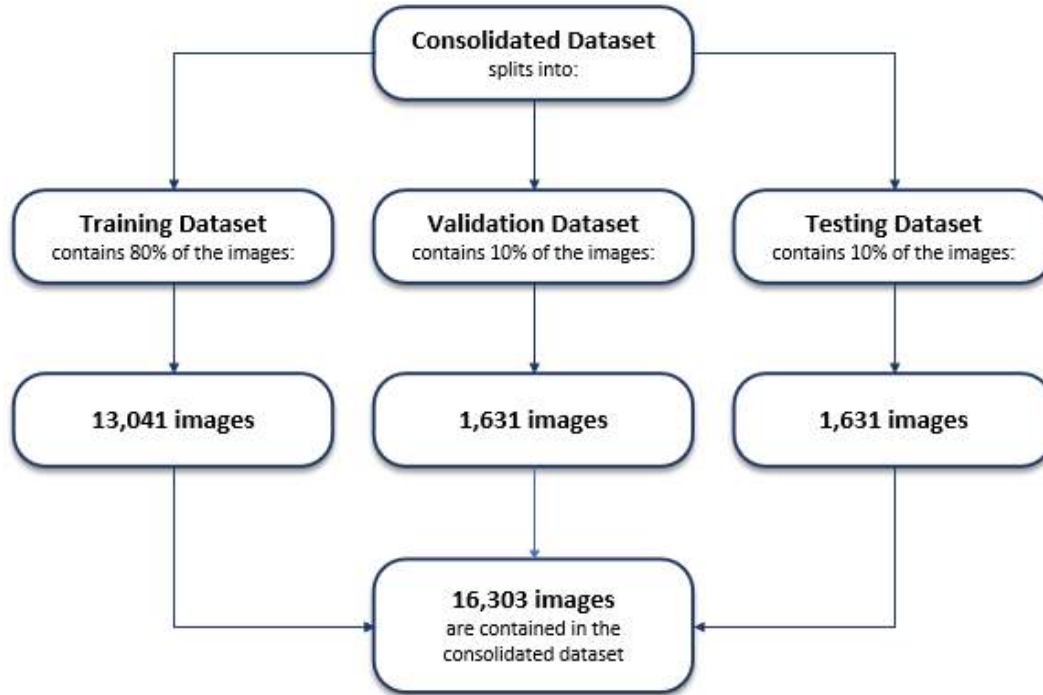


Figure 4.1 The number of training, validation and testing images included in the consolidated dataset.

4.2 Experimental set up

Training and evaluating the YOLOv3 model require significant computational resources and a high-performance system. The hardware configuration (Figure 4.2) and software environment (Figure 4.3) are discussed below.

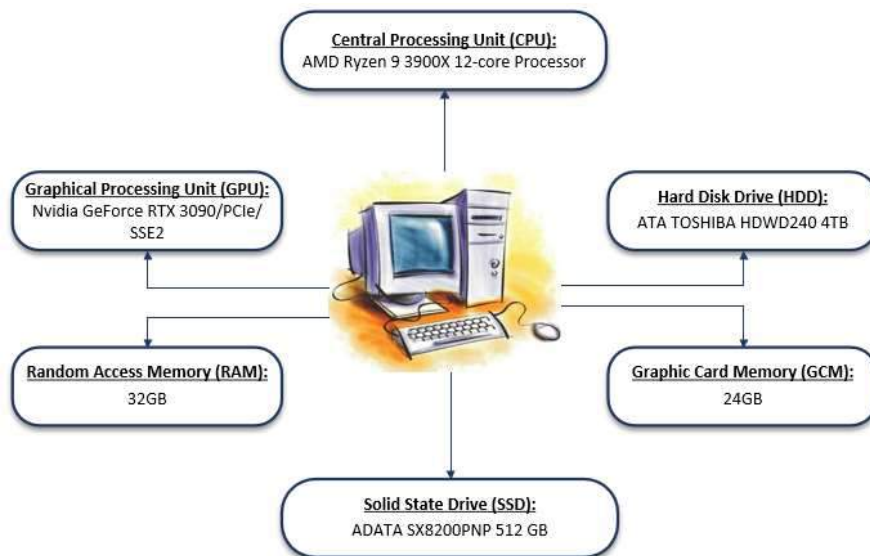


Figure 4.2 Hardware Configuration of the system

The CPU that is used for our study has 12 cores. The GPU is RTX 3090 and is able to handle deep learning and object detection tasks. In terms of Graphics Card Memory (GCM), we used 24GB that may handle images and complex textures (Mu et al., 2011). The RAM of the computing system is 32GB, and the system is equipped with 512 GB SSD and 4TB HDD.

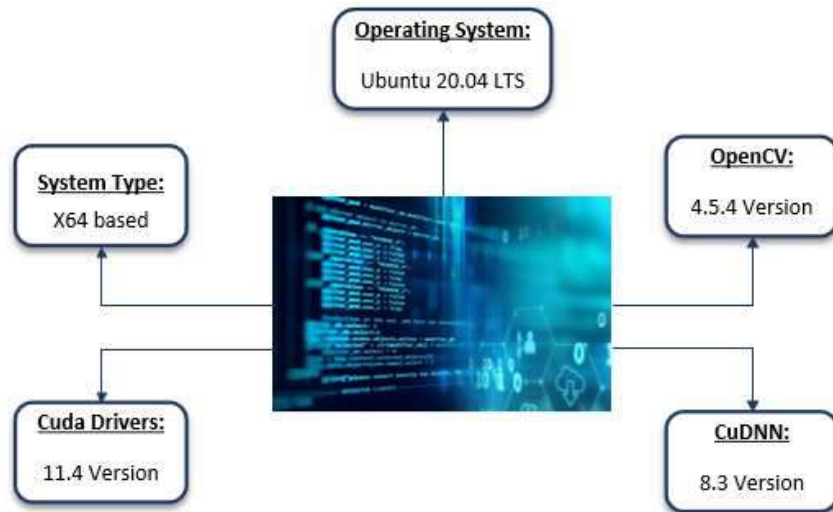


Figure 4.3 Software components of the system

The OpenCV toolkit is an open-source computer vision library known for its extensive range of features, appropriate for image and video processing. This toolkit plays a crucial role in object detection applications and is also integrated into the YOLOv3 model (Culjak et al., 2012).

The CUDA (Compute Unified Device Architecture) and cuDNN drivers were installed in order to optimize the utilization of the RTX 3090 graphics card. CUDA is a platform with computing capabilities and API developed by NVIDIA. This platform allows developers to utilize GPUs for general-purpose computing tasks, accelerating computation-heavy processes, such as deep learning, leading to faster execution and performance (Kirk, 2007). CuDNN (CUDA Deep Neural Network) is a GPU library developed by NVIDIA as well that offers highly optimized implementations of deep learning operations, such as convolutions, pooling, normalization, and activation functions. It is designed to work along with CUDA, enabling faster training by leveraging the processing power of NVIDIA GPUs (Chetlur et al., 2014).

4.3 YOLOv3 hyperparameters

Hyperparameters play a critical role in training any neural network model, since they can significantly affect the performance and accuracy of the model. YOLOv3 provides its users the capability of adjusting various model parameters to improve training and, ultimately, model performance. Generally, the choice of hyperparameters should be based on the available computational resources, the requirements of the specific study/task, and the performance of the model on the validation data.

4.3.1 Hyperparameters determined by the characteristics of the dataset

In YOLOv3, pre-trained convolutional weights are used from the model initially trained on the ImageNet dataset. This method, known as transfer learning, accelerates the training process by leveraging general features like edges and textures learned during ImageNet training. During fine-tuning on a new dataset, the Darknet backbone's pre-trained weights are adjusted, while the YOLO detection layers are fine-tuned to adapt to the specific task. The detection layers are not trained from scratch, but they are updated based on the pre-trained weights.

During fine-tuning on a new dataset, the Darknet backbone's pre-trained weights are typically adjusted or optionally frozen, while the YOLO detection layers are fine-tuned to adapt to the specific task. The detection layers are not trained from scratch but are updated based on the pre-trained weights.

To adjust the model to the new data, certain configuration options are set in the YOLOv3 ".cfg" file. The ".cfg" file is used to define the the architecture and hyperparameters of the neural network. The adjustments in the "cfg" file for our study, are shown in the following table and analyzed below (Redmon and Farhadi, 2018).

T
a

Hyperparameter	Default Value		Updated Value	
	Darknet-53	Resnet-152	Darknet-53	Resnet-152
Classes	80	1	4	4
Max Batches	500,200	10,000	8,000	8,000
Filters	255	24	27	36
Steps	400000, 450000	4000, 6000, 8000, 9000	6400, 7200	3200, 4800, 6400,7200

In this research, specific changes were made to four of the hyperparameters:

- The maximum number of classes
- The maximum number of batches
- The filters that are before the convolutional level of each detection head
- The number of steps

Classes

The number of classes has been reduced from 80 to 4 for the YOLOv3 with Darknet-53 backbone network (Table 4.2) and increased from 1 to 4 for the YOLOv3 with Resnet-152 backbone network (Table 4.2 – see also class adjustment in Figure 4.4. The class numbering is as follows:

- Person indicated as class 0
- Car indicated as class 1
- Long vehicle indicated as class 3
- Bike indicated as class 4

```

609
610 [yolo]
611 mask = 6,7,8
612 anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,98, 156,198, 373,326
613 classes=4
614 num=9
615 jitter=.3
616 ignore_thresh = .7
617 truth_thresh = 1
618 random=1
619

```

Figure 4.4 Illustration of "classes" adjustment in the configuration file

Max Batches

The number of batches has been changed from 500,200 to 8,000 for the Darknet-53 network and from 10,000 to 8,000 for the Resnet-152 backbone network (Figure 4.5) The total number of iterations represents the number of times the model processes a batch of data during training. The maximum number of batches refers to the number of batches per epoch. The number of maximum batches is provided by (Sujee et al., 2020):

$$Max_batches = 2000 \times n \quad (4.1)$$

where,

- n is the number of classes
- 2000 is a proposed value from YOLOv3 (Alexey, 2024)

Given that in this case we have four classes, the number of max batches was set to 8,000 in the configuration file. Note that in the original YOLOv3 model 500,200 max batches is used, although the COCO dataset, which was originally used for training, includes 80 classes. Based on Equation 4.1, max batches should be 160,000 (2000×80) instead of 500,200. This happens because our UAV dataset has fewer classes and images in the training set rather than COCO dataset which has approximately 118,000 training images. If we had followed the COCO dataset approach, an increase in the number of iterations would have possibly led to overfitting due to the corresponding increase in the number of epochs (Ghosh et al., 2021). This approach would have slowed down the training process of our study.

Therefore, to achieve a better performance we followed Equation 4.1, which might not be the ideal for large datasets like COCO. The same approach was followed for Resnet-152.

An epoch represents one complete pass through the entire training dataset by the model. After each epoch, the model updates its weights based on the errors it made during that pass. Since a single pass

through the data is usually not enough for the model to learn effectively, multiple epochs are typically required to achieve optimal performance.

For estimating the number of epochs, we need to firstly calculate the number of iterations per epoch from the formula below:

$$\begin{aligned} \text{number of iterations per epoch} &= \frac{\text{number of images of the training set}}{\text{batch size}} & (4.2) \\ &= \frac{13,041}{64} = 203,77 \end{aligned}$$

$$\text{number of epochs} = \frac{\text{number of max batches}}{\text{number of iterations per epoch}} = \frac{8,000}{203,77} = 39,3 \approx 40 \quad (4.3)$$

where,

batch size refers to the number of training examples processed by the model in one iteration before updating its internal parameters (weights).

```

1 [net]
2 # Testing
3 #batch=1
4 #subdivisions=1
5 # Training
6 batch=64
7 subdivisions=16
8 width=352
9 height=352
10 channels=3
11 momentum=0.9
12 decay=0.0005
13 angle=0
14 saturation = 1.5
15 exposure = 1.5
16 hue=.1
17
18 learning_rate=0.001
19 burn_in=1000
20 max_batches = 8000
21 policy=steps
22 steps=6400,7200
23 scales=.1,.1
24
25 [convolutional]
26 batch_normalize=1
27 filters=32
28 size=3
29 stride=1
30 pad=1
31 activation=leaky

```

Figure 4.5 Illustration of "max_batches" in the configuration file

Filters

The filters are placed in front of the convolutional layer, in each detection head, and are used for feature extraction/detection to produce the characteristics of the anchor boxes. They are numeric matrices the

dimensions of which are set during initialization and cannot be modified afterwards. Every filter consists of weights/numeric values that are adapted during training.

In Darknet-53, we reduced this number from 255 to 27 (Figure 4.6). At the same time the number of filters in Resnet-152 was increased from 24 to 36 (Table 4.2) The number of filters is computed by the equation below (Tepteris et al., 2023):

$$\text{Number of filters} = (n_{\text{classes}} + 5) \times n_{\text{anchor boxes}} \quad (4.4)$$

where,

- n_{classes} is the number of classes
- $n_{\text{anchor boxes}}$ represents the number of anchor boxes in YOLOv3
- 5 represents the 4 characteristics of the bounding box plus an objectness score

Darknet-53 utilizes 3 anchor boxes in each head, therefore the number of filters was set to 27 (Figure 4.6). In the configuration files that Resnet-152 was selected as a backbone network, the number of filters takes the value of 36 because it utilizes 4 anchor boxes in each head.

```

602 [convolutional]
603 size=1
604 stride=1
605 pad=1
606 filters=27
607 activation=linear
608
609
610 [yolo]
611 mask = 0,7,8
612 anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
613 classes=4
614 num=9
615 jitter=.3
616 ignore_thresh = .7
617 truth_thresh = 1
618 random=1

```

Figure 4.6 Illustration of "filters" in the configuration file

Steps

The number of steps specifies for how many steps the learning rate will remain constant. This parameter is suggested to be 80% and 90% of the maximum value of the batch. In our case the maximum number of batches is 8000. Therefore, this means that the steps will take the values of (Sujee et al., 2020) = see Figure 4.7.

$$0.8 \times \text{max batches} = 0.8 \times 8000 = 6400 \quad (4.5)$$

$$0.9 \times \text{max batches} = 0.9 \times 8000 = 7200 \quad (4.6)$$

```

1 [net]
2 # Testing
3 #batch=1
4 #subdivisions=1
5 # Training
6 batch=64
7 subdivisions=16
8 width=352
9 height=352
10 channels=3
11 momentum=0.9
12 decay=0.0005
13 angle=0
14 saturation = 1.5
15 exposure = 1.5
16 hue=.1
17
18 learning_rate=0.001
19 burn_in=1000
20 max_batches = 8000
21 policy=steps
22 steps=6400,7200
23 scales=.1,.1

```

Figure 4.7 Illustration of “steps” in the configuration file

The same process is also followed in order to calculate the accurate number of steps for the Resnet-152 backbone. In summary, these parameter adjustments were made to improve the performance and efficiency of the YOLOv3 model for the specific object detection task.

4.3.2 Hyperparameters associated with the YOLOv3 architecture and functionality

Hyperparameters are known for their significant role in shaping the model's performance. In this Section, we overview of YOLOv3 hyperparameters related to both the backbone network and the YOLO heads.

Table 4.3 and Table 4.4 illustrate the hyperparameters of the backbone network. They briefly describe their functionality and explain the impact on the YOLOv3 algorithm when their values are altered.

Table 4.3 Hyperparameters in the backbone network related to architecture

Hyperparameter	Functionality	Impact
Backbone Depth	Number of layers in the backbone network for feature extraction (Redmon and Farhadi, 2018).	Increasing depth improves feature extraction but raises computational cost and risk of overfitting. Decreasing depth reduces feature complexity and detection capability
Convolution Kernel Size	Size of the filter applied to the input data (Öztürk et al., 2018).	Changing size affects receptive field and feature extraction but may not yield significant improvements and could increase computational demands
Convolutional Stride	Defines how much the filter moves across the input image (Riad et al., 2022).	Default strides balance resolution and efficiency. Changing strides affects feature map resolution which can impact both the model's accuracy and processing speed. Larger strides reduce the resolution but increase efficiency, while smaller strides are

		able to capture more details but require more computational resources.
Dilated Convolution	Introduces gaps in kernels to cover larger receptive fields (Zhang et al., 2017).	YOLOv3 uses standard convolutions. Adding dilated convolutions might capture more context and improve performance

Table 4.4 Hyperparameters in the backbone network related to training

Hyperparameter	Functionality	Impact of Changing Values
Input Size	Dimensions of the input images fed into the network (Kamal, 2021).	Affects detection accuracy and computational cost. Larger sizes improve detail but increase processing time; smaller sizes reduce detail but speed up processing
Input Normalization	Scaling of pixel values to a specific range (e.g., [0,1]) (Aksu et al., 2019).	Ensures consistency and stability during training. Improper normalization can affect model's performance
Data Augmentation	Techniques to artificially increase training data (e.g., flips, rotations) (van Dyk and Meng, 2001).	Enhances model robustness and generalization
Image Distortion	Altering image properties (e.g., blurring) (Buczowski and Stasiński, 2019).	Can improve model robustness by simulating real-world variations. Excessive distortion can affect the quality of training data

Table 4.5 and Table 4.6 overview the hyperparameters of the YOLO heads. They briefly describe their functionality and explains the impact on the YOLOv3 algorithm when their values are altered.

Table 4.5 Hyperparameters in the YOLO heads related to architecture

Hyperparameter	Functionality	Impact
Number of Layers	Number of layers in the YOLO detection heads responsible for predicting bounding boxes and class probabilities (Redmon and Farhadi, 2018).	Increasing the number of layers can improve feature representation but may add complexity and risk of overfitting. YOLOv3 uses a fixed number of layers for a balanced approach
Dropout	Regularization technique where a subset of layers is randomly dropped (or disabled) during training to prevent overfitting (Alexey, 2024).	Adding dropout can help prevent overfitting but may reduce model capacity if set too high.

Spatial Pyramid Pooling (SPP)	Technique used to handle varying object scales and improve feature extraction by pooling over different spatial resolutions (He et al., 2015).	Can enhance the model's ability to detect objects at multiple scales, but YOLOv3 does not use SPP in its default architecture.
Path Aggregation Network (PAN)	Enhances feature representation by combining features from different layers (Liu et al., 2018).	Improves feature combination and detection performance, but YOLOv3 does not include PAN by default.
Multi-scale Output	Predicts bounding boxes at multiple scales to detect objects of various sizes (Cai et al., 2016)	Helps in detecting objects at different scales but increases computational complexity. YOLOv3 uses multi-scale outputs by default for better detection

Table 4.6 Hyperparameters in the YOLO heads related to architecture related to training

Hyperparameter	Functionality	Impact of Changing Values
Batch Size	Number of images processed in one training iteration (Redmon et al., 2016).	Larger batch sizes improve training stability and use GPU resources more effectively but require more memory. Smaller batch sizes may reduce memory usage but could increase training time.
Learning Rate	Learning rate controls the step size during optimization (Igiri et al., 2021).	Higher learning rates can speed up training but may lead to instability. Lower rates can offer more stability but may lead to local minimums
Subdivision	The process of dividing a batch of training data into smaller subsets to manage memory during training. The model's weights are updated after all subsets of the batch have been processed (Liu et al., 2020).	Subdividing the batch allows training on larger effective batch sizes with less GPU memory. Higher subdivisions reduce memory usage but may slow down training.
Anchor Boxes	Predefined bounding box sizes used to predict object locations (Kamal, 2021)	Properly tuned anchor boxes improve detection performance. Not well defined anchors can reduce detection accuracy
IoU Threshold	Intersection over Union threshold for considering a	Higher thresholds increase precision but might miss some

	predicted box as a positive detection (Kamal, 2021)	detections. Lower thresholds increase recall but may include false positives
Confidence Threshold	Minimum confidence score required for a prediction to be considered (Wenkel et al., 2021)	Higher thresholds reduce false positives but might miss some true positives. Lower thresholds may increase false positives
NMS Threshold	Threshold for Non-Maximum Suppression to filter out overlapping bounding boxes (Bodla et al., 2017).	A higher NMS threshold results in fewer boxes being suppressed, which might increase overlap. A lower threshold suppresses more boxes, reducing overlap
Activation Function	Function applied to the output of neurons (e.g., ReLU, Leaky ReLU) (Sharma et al., 2020).	Different activation functions can impact the training dynamics and final performance
Loss Function	Function used to measure the difference between predicted and ground truth values (Kamal, 2021)	YOLOv3 uses a combination of localization, confidence, and classification losses. Adjusting this can affect how well the model learns
Freeze Layers	Layers that are not updated during training (frozen layers) (Brock et al., 2017).	Freezing layers can help in transfer learning or stabilize training. Unfreezing layers allows the model to adapt more fully to new data
Focal Loss	Loss function that reduces the relative loss for well-classified examples and puts more focus on hard, misclassified examples (Mukhoti et al., 2020).	Improves detection of small or difficult objects by addressing class imbalance.
Input Color Space	Color space of the input images (e.g., RGB, BGR) (Díaz-Cel et al., 2019).	Consistent color space helps in accurate feature extraction
Transfer Learning	Using pre-trained weights to initialize the model (Ribani and Marengoni, 2019).	Can speed up feature merging and improve performance. Without transfer learning, training from scratch may be required
Batch Normalization	Normalization technique to stabilize and accelerate training (Ioffe and Szegedy, 2015b).	Helps in faster feature inheritance and improves model stability. YOLOv3 includes batch normalization in its default architecture

Group Normalization	Normalization technique applied across groups of channels (Wu and He, 2018)	Alternative to batch normalization, useful for small batch sizes. YOLOv3 uses batch normalization instead of group normalization
---------------------	--	--

4.3.3 Hyperparameters selection for our research

After investigation on the YOLOv3 hyperparameters, we ended up on selecting the six hyperparameters presented in Table 4.7: image resolution, anchor dimensions, backbone network, data augmentation, dilated convolution, and box loss. The Table also indicates the hyperparameter levels to be used on the experiments.

Table 4.7 Default and updated values of hyperparameters selected for our research

Hyperparameter Name	Default Value	Updated Value 1	Updated Value 2
Image Resolution	416 x 416	352 x 352	832 x 832
Anchor Dimensions	Default	New	
Backbone Network	Darknet-53	ResNet-152	
Data Augmentation	Default	Mosaic	
Dilated Convolution	Yes	No	
Box Loss	IoU	DioU	

More specifically:

Image Resolution

Image resolution refers to the dimensions (width x height) of the input images provided to the YOLOv3 algorithm during training. It affects the level of detail available for object detection and influences computational requirements. YOLOv3 is trained with an image resolution of 416x416. As shown in Table 4.7 we reduced the image resolution to 352x352 pixels in order to decrease the training time needed. We also used the default image input size as well as an increased level to 832x832 pixels (see also Fig. 4.8). This higher resolution helps the model to collect more information as the input progresses through the network, but it also increases training time.

Image Resolution 352x352	Image Resolution 416x416	Image Resolution 832x832
1 [net]	1 [net]	1 [net]
2 # Testing	2 # Testing	2 # Testing
3 #batch=1	3 #batch=1	3 #batch=1
4 #subdivisions=1	4 #subdivisions=1	4 #subdivisions=1
5 # Training	5 # Training	5 # Training
6 batch=64	6 batch=64	6 batch=64
7 subdivisions=16	7 subdivisions=64	7 subdivisions=64
8 width=352	8 width=416	8 width=832
9 height=352	9 height=416	9 height=832
10 channels=3	10 channels=3	10 channels=3
11 momentum=0.9	11 momentum=0.9	11 momentum=0.9
12 decay=0.0005	12 decay=0.0005	12 decay=0.0005
13 angle=0	13 angle=0	13 angle=0
14 saturation = 1.5	14 saturation = 1.5	14 saturation = 1.5
15 exposure = 1.5	15 exposure = 1.5	15 exposure = 1.5
16 hue=.1	16 hue=.1	16 hue=.1

Figure 4.8 Illustration of the three different image resolution options in the configuration file

Backbone Network

Backbone Network, as mentioned in chapter 3, serves as the feature extraction component of the algorithm, responsible for processing input images and extracting hierarchical features that facilitate object detection. The backbone network architecture used in YOLOv3 is Darknet-53, which is also used in experiments. We also experimented with ResNet-152. Resnet-152 consists of 152 layers instead of the 53 layers of Darknet-53.

Anchor Dimensions

Anchor dimensions define the default bounding boxes that are used to predict object locations and sizes. Once the anchor dimensions are determined, they are used during training to predict bounding boxes for detected objects. YOLOv3 predicts bounding box coordinates relative to these anchor boxes, along with confidence scores for object presence and class probabilities, as it mentioned in Chapter 3. The default anchor dimensions that are used in the yolov3.cfg file with Darknet-53 as a backbone layer are:

[10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326] as illustrated in Figure 4.9.

```
[yolo]
mask = 0,1,2
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
classes=4
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

Figure 4.9 Illustration of the default anchor dimensions for the YOLO head responsible for detecting small objects

Each detection head uses a different set of boxes to predict objects, selected by the "masks" parameter in each head. The "masks" parameter is an index containing three values that specify the anchor boxes used in each grid cell. The green box in Figure 4.9 illustrates a small detection head configured with the "masks" index [0, 1, 2]. Specifically, index 0 corresponds to anchor box dimensions of 10x13, index 1 to 16x30, and index 2 to 33x23. The medium detection head is configured with the "mask" index [4, 5, 6] (Figure 4.10) corresponding to anchor box dimensions 30x61, 62x45, 59x119 respectively. Finally, a large detection head

is configured with the “mask” index [6, 7, 8] (Figure 4.11) corresponding to anchor dimensions 116x90, 156x198, 373x326 respectively.

```
[yolo]
mask = 1,4,5
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
classes=4
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

Figure 4.10 Illustration of the default anchor dimensions for the YOLO head responsible for detecting medium objects

```
[yolo]
mask = 6,7,8
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
classes=4
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

Figure 4.11 Illustration of the default anchor dimensions for the YOLO head responsible for detecting large objects

Even though anchor boxes are fine-tuned during training to fit objects better, it is important to start with good initial anchor boxes. This helps them get refined more effectively and quickly during training. In YOLOv3, the original anchor boxes were created using the k-means algorithm on the COCO dataset. This algorithm groups objects by their size and shape, and finds the best anchor box sizes (Oti et al., 2021). We used the same algorithm for our UAV dataset and applied by using the command in Table 4.8. The function `calc_anchors` (Table 4.8) analyzes the dataset's bounding boxes and clusters them to generate optimized anchor boxes by specifying the four parameters below:

`calc_anchors cfg/"NAME_OF_THE_DATA_FILE".data`, which specifies the configuration file that contains information about the dataset (e.g., the path to the images and labels).

`num_of_clusters "NUMBER_OF_CLUSTERS"`, which defines the number of clusters (anchor boxes) to calculate. These clusters are determined using a k-means clustering to optimize the anchor boxes for the dataset.

`-width "NUMBER_OF_IMAGE_WIDTH"` and `-height "NUMBER_OF_IMAGE_HEIGHT"` specify the width and height of the images for which the anchor boxes will be optimized. This helps improve the detection accuracy of the YOLO model by better matching the size and shape of the objects in the dataset.

Table 4.8 Representation of the use of the k-means algorithm

Command	<code>./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"</code>
Example	<code>./darknet detector calc_anchors cfg/UAV_65.data -num_of_clusters 9 -width 832 -height 832</code>

We applied k-means clustering to calculate nine new optimal anchor box sizes (three aspect ratios for each head) based on our UAV dataset. This algorithm was used to find the best-fitting anchor box sizes for objects in our images at specific resolutions. As shown in Tables 4.9 and 4.10, we adjusted the “anchors” hyper-parameter across all the detection heads in our study to match the image resolution derived. Different anchor box sizes were applied for images with resolutions of 352x352, 416x416 and 832x832 pixels, respectively on both backbone networks, Darknet-53 (Table 4.9) and Resnet-152 (Table 4.10). Therefore, we conducted the k-means clustering algorithm three times to identify the 9 optimal anchor boxes in the Darknet-53 backbone case and the 12 optimal anchor boxes in the ResNet-152 backbone case (Wu et al., 2019). Note that Resnet-152 has 12 anchor boxes in total, as it is a deeper network utilizing more layers (Alexey, 2024)

```
[yolo]
mask = 6,7,8
anchors = 2, 5, 7, 7, 5, 13, 10, 12, 9, 21, 17, 16, 17, 31, 30, 30, 44, 61
classes=4
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

Figure 4.12 Illustration of the updated anchor dimensions in the configuration file

The updated values for anchor boxes in Darknet-53 backbone network, based on the three different image resolutions, are illustrated in the table below. In Table 4.9, the first three pairs ([10,13, 16,30, 33,23, ...]) correspond to the three anchor boxes of the YOLO head that detects small objects. The next three pairs ([..., 30,61, 62,45, 59,119, ...]) are referring to the three anchor boxes of the YOLO heads responsible for detecting medium objects and the last three pairs ([..., 116,90, 156,198, 373,326, ...]) for the YOLO heads detecting large objects. The same approach is followed for the rest of the image resolutions selected for our study.

Table 4.9 Anchor boxes for Darknet-53

Image Resolution	Anchor Boxes
YOLOv3 with Darknet-53 as backbone (trained on COCO dataset)	
416x416	[10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326]
YOLOv3 with Darknet-53 as backbone (trained on UAV datasets)	
352X352	[2,5, 7,7, 5,13, 10,12, 9,21, 17,16, 17,31, 30,30, 44,61]
416X416	[7, 10, 11, 11, 8, 18, 12, 17, 18, 15, 15, 24, 25, 21, 23, 34, 40, 59]
832X832	[13, 22, 21, 25, 17, 39, 33, 24, 27, 36, 32, 52, 45, 39, 48, 69, 82,121]

The updated values for anchor boxes in Resnet-152 backbone network, based on the three different image resolutions, are illustrated in the table below. In Resnet-152 backbone network we have four anchor boxes. Therefore, in Table 4.10, the first four pairs ([8,8, 10,13, 16,30, 33,23, ...]) are responsible for the four anchor boxes of the yolo head that detects small objects. The next four pairs ([..., 32,32, 30,61, 62,45, 59,119, ...]) are referring to the four anchor boxes of the yolo heads responsible for detecting medium objects and the last four pairs ([..., 80,80, 116,90, 156,198, 373,326, ...]) for the yolo heads detecting large objects. The same approach is followed for the rest of the image resolutions selected for our study.

Table 4.10 Anchor boxes for Resnet-152

Image Resolution	Anchor Boxes
YOLOv3 with ResNet-152 as backbone (trained on COCO dataset)	
416x416	[8,8, 10,13, 16,30, 33,23, 32,32, 30,61, 62,45, 59,119, 80,80, 116,90, 156,198, 373,326]
YOLOv3 with ResNet-152 as backbone (trained on UAV datasets)	
352x352	[2,4, 3,9, 7,7, 6,13, 10,12, 8,22, 16,12, 14,19, 25,20, 18,33, 34,39, 50,71]
416x416	[7, 10, 11, 11, 8, 18, 12, 17, 18, 15, 15, 24, 25, 21, 23, 35, 42, 59]
832x832	[13, 22, 20, 24, 17, 39, 32, 23, 26, 33, 31, 49, 44, 38, 47, 68, 80,118]

Data Augmentation

Data Augmentation artificially increases the diversity of training data by applying various transformations to the original images. Techniques such as random cropping, flipping, and rotation are incorporated in the data augmentation hyperparameter. YOLOv3 does not explicitly specify a standard set of data augmentation techniques.

To differentiate our experiments and investigate how efficiently the YOLOv3 algorithm will work after the introduction of data augmentation, we added the mosaic augmentation. Mosaic augmentation is a method that combines multiple images to form a single mosaic image (Figure 4.13). This mosaic image is then exposed to random transformations like flipping, scaling, and translation. This creates a synthetic training sample that contains information from multiple original images, effectively increasing the variability and complexity of the training data (Li et al., 2023).

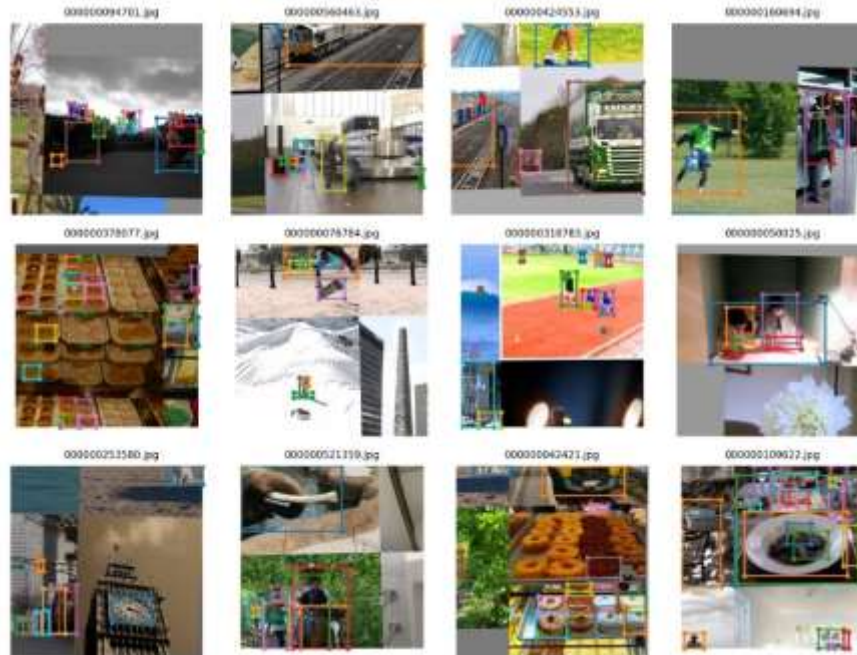


Figure 4.13 Illustration of mosaic augmentation (Alexey, 2020)

The mosaic augmentation process in YOLOv3 typically follows these steps (Bochkovskiy et al., 2020):

1. Source Image Selection: Four images are randomly chosen from the training dataset
2. Mosaic Image Creation: The selected images are organized into a mosaic layout, with each image filling one quadrant
3. Random Transformations: Various transformations, including flipping and scaling are applied randomly to the entire mosaic image
4. Bounding Box Adjustments: Bounding box coordinates and object labels within the mosaic image are modified to align with their new positions and orientations
5. Training Phase: The augmented mosaic image is utilized as input during training to refine the parameters of the YOLOv3 model.

The modification applied to the configuration (cfg) files includes the introduction of a parameter denoted as "mosaic," set to a value of 1 (mosaic=1) (Figure 4.14). This parameter is added within the section that specifies how the training data is augmented during each epoch of training (Shorten and Khoshgoftaar, 2019).

```

1 [net]
2 # Testing
3 #batch=1
4 #subdivisions=1
5 # Training
6 batch=64
7 subdivisions=64
8 width=832
9 height=832
10 channels=3
11 momentum=0.9
12 decay=0.0005
13 angle=0
14 saturation = 1.5
15 exposure = 1.5
16 hue=.1
17
18
19 learning_rate=0.0005
20 burn_in=1000
21 max_batches = 8000
22
23 policy=sgdr
24 sgdr_cycle=1000
25 sgdr_mult=2
26 steps=3200,4800,6400,7200
27 #scales=1, 1, 0.1, 0.1
28
29 #Data augmentation-mosaic
30 mosaic=1
31
32 [convolutional]
33 batch_normalize=1
34 filters=64
35 size=7
36 stride=2
37 pad=1
38 activation=leaky
39
40 [maxpool]
41 size=2
42 stride=2
43

```

Figure 4.14 Illustration of “mosaic” in the configuration file

Dilated Convolution

Dilated convolution is a technique that involves modifying the convolutional kernel by inserting zeros between its weights. Normally, a convolutional kernel is a small grid of numbers (weights) that processes over the image to detect features, such as patterns, edges, textures etc. In dilated convolutions, zeros are inserted between these weights, increasing the distance between the points the kernel captures from the image (see Figure 4.15). This increases the receptive field of each neuron, meaning each neuron can “see” a larger area of the input image. As a result, the neuron can gather more information without increasing the number of parameters or computational resources. YOLOv3 does not use dilated convolutions by default. However, applying different dilation rates in various layers can help the network capture features at multiple scales (Gashi et al., 2017).

The default setting in YOLOv3 configuration file is dilation=1, even though it’s not written in the configuration file. A dilation rate of 1 means that the convolutional kernel operates normally without any added zeros, functioning just like a standard convolution (Zhang et al., 2017).

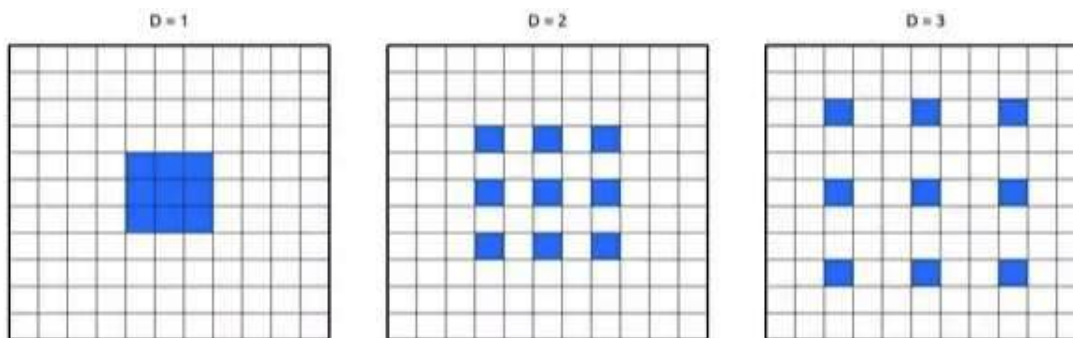


Figure 4.15 Dilated convolution filters with dilation rates $D = 1$, $D = 2$, $D = 3$ respectively (Heffels and Vanschoren, 2020)

For our research, we modified the convolutional layers in the YOLOv3 configuration file to include dilation, by setting two different values 2 and 3 in two of the heads, respectively. More specifically:

- **Dilation=2:** The model inserts one zero between each weight in the convolutional kernel. This doubles the receptive field, meaning the network can "see" more of the image around each point it processes. This can be useful for detecting larger objects (Figure 4.15).
- **Dilation=3:** The model adds two zeros between each weight, tripling the receptive field. This allows the network to capture even more context, which can help in detecting even larger objects or understanding more complex scenes (Figure 4.15).

Table 4.11 Illustration of 'dilation' in the configuration file

Dilation = 3	Dilation = 2	Dilation = 3
<code>[convolutional]</code> <code>dilation=3</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=1024</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>dilation=2</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=512</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>dilation=1</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=256</code> <code>activation=leaky</code>
<code>[convolutional]</code> <code>batch_normalize=1</code> <code>filters=512</code> <code>size=1</code> <code>stride=1</code> <code>pad=1</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>batch_normalize=1</code> <code>filters=256</code> <code>size=1</code> <code>stride=1</code> <code>pad=1</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>batch_normalize=1</code> <code>filters=128</code> <code>size=1</code> <code>stride=1</code> <code>pad=1</code> <code>activation=leaky</code>
<code>[convolutional]</code> <code>dilation=3</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=1024</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>dilation=2</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=512</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>dilation=1</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=256</code> <code>activation=leaky</code>
<code>[convolutional]</code> <code>batch_normalize=1</code> <code>filters=512</code> <code>size=1</code> <code>stride=1</code> <code>pad=1</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>batch_normalize=1</code> <code>filters=256</code> <code>size=1</code> <code>stride=1</code> <code>pad=1</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>batch_normalize=1</code> <code>filters=128</code> <code>size=1</code> <code>stride=1</code> <code>pad=1</code> <code>activation=leaky</code>
<code>[convolutional]</code> <code>dilation=3</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=1024</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>dilation=2</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=512</code> <code>activation=leaky</code>	<code>[convolutional]</code> <code>dilation=1</code> <code>batch_normalize=1</code> <code>size=3</code> <code>stride=1</code> <code>pad=1</code> <code>filters=256</code> <code>activation=leaky</code>

Specifically, for the first YOLO head, we set dilation=3 and for the second YOLO head we used dilation=2. For the last YOLO head, we kept the default dilation=1. We did this in the hope that the model detect objects of different sizes more effectively. The first YOLO head, with dilation=3, looks at a larger area of the image, which is useful for spotting bigger objects or understanding the overall scene. The second YOLO head, with dilation=2, focuses on medium-sized features, balancing between the big picture and smaller details. Finally, the third YOLO head, with dilation=1, focuses on the smallest details, helping to accurately detect small objects. This approach allows the network to first gather broad information and then gradually zoom in, making it better at detecting objects of all sizes.

Box Loss

In YOLOv3, the box loss is the part of the loss function that is responsible for measuring how well the predicted bounding boxes match the ground truth bounding boxes. By minimizing box loss during training, YOLOv3 predicts bounding boxes that accurately contain objects and assign high confidence scores to grid cells that contain objects. In the original YOLOv3 configuration file, the default loss function is not

described as a single function but is defined by the structure of the network. For further information on YOLOv3 loss function see (Tepteris et al., 2023).

Intersection over Union (IoU) and Distance Intersection over Union (DIoU) are metrics used to evaluate and improve the accuracy of these bounding box predictions, with DIoU providing additional geometric context to enhance performance.

More specifically:

1. **Intersection over Union (IoU):** IoU measures the overlap between a predicted bounding box and the ground truth bounding box. IoU-based loss functions focus on the area of overlap between the two boxes, providing a more direct measure of how well the predicted box matches the ground truth in terms of spatial dimensions. More specifically IoU loss is defined by the formula:

$$IoU\ Loss = 1 - IoU = 1 - \frac{Area\ of\ Intersection}{Area\ of\ Union} \quad (4.7)$$

where IoU ranges from 0 to 1 (area of Intersection: The overlapping area between the predicted and ground truth bounding boxes. Area of Union: The total area covered by both the predicted and ground truth bounding boxes). An IoU value of 1 means a perfect overlap, while a value closer to 0 indicates minimal overlap. By minimizing this IoU loss, we encourage the model to maximize the overlap between the predicted and ground truth boxes. Nevertheless, IoU faces some limitations. Although it provides a measure of overlap, it doesn't consider the distance between the centers of the predicted and ground truth boxes, which can be an issue when boxes have similar IoU values but different placements.

2. **Distance IoU (DIoU):** Extends IoU by taking into account the distance between the center points of the predicted and ground truth bounding boxes. DIoU considers not only the overlap but also the distance between the centers of the boxes, addressing situations where two boxes may have similar IoU values but different placements. More specifically DIoU loss is calculated by the formula:

$$DIoU\ Loss = 1 - IoU + \frac{d^2(c_p, c_g)}{c^2} \quad (4.8)$$

where,

- $d^2(c_p, c_g)$ is the Euclidean distance between the center points of the predicted box c_p and the ground truth box c_g .
- c is the diagonal length of the smallest enclosing box that contains both the predicted and the ground truth boxes.

DIoU encourages the model to predict boxes that are not only overlapping but also more accurately located (Zheng et al., 2020).

The loss function, including box loss, is usually defined within the training script. By adjusting certain parameters and settings in the YOLOv3 cfg files, box loss can be influenced.

```
[yolo]
mask = 0,7,8
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
classes=4
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
iou_thresh=0.5
iou_normalizer=0.07
iou_loss=diou
```

Figure 4.16 Illustration of the DIoU in the configuration file

Specifically, while we are setting `iou_loss = diou` in the configuration file (Figure 4.16, red box), additional parameters such as `iou_thresh` and `iou_normalizer` are specified to control aspects of the loss function and how it is applied during training (Figure 4.16, blue box). These parameters (`iou_loss=diou`, `iou_thresh=0.5`, `iou_normalizer=0.07`) would be added to all three YOLO layers. This happens because each layer requires settings that determine how the model calculates and optimizes the bounding box predictions for different scales.

In detail:

- The `iou_loss = diou` parameter specifies that the loss function for bounding box prediction should use Distance IoU (DIoU) instead of the default loss function
- The `iou_thresh=0.5` (proposed by literature: (Alexey, 2024)) parameter sets the threshold for determining whether a predicted bounding box is considered as correct (true positive (TP)) or not (false positive(FP)). During training, if the IoU between the predicted box and any ground truth box is greater than or equal to 0.5, the prediction is considered as correct. Otherwise, if the IoU is less than 0.5, the prediction is considered as incorrect. This parameter is added to help us filter out low-quality bounding boxes during training and validation processes. Setting the IoU threshold to 0.5 helps us in identifying objects accurately without being too tolerant or too strict. It is a standard, widely accepted value that has been empirically tested to perform well in most object detection scenarios.
- The `iou_normalizer=0.07` (proposed by literature: (Alexey, 2024)) used to control how much the IoU loss (related to the accuracy of the bounding boxes) affects the overall training of the model. This parameter is added to keep a balance between different types of errors the model needs to minimize. The 0.07 value was chosen based on experiments to balance training. It is low enough to ensure that while IoU contributes to defining the box locations, it doesn't skip other important tasks, like finding objects or recognizing them in an image (Alexey, 2024).

Synopsis

Summarizing all the details analyzed in this last Section, the hyperparameter values for Darknet-53 and Resnet-152 are illustrated in Table 4.12 and Table 4.13 respectively. The final configuration files for our study are conducted based on the combination of these values.

Table 4.12 Hyperparameter values for Darknet-53

<u>Hyperparameter Name</u>	<u>Darkent-53 as Backbone Network</u>		
Image Resolution	352 x 352	416 x 416	832 x 832
Anchor Dimensions	[2,5, 7,7, 5,13, 10,12, 9,21, 17,16, 17,31, 30,30, 44, 61]	[7, 10, 11, 11, 8, 18, 12, 17, 18, 15, 15, 24, 25, 21, 23, 34, 40, 59]	[13, 22, 21, 25, 17, 39, 33, 24, 27, 36, 32, 52, 45, 39, 48, 69, 82, 121]
Data Augmentation	None / Mosaic	None / Mosaic	None / Mosaic
Dilated Convolution	None / 1,2,3	None / 1,2,3	None / 1,2,3
Box Loss	IoU / DioU	IoU / DioU	IoU / DioU

Table 4.13 Hyperparameter values for Resnet-152

<u>Hyperparameter Name</u>	<u>Resnet-152 as Backbone Network</u>		
Image Resolution	352 x 352	416 x 416	832 x 832
Anchor Dimensions	[2,4, 3,9, 7,7, 6,13, 10,12, 8,22, 16,12, 14,19, 25,20, 18,33, 34,39, 50,71]	[7, 10, 11, 11, 8, 18, 12, 17, 18, 15, 15, 24, 25, 21, 23, 35, 42, 59]	[13, 22, 20, 24, 17, 39, 32, 23, 26, 33, 31, 49, 44, 38, 47, 68, 80,118]
Data Augmentation	None / Mosaic	None / Mosaic	None / Mosaic
Dilated Convolution	None / 1,2,3	None / 1,2,3	None / 1,2,3

<u>Hyperparameter Name</u>		<u>Resnet-152 as Backbone Network</u>	
Box Loss	IoU / DioU	IoU / DioU	IoU / DioU

Chapter 5 Experimental Analysis

In Chapter 5 we study how to optimize the training process of YOLOv3 by tuning selected training hyperparameters of the algorithm. We also analyze the effect of these hyperparameters and their interactions on model performance. For tuning the training hyperparameters, we conducted full factorial experiments by varying their levels and measuring their impact. For quantifying the effects of the hyperparameters on mean Average Precision (mAP), we use Analysis of Variance (ANOVA). We have also identified the best-performing combinations of hyperparameters and explained the reasons that models trained under these combinations perform better and improve object detection accuracy.

5.1 Full Factorial Design

The hyperparameters used in the experimental investigation are those identified in Chapter 4; i.e., image resolution, the use of dilated convolutions, box loss functions, anchor dimensions, backbone network types, and data augmentation methods. For the experimentation, we used a full factorial design with two levels per hyperparameter (factor) except for image resolution that was varied in three levels. The full factorial generates all possible combinations of the among the factor levels, aiming to identify the significant effects of the hyperparameters and their interactions on training effectiveness, as well as the best combination that optimizes the performance of YOLOv3 model. The full factorial design consists of 96 ($2^5 \times 3$) experiments, as shown in Table 5.1. This Table provides the full experimental design, that is all hyperparameter combinations used in our work.

Table 5.1 Multilevel factorial design of our study

A/A	Image Resolution	Dilated Convolution	Box Loss	Anchor Dimensions	Backbone Network	Data Augmentation
1	352x352	Yes	IoU	Default	Darknet-53	Default
2	352x352	Yes	IoU	Default	Darknet-53	Mosaic
3	352x352	Yes	IoU	Default	ResNet-152	Default
4	352x352	Yes	IoU	Default	ResNet-152	Mosaic
5	352x352	Yes	IoU	New	Darknet-53	Default
6	352x352	Yes	IoU	New	Darknet-53	Mosaic
7	352x352	Yes	IoU	New	ResNet-152	Default
8	352x352	Yes	IoU	New	ResNet-152	Mosaic
9	352x352	Yes	DIoU	Default	Darknet-53	Default
10	352x352	Yes	DIoU	Default	Darknet-53	Mosaic
11	352x352	Yes	DIoU	Default	ResNet-152	Default
12	352x352	Yes	DIoU	Default	ResNet-152	Mosaic
13	352x352	Yes	DIoU	New	Darknet-53	Default
14	352x352	Yes	DIoU	New	Darknet-53	Mosaic
15	352x352	Yes	DIoU	New	ResNet-152	Default
16	352x352	Yes	DIoU	New	ResNet-152	Mosaic
17	352x352	No	IoU	Default	Darknet-53	Default

A/A	Image Resolution	Dilated Convolution	Box Loss	Anchor Dimensions	Backbone Network	Data Augmentation
18	352x352	No	IoU	Default	Darknet-53	Mosaic
19	352x352	No	IoU	Default	ResNet-152	Default
20	352x352	No	IoU	Default	ResNet-152	Mosaic
21	352x352	No	IoU	New	Darknet-53	Default
22	352x352	No	IoU	New	Darknet-53	Mosaic
23	352x352	No	IoU	New	ResNet-152	Default
24	352x352	No	IoU	New	ResNet-152	Mosaic
25	352x352	No	DIoU	Default	Darknet-53	Default
26	352x352	No	DIoU	Default	Darknet-53	Mosaic
27	352x352	No	DIoU	Default	ResNet-152	Default
28	352x352	No	DIoU	Default	ResNet-152	Mosaic
29	352x352	No	DIoU	New	Darknet-53	Default
30	352x352	No	DIoU	New	Darknet-53	Mosaic
31	352x352	No	DIoU	New	ResNet-152	Default
32	352x352	No	DIoU	New	ResNet-152	Mosaic
33	416x416	Yes	IoU	Default	Darknet-53	Default
34	416x416	Yes	IoU	Default	Darknet-53	Mosaic
35	416x416	Yes	IoU	Default	ResNet-152	Default
36	416x416	Yes	IoU	Default	ResNet-152	Mosaic
37	416x416	Yes	IoU	New	Darknet-53	Default
38	416x416	Yes	IoU	New	Darknet-53	Mosaic
39	416x416	Yes	IoU	New	ResNet-152	Default
40	416x416	Yes	IoU	New	ResNet-152	Mosaic
41	416x416	Yes	DIoU	Default	Darknet-53	Default
42	416x416	Yes	DIoU	Default	Darknet-53	Mosaic
43	416x416	Yes	DIoU	Default	ResNet-152	Default
44	416x416	Yes	DIoU	Default	ResNet-152	Mosaic
45	416x416	Yes	DIoU	New	Darknet-53	Default
46	416x416	Yes	DIoU	New	Darknet-53	Mosaic
47	416x416	Yes	DIoU	New	ResNet-152	Default
48	416x416	Yes	DIoU	New	ResNet-152	Mosaic
49	416x416	No	IoU	Default	Darknet-53	Default
50	416x416	No	IoU	Default	Darknet-53	Mosaic
51	416x416	No	IoU	Default	ResNet-152	Default
52	416x416	No	IoU	Default	ResNet-152	Mosaic
53	416x416	No	IoU	New	Darknet-53	Default
54	416x416	No	IoU	New	Darknet-53	Mosaic
55	416x416	No	IoU	New	ResNet-152	Default
56	416x416	No	IoU	New	ResNet-152	Mosaic
57	416x416	No	DIoU	Default	Darknet-53	Default
58	416x416	No	DIoU	Default	Darknet-53	Mosaic

A/A	Image Resolution	Dilated Convolution	Box Loss	Anchor Dimensions	Backbone Network	Data Augmentation
59	416x416	No	DIoU	Default	ResNet-152	Default
60	416x416	No	DIoU	Default	ResNet-152	Mosaic
61	416x416	No	DIoU	New	Darknet-53	Default
62	416x416	No	DIoU	New	Darknet-53	Mosaic
63	416x416	No	DIoU	New	ResNet-152	Default
64	416x416	No	DIoU	New	ResNet-152	Mosaic
65	832x832	Yes	IoU	Default	Darknet-53	Default
66	832x832	Yes	IoU	Default	Darknet-53	Mosaic
67	832x832	Yes	IoU	Default	ResNet-152	Default
68	832x832	Yes	IoU	Default	ResNet-152	Mosaic
69	832x832	Yes	IoU	New	Darknet-53	Default
70	832x832	Yes	IoU	New	Darknet-53	Mosaic
71	832x832	Yes	IoU	New	ResNet-152	Default
72	832x832	Yes	IoU	New	ResNet-152	Mosaic
73	832x832	Yes	DIoU	Default	Darknet-53	Default
74	832x832	Yes	DIoU	Default	Darknet-53	Mosaic
75	832x832	Yes	DIoU	Default	ResNet-152	Default
76	832x832	Yes	DIoU	Default	ResNet-152	Mosaic
77	832x832	Yes	DIoU	New	Darknet-53	Default
78	832x832	Yes	DIoU	New	Darknet-53	Mosaic
79	832x832	Yes	DIoU	New	ResNet-152	Default
80	832x832	Yes	DIoU	New	ResNet-152	Mosaic
81	832x832	No	IoU	Default	Darknet-53	Default
82	832x832	No	IoU	Default	Darknet-53	Mosaic
83	832x832	No	IoU	Default	ResNet-152	Default
84	832x832	No	IoU	Default	ResNet-152	Mosaic
85	832x832	No	IoU	New	Darknet-53	Default
86	832x832	No	IoU	New	Darknet-53	Mosaic
87	832x832	No	IoU	New	ResNet-152	Default
88	832x832	No	IoU	New	ResNet-152	Mosaic
89	832x832	No	DIoU	Default	Darknet-53	Default
90	832x832	No	DIoU	Default	Darknet-53	Mosaic
91	832x832	No	DIoU	Default	ResNet-152	Default
92	832x832	No	DIoU	Default	ResNet-152	Mosaic
93	832x832	No	DIoU	New	Darknet-53	Default
94	832x832	No	DIoU	New	Darknet-53	Mosaic
95	832x832	No	DIoU	New	ResNet-152	Default
96	832x832	No	DIoU	New	ResNet-152	Mosaic

5.2 Experimental set-up and execution

The first step before we start executing the experiments was to create the appropriate configuration files (.cfg) containing all combinations of hyperparameters of Table 5.1; these files represent the models used in our study. Thus, we created 96 configuration files along with the corresponding 96 .data file (Figure 5.1) Each .data file has the same name with the .cfg file and it contains information which is essential for training.



Figure 5.1 Configuration files (.cfg) along with the corresponding data files (.data)

More specifically, as illustrated in Figure 5.2, there are five variables included within each .data file.

```
1 classes = 4
2 train = /media/deopsys/Hard_Disk/Anna/Final_Datasets/train.txt
3 valid = /media/deopsys/Hard_Disk/Anna/Final_Datasets/valid.txt
4 names = /home/deopsys/Documents/darknet/data/UAV.names
5 backup = /media/deopsys/Hard_Disk/Anna/Experiments/UAV_1_cfg/second_run
```

Figure 5.2 Variables from the first .data file corresponding to the first .cfg file in our experiments

More specifically:

- classes: the variable “classes” (Figure 5.2) indicates the number of classes on which the model is being trained. In our case we have 4 classes, person, car, long vehicle and bike (Figure 5.3).



Figure 5.3 Txt file including the object classes

- train: the variable “train” (Figure 5.2) refers to the path of a text file that contains a list of file paths for all training images. This file, named *Train.txt*, contains paths to images from the training subset used in our research.

- **valid:** the variable "valid" indicates the path to a text file that contains a list of file paths for all validation images. This file, named *Val.txt*, contains paths to images from the validation subset used in our research.
- **names:** the variable "names" (Figure 5.2) specifies the path of a file that contains the names of the object classes.
- **backup:** the variable "backup" (Figure 5.2) specifies the directory where the weights of the trained model will be stored. During training, the algorithm is configured to save these weights every 1000 iterations.

The only difference between each .data file is the "backup" path, which is modified to correspond to each .cfg file. This prevents the generated training weights from being overwritten.

The training process is executed using the following command:

Terminal Command for training process with Darknet-53 as backbone network
<pre>for i in 1 2 5 6 9 10 13 14 17 18 21 22 25 26 29 30 33 34 37 38 41 42 45 46 49 50 53 54 57 58 61 62 65 66 69 70 73 74 77 78 81 82 85 86 89 90 93 94; do ./darknet detector train /home/deopsys/Documents/darknet/cfg/Anna_experiments/cfg/UAV_\${i}.data /home/deopsys/Documents/darknet/cfg/Anna_experiments/cfg/UAV_\${i}.cfg darknet53.conv.74 -map tee UAV_output\${i}.txt; done</pre>
Terminal Command for training process with ResNet-152 as backbone network
<pre>for i in 3 4 7 8 11 12 15 16 19 20 23 24 27 28 31 32 35 36 39 40 43 44 47 48 51 52 55 56 59 60 63 64 67 68 71 72 75 76 79 80 83 84 87 88 91 92 95 96; do ./darknet detector train /home/deopsys/Documents/darknet/cfg/Anna_experiments/cfg/UAV_\${i}.data /home/deopsys/Documents/darknet/cfg/Anna_experiments/cfg/UAV_\${i}.cfg resnet152.201 - map tee UAV_output\${i}.txt; done</pre>

Figure 5.4 Execution command for conducting the experiments

These commands above (Figure 5.4) are used to automate the training process of YOLOv3 using the Darknet framework, with different backbone networks (Darknet-53 and ResNet-152).

More specifically:

- *for i in ...; do ... done:* This loop iterates through a predefined sequence of numbers (e.g., 1, 2, 5, 6, etc.) that have Darknet-53 as backbone network. Each number corresponds to a specific .cfg file
- *./darknet detector train:* This is the Darknet command to start training a certain model
- */home/deopsys/Documents/darknet/cfg/Anna_experiments/cfg/UAV_\${i}.data:* This specifies the path to the .data file for the corresponding .cfg file. The variable $\${i}$ dynamically changes the file based on the loop iteration

- `/home/deopsys/Documents/darknet/cfg/Anna_experiments/cfg/UAV_{$i}.cfg`: This specifies the path to the `.cfg` file, which contains the model architecture and hyperparameters
- `darknet53.conv.74`: This file contains the pre-trained weights for the Darknet-53 backbone network, used to initialize the training process
- `-map`: This flag enables the calculation and display of the Mean Average Precision (mAP) during training
- `| tee UAV_output{$i}.txt`: This part of the command logs the output of the training process to a file named `UAV_output{$i}.txt`, allowing for later review.

This 2nd command follows the same structure as the Darknet-53 command, but it has some differences, on loop sequence and the type of backbone network.

- `for i in ...; do ... done`: This loop iterates through a predefined sequence of numbers (e.g., 3, 4, 7, 8, etc.), that have ResNet-152 as backbone network. Each number corresponds to a specific `.cfg` file. The sequence of numbers is different, indicating that different configurations or experiments are being run
- `resnet152.201`: This file contains the pre-trained weights for the ResNet-152 backbone network, which is used to initialize the model training instead of Darknet-53.

All 96 experiments were conducted two times in order to support the analysis of the experimental results using Analysis of Variance (ANOVA). Thus, the total number of experiments was 192. It is noted that by conducting our experiments twice we are able to have a more robust training process using the same hyperparameter settings. Minor differences between the mAP results from the two runs suggest that the model is consistently trained.

As part of the overall training process, the validation process is also conducted to evaluate the weights generated by the algorithm during training, producing a mean Average Precision (mAP) result for each validation run (see Table 5.3). Our models were configured to perform the first validation run at the 400th iteration, with subsequent validations occurring every 100 iterations (e.g., 400, 500, 600, ..., 7900, 8000) up to the 8000th iteration, which marks the completion of the training process for a single model. Consequently, each experiment consisted of 77 validation runs, resulting in 77 distinct mAP values. Additionally, the trained weights were saved every 1000 iterations in the backup path specified in the `.data` file.

After completing the training process, we conducted the testing phase for each of our 96 experiments. For testing our trained models, we used the testing set from the modified UAV dataset and applied the best weights from both training runs (those achieving the highest mAP during validation). We modified the `.data` files accordingly, while the `.cfg` files remained unchanged (Figure 5.1). Specifically, for the testing phase, all `.data` files contained the information shown in Figure 5.5.

Once we set-up our experiments and created the 96 `.cfg` and `.data` files, we started the testing process by conducting a command in the terminal (Figure 5.6).

```

1 classes= 4
2 train = /media/deopsys/Hard_Disk/Anna/Final_Datasets/train.txt
3 valid = /media/deopsys/Hard_Disk/drone_files/all_experiments_1.txt
4 names = /home/deopsys/Documents/darknet/data/UAV.names

```

Figure 5.5 Information included in all .data files used for testing

In Figure 5.5, the "classes" and "names" parameters remain unchanged from those used during training. However, the "valid" parameter was modified to point to a text file (all_experiments_1.txt), which contains the file paths for all the testing images. Additionally, the "train" and "backup" parameters were excluded from the .data files used for testing, as these parameters are only relevant during the training phase.

The following command was used to execute the testing process after completing both training runs for each hyperparameter combination:

Terminal Command for testing process

```

for i in {1..96}; do ./darknet detector map
/home/deopsys/Documents/darknet/cfg/Anna/UAV.data
/home/deopsys/Documents/darknet/cfg/Anna/UAV_${i}.cfg
/media/deopsys/Hard_Disk/Anna/Experiments/UAV_${i}_cfg/UAV_${i}_best.weights -points
101 -thresh 0.25 -iou_thresh 0.5; done
>/media/deopsys/Hard_Disk/Anna/Experiments/output_${i}.txt

```

Figure 5.6 Execution command for the testing process of our experiments

The command in Figure 5.6 executes a loop that runs 96 iterations (*from* $i = 1$ to $i = 96$), each time evaluating a different YOLO model. The command `./darknet detector map` activates the testing process of YOLOv3. For each iteration, it loads the configuration file (.cfg) (`UAV_${i}.cfg`), the corresponding .data file (`UAV.data`) and the best weights file (`UAV_${i}_best.weights`) created from the first training run of the corresponding .cfg files, to compute the mean average precision (mAP) for the model using the `detector map` command. The evaluation is performed with a confidence threshold of 0.25 and an Intersection over Union (IoU) threshold of 0.5, which affects which detections are considered valid. The `-points 101` argument specifies that the precision-recall curve should be evaluated at 101 points. After each evaluation, the output is saved into a separate text file (`output_${i}.txt`), where $\${i}$ corresponds to the current iteration number, allowing the results of all 96 evaluations to be stored separately.

Therefore, in this command we have the input files, the evaluation parameters and the output files:

- Input files: For each iteration the configuration file (.cfg), the data file (.data), and trained weights (.weights) are loaded
- Evaluation parameters: The detection threshold is set to 0.25, and the IoU threshold is set to 0.5
- Output files: The results of each evaluation are finally saved in a corresponding text file in the `output_${i}.txt`.

Once the testing process of the 96 models is completed, we execute the same command using the best weights files from the second training run. As a result, from both runs of the testing process, we obtain detailed line reports that include the mAP metric for each model in each run.

During both training and testing, the model generates multiple results for evaluation, as illustrated in Table 5.2.

Table 5.2 Key class metrics during validation

Evaluation metrics for each class	<pre>class_id = 0, name = person, ap = 14.50% (TP = 5294, FP = 7957) class_id = 1, name = car, ap = 59.74% (TP = 16482, FP = 10251) class_id = 2, name = long vehicle, ap = 38.01% (TP = 4020, FP = 4275) class_id = 3, name = bike, ap = 15.59% (TP = 1479, FP = 2756)</pre>
Evaluation metrics for all classes	<pre>for conf_thresh = 0.25, precision = 0.52, recall = 0.39, F1-score = 0.44 for conf_thresh = 0.25, TP = 27275, FP = 25239, FN = 43331, average IoU = 38.68 % IoU threshold = 50 %, used 101 Recall-points mean average precision (mAP@0.50) = 0.319614, or 31.96 %</pre>

The validation report provides the following information for each class:

- *class_id*: For example, index value 1 represents the class "car"
- *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 class
- *avg IoU*: represents the average Intersection over Union (IoU) across all images in the validation set for the class.

Also, the validation report provides the following information for all classes:

- *Precision*: is the overall prediction accuracy across all classes
- *Recall*: evaluates the ability of the detector to locate the annotated 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)*: sums the Average Precision (AP) of each individual class and then divides the total AP value by the number of the classes.

Among the various metrics provided in the line reports, we utilized the mean Average Precision (mAP) to assess the performance of the models in both the validation and testing stages. However, the mAP results serve different purposes in each phase:

- During the training process, the model continuously adjusts its parameters to improve object detection in the input data. These adjusted parameters are stored in the form of weights. To ensure the model is learning effectively, these weights are periodically validated. In this intermediate validation step, the model applies the current set of trained weights to a validation dataset to evaluate its detection and classification performance. The mAP values calculated during validation are displayed in the progress chart (showing only mAP results) throughout training. These validation mAP results are crucial for verifying that the model is learning correctly. High validation mAP values during validation indicate that the model is avoiding overfitting and that training on the given dataset is proceeding as expected.
- During the testing process, the best-performing weights from training (those that achieved the highest mAP during validation) are applied to the testing dataset. The testing phase evaluates the model's detection and classification performance on independent data. If the mAP value obtained during testing is similar to the highest mAP values from validation, it confirms that the model has been trained effectively and performs as expected. Conversely, if the testing mAP value is significantly lower than the best validation mAP, it suggests that the model's performance on new data is not as robust as indicated during validation, implying that the training may not have been successful, perhaps due to overfitting.

5.3 Experimental results and analysis

In the validation and testing processes, as mentioned in Chapter 3, several key metrics such as recall, precision, F1-score, average precision (AP), and mean average precision (mAP), can be used to evaluate a model's performance. Among these metrics, we focused on mAP to analyze the outcomes of our experiments. The Average Precision (AP) metric evaluates the accuracy of object detection models by measuring how well individual objects are identified. Then the mAP metric is the mean of these AP values across all classes, providing an overall evaluation of the model's ability to detect various objects in the dataset. This makes it an effective metric for comparing overall model performance.

More specifically, we used the best mAP achieved in validation to evaluate and compare training performance. The best mAP refers to the highest mean average precision (mAP) value achieved by the validation process during training for each experiment. In testing, for each model, we used the weights corresponding to the best mAP achieved in validation. To assess model performance in testing, we used the mAP value achieved by the trained model using the independent testing dataset.

The table below presents the best mAP results from both training/validation and testing processes for the 1st and 2nd experiment runs, the average mAP across the two runs as well as the percentage difference between their values.

Table 5.3 Best mAP and the average best mAP in the 1st and 2nd run of the training and testing processes

Model Number	Training/Validation			Testing			Difference between the avg mAP values (training - testing)
	Best mAP (1 st run)	Best mAP (2 nd run)	Avg mAP between 1 st & 2 nd run	Best mAP (1 st run)	Best mAP (2 nd run)	Avg mAP between 1 st & 2 nd run	
	1		2	1		4	
	2		4	1		4	

Model Number	Training/Validation			Testing			Difference between the avg mAP values (training - testing)
	Best mAP (1 st run)	Best mAP (2 nd run)	Avg mAP between 1 st & 2 nd run	Best mAP (1 st run)	Best mAP (2 nd run)	Avg mAP between 1 st & 2 nd run	
		3	3		2	2	
		8	1		7	9	
	3		3	2		3	
	4		4	3		7	

Model Number	Training/Validation			Testing			Difference between the avg mAP values (training - testing)
	Best mAP (1 st run)	Best mAP (2 nd run)	Avg mAP between 1 st & 2 nd run	Best mAP (1 st run)	Best mAP (2 nd run)	Avg mAP between 1 st & 2 nd run	
		4	4		3	3	

The first observation from the values of Table 5.3 is that the training/validation mAP values are higher than the testing mAP values. This is expected, since the testing dataset is not part of the training dataset. In some cases, however, the difference between the training/validation and testing average (between the two replicates) mAP is significant (e.g. more than 10%).

Table 5.4 Number of objects in the training, validation and testing datasets

Number of Objects			
	Training Dataset	Validation Dataset	Testing Dataset
Number of people	170,744	16,544	16,730
Number of Cars	154,478	18,205	16,257
Number of Long Vehicles	42,068	5,012	4,986
Number of Bikes	102,893	9,195	8,967

The significant difference in mean Average Precision (mAP) between training/validation and testing can be attributed to the following reasons:

- **Overfitting:** The model may learn specific patterns in the training data but fail to generalize to unseen data. This results in high performance during validation but a notable drop during testing, indicating a lack of robustness (Montesinos López et al., 2022).
- **Class Imbalance:** The consolidated dataset contains a higher number of objects like people (213,245 total objects) (see Table 4.1) and cars (205,893 total objects) (see Table 4.1), leading the

model to focus more on these dominant classes. Underrepresented classes like long vehicles (52,961 total objects) (see Table 4.1) and bikes (130,141 total objects) (see Table 4.1) cause the model to perform worse in these classes. During testing, this imbalance in class representation can lead to the model being biased toward classes with more samples, resulting in higher mAP during training/validation but lower mAP during testing. (Crasto, 2024).

- **Dataset Composition Differences:** The testing subset, although it is part of the consolidated dataset, may not reflect the class distributions across the individual datasets, enhancing the impact of the class representation. For example, the UAV Vehicle Detection dataset includes 20,647 cars but only 241 bikes (see Table 4.1), while the Stanford dataset includes a larger number of bikes (83,600) (see Table 4.1) and has no persons at all (see Table 4.1). Meanwhile, the VisDrone2019DET dataset shows a more balanced distribution, with 158,914 persons, 46,721 long vehicles, and 46,300 bikes (see Table 4.1). If the testing subset is split disproportionately, it could include images from one dataset (e.g., more Stanford images with many bikes or more UAV Vehicle Detection images with very few bikes). In this case, the representation of classes is not accurate and it contributes to the significant difference in mAP between training and testing (Santos et al., 2024)
- **Differences in Image Complexity:** UAV images often vary in resolution, lighting, and angles. These variations can make it difficult for the model to perform consistently across all datasets, especially when testing images differ in distribution from those in the training set (Hakala et al., 2013).

A second observation has to do with the significant differences in performance of models trained under different combinations of the hyperparameters. For example, in training/validation the best performing models reach a mAP over 60% (see models 74 and 90), while others have significantly lower performance with mAP of 15% (see models 35, 51). This illustrates clearly the significance of selecting the appropriate levels of the hyperparameters in model training, and the need for optimizing these hyperparameters in order to achieve effective training of a high performing model.

As we can see in Table 5.3 the highest mAP from the validation process is achieved in the 90th experiment for both runs. The best mAP in the 90th experiment reaches 60.99% in our first run and 60.76% This indicates result consistency in terms of the mAP value. The model that achieves this performance corresponds to the following hyperparameter combination (Table 5.1):

- Image Resolution: 832x832
- Dilated Convolution: No
- Box Loss: DIoU
- Anchor Dimensions: Default
- Backbone: Darknet-53
- Data Augmentation: Mosaic

At the same, we can observe from Table 5.3 that the highest mAP during the testing process is also achieved in the 90th experiment for both runs. More specifically, the best mAP in the 90th testing

experiment reaches 52.37% in our first run and 52.65% (Table 5.3) in the second run. Achieving the highest performance across validation and testing by the same model is encouraging and points to the ability of the method to correctly tune the training hyperparameters.

The best trained model performs reliably on unseen data (testing process) with only a slight drop in performance from the training process. Furthermore, the selected levels of hyperparameters lead to an effective model across different image datasets.

The 74th experiment also displays good performance, close to that of the 90th experiment. More specifically, the best mAP in the 74th experiment reaches 60.72% in our first run and 60.50% (Table 5.3) in the second. Similarly, in the testing process the 74th experiment achieves the second good performance, like in the training process. The best mAP reaches 52.27% in the first run and 52.24% in the second run. The hyperparameter combination of this model is similar to the 90th experiment, with the only difference in the dilated convolution. The model in the 74th experiment was trained under the following hyperparameter parameter combination (Table 5.1):

- Image Resolution: 832x832
- Dilated Convolution: Yes
- Box Loss: DIoU
- Anchor Dimensions: Default
- Backbone: Darknet-53
- Data Augmentation: Mosaic

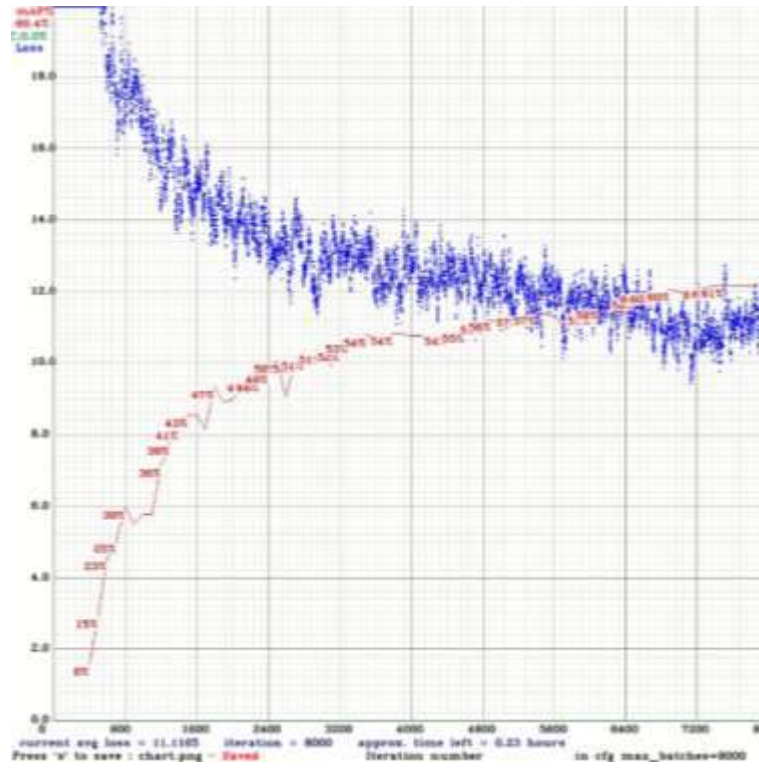


Figure 5.7 Training progress chart of the 90th experiment

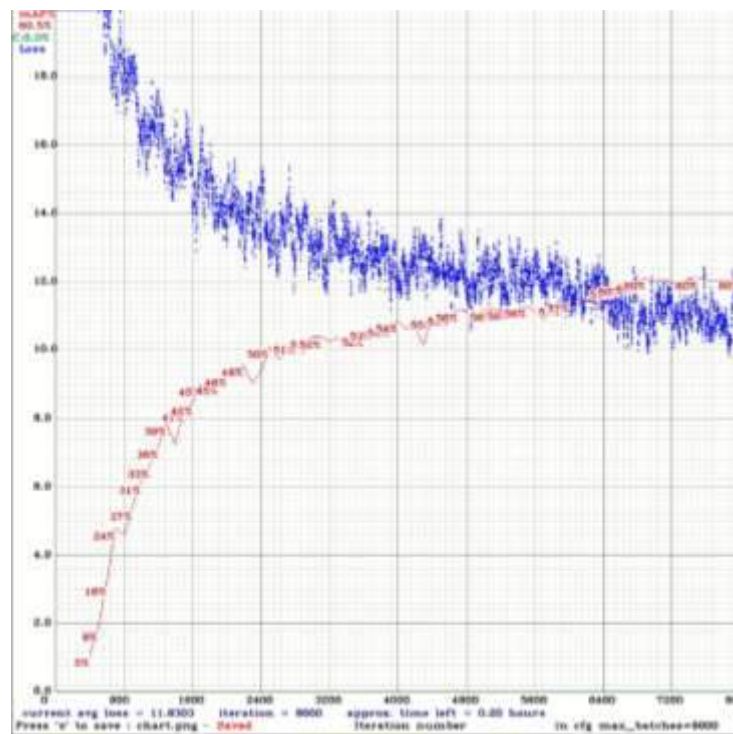


Figure 5.8 Training progress chart of the 74th experiment

Figure 5.7 illustrates the chart for the 90th experiment (the best-performing model). The x-axis represents the number of iterations as the training process progresses, while the y-axis. The blue curve of the chart represents the training loss, and the red curve shows the mAP results throughout training. The training loss curve indicates how effectively the model adapts to the dataset, with lower values reflecting better performance. This curve decreases steadily during training and stabilizes between values of 9 and 13 (Figure 5.7). The red curve represents the model's ability to detect specific objects in the validation data. In the chart is illustrated a significant improvement in mAP, increasing from 8% to 61% (Figure 5.7), which demonstrates good performance.

A similar performance is observed in the 74th experiment. The training loss curve (blue) decreases and stabilizes between values of 9 and 13 (Figure 5.8), as seen in the 90th experiment. The red curve is also quite similar to the 90th experiment, with the mAP increasing from 5% to 60% (Figure 5.8).

In both experiments, there are no indications of overfitting or underfitting, as the mAP values remain stable without significant drops or rises during training. Therefore, the model, in both cases, demonstrates strong object detection and classification abilities on the modified UAV dataset, achieving a high mAP of 61% and 60%.

5.4 Hyperparameter effects on mean Average Precision (mAP)

In our study, we used Analysis of Variance (ANOVA) in order to evaluate the impact of each selected hyperparameter and their interactions on the mAP metric. This analysis enables us to identify which hyperparameters, and interactions of them, significantly influence the training performance of the YOLOv3 model as measured by mAP.

ANOVA tests the following hypotheses:

- Null hypothesis (H_0): A factor or factor interaction has no significant effect on mAP.
- Alternative hypothesis (H_a): A factor or interaction has a significant effect on mAP.

We conducted ANOVA using the MiniTab software, analyzing the experiments listed in Table 5.1 and the best mAP results from both training runs, as shown in Table 5.5. Following this approach, it enables us to identify the key factors influencing model performance and optimizes hyperparameter tuning for mAP improvement.

The table below illustrates the design summary for our analysis.

T
a

Multilevel Factorial Design			
Design Summary			
Factors	6	Replicates	2

Multilevel Factorial Design			
Base Runs	96	Total Runs	192
Base Blocks	1	Total Blocks	1
Number of levels: 3, 2, 2, 2, 2, 2			

Table 5.5 provides a design summary of an Analysis of Variance (ANOVA) study conducted in MiniTab. This summary indicates that a multilevel factorial design was used to study how different values of the six selected hyperparameters, as well as their interactions, affect the mAP. The ANOVA will help us to determine which hyperparameters and their interactions have a statistically significant impact on the YOLOv3 model's performance.

More specifically, the table illustrates:

- **Factors: 6** – This means that six different hyperparameters (factors) were selected for our experiments, which are image resolution, dilated convolution, box loss, anchor dimensions, backbone network and data augmentation (Table 5.1)
- **Replicates: 2** – The experiments were repeated twice
- **Base runs: 96** – The experiment initially consists of 96 base runs, indicating the number of unique combinations of hyperparameter levels being tested in one iteration
- **Total runs: 192** – This reflects the total number of runs for the experiment, which is obtained by multiplying the number of base runs (96) by the number of replicates (2). So, 192 total runs were performed to evaluate the model's performance with different hyperparameter combinations
- **Base blocks: 1 & Total blocks: 1** – No blocking was used in our experiments. This means that there were no additional sources of variation (such as time, location, or different batches) that needed to be controlled by creating blocks. The entire experiment was treated as one block, meaning the focus was on the six selected hyperparameters and their interactions.
- **Number of levels: 3, 2, 2, 2, 2, 2** – This shows the number of levels for each factor (hyperparameter). One hyperparameter, image resolution, has 3 different levels (values), while the remaining five have 2 levels each.

5.4.1 ANOVA analysis on best mAP results from the training/validation process

The training (validation) results of the ANOVA analysis, in our study, are illustrated in Table 5.6, including sources of variation with their Degrees of Freedom (DF), Adjusted Sum of Squares (Adj SS), Adjusted Mean Squares (Adj MS), F-values, and p-values.

- **Degrees of Freedom (DF)** is a statistical term that indicates the number of independent values we can use in our calculations. The degrees of freedom depend on the number of levels being compared. Generally, for each parameter, the DF is calculated as the number of levels minus one. For example, in the image resolution factor, where we selected 3 levels (values) (352x352, 416x416, 832x832), the degrees of freedom would be $3-1=2$. This means that we can freely vary 2 of the values while the last one is determined by the others.

- **Adjusted Sum of Squares (Adj SS)** measures how much the variance in mAP (mean Average Precision) is explained by each hyperparameter or interaction of hyperparameters after removing the effects of other factors and their interactions in the model. For example, the Adjusted Sum of Squares for image resolution is 0.90764, meaning this amount of variance in mAP is due to changes in image resolution (Kutner, 2005).
- **Adjusted Mean Squares (Adj MS)** is the average variance related to each hyperparameter or interaction of hyperparameters, adjusted for the degrees of freedom in the model. The value of Adj MS is calculated by dividing the Adjusted Sum of Squares for image resolution by its degrees of freedom, which in our case is 2 (Kutner, 2005).

More specifically, it is calculated by the formula:

$$MS_{adj} = \frac{SS_{adj}}{DF} \quad (5.1)$$

For instance, the Adjusted Mean Square for image resolution is 0.45382. ($\frac{0.90764}{2}$).

- **F-value** measures how much a hyperparameter or interaction of hyperparameters affects the results compared to the error.

It is calculated by dividing the Adjusted Mean Square of the factor by the Adjusted Mean Square of the error, as shown in equation 5.2 below:

$$F - Value = \frac{MS_{adj}}{MS_{adj/error}} \quad (5.2)$$

A high F-value suggests that the factor significantly impacts mAP (mean Average Precision). For example, in our study, the “Backbone Network” factor has the highest F-value, 1407.32 (Table 5.6), meaning it has the greatest impact on mAP. As it is illustrated in Table 5.6, the second higher F-value is 263.36 (Table 5.6) for the “Image Resolution” factor.

- **P-value** indicates the likelihood of making an error when selecting the alternative hypothesis. A small p-value (less than 0.05) means the results are important for our study, so we reject the null hypothesis (H_0). On the other hand, a high p-value suggests that the results might have happened by chance, which means we don't reject the null hypothesis. In our case, all six factors have a p-value equal to 0 (Table 5.6), meaning that all of them significantly impact mAP (mean Average Precision). Although, factors like “Dilated Convolution” and “Anchor Dimensions” have smaller F-values than the other factors, it seems that they also affect mAP (Archdeacon, 1994).

Table 5.6 below illustrates the importance of each factor selected for our study as well as their interactions on the mAP.

Table 5.6 Analysis of Variance during training process

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	95	4.70604	0.04954	28.75	0
Linear	7	3.9989	0.57127	331.52	0
Image Resolution	2	0.90764	0.45382	263.36	0
Dilated Convolution	1	0.04359	0.04359	25.3	0
Box Loss	1	0.16424	0.16424	95.31	0
Anchor Dimensions	1	0.11865	0.11865	68.85	0
Backbone Network	1	2.42507	2.42507	1407.32	0
Data Augmentation	1	0.33971	0.33971	197.14	0
2-Way Interactions	20	0.48613	0.02431	14.11	0
Image Resolution*Dilated Convolution	2	0.0029	0.00145	0.84	0.434
Image Resolution*Box Loss	2	0.01322	0.00661	3.84	0.025
Image Resolution*Anchor Dimensions	2	0.02111	0.01056	6.13	0.003
Image Resolution*Backbone Network	2	0.00899	0.0045	2.61	0.079
Image Resolution*Data Augmentation	2	0.02654	0.01327	7.7	0.001
Dilated Convolution*Box Loss	1	0.00014	0.00014	0.08	0.777
Dilated Convolution*Anchor Dimensions	1	0.00075	0.00075	0.44	0.51
Dilated Convolution*Backbone Network	1	0.07844	0.07844	45.52	0
Dilated Convolution*Data Augmentation	1	0.00673	0.00673	3.91	0.051
Box Loss*Anchor Dimensions	1	0.02888	0.02888	16.76	0
Box Loss*Backbone Network	1	0.00018	0.00018	0.1	0.749
Box Loss*Data Augmentation	1	0.02104	0.02104	12.21	0.001
Anchor Dimensions*Backbone Network	1	0.00002	0.00002	0.01	0.91
Anchor Dimensions*Data Augmentation	1	0.00335	0.00335	1.94	0.167
Backbone Network*Data Augmentation	1	0.27382	0.27382	158.9	0
3-Way Interactions	30	0.1262	0.00421	2.44	0.001
Image Resolution*Dilated Convolution*Box Loss	2	0.00713	0.00356	2.07	0.132
Image Resolution*Dilated Convolution*Anchor Dimensions	2	0.00193	0.00097	0.56	0.572
Image Resolution*Dilated Convolution*Backbone Network	2	0.01254	0.00627	3.64	0.03

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Image Resolution*Dilated Convolution*Data Augmentation	2	0.00259	0.00129	0.75	0.474
Image Resolution*Box Loss*Anchor Dimensions	2	0.00026	0.00013	0.08	0.927
Image Resolution*Box Loss*Backbone Network	2	0.00108	0.00054	0.31	0.732
Image Resolution*Box Loss*Data Augmentation	2	0.00076	0.00038	0.22	0.803
Image Resolution*Anchor Dimensions*Backbone Network	2	0.00728	0.00364	2.11	0.127
Image Resolution*Anchor Dimensions*Data Augmentation	2	0.00794	0.00397	2.3	0.105
Image Resolution*Backbone Network*Data Augmentation	2	0.0177	0.00885	5.13	0.008
Dilated Convolution*Box Loss*Anchor Dimensions	1	0.00083	0.00083	0.48	0.489
Dilated Convolution*Box Loss*Backbone Network	1	0.00878	0.00878	5.1	0.026
Dilated Convolution*Box Loss*Data Augmentation	1	0.00006	0.00006	0.03	0.855
Dilated Convolution*Anchor Dimensions*Backbone Network	1	0.01184	0.01184	6.87	0.01
Dilated Convolution*Anchor Dimensions*Data Augmentation	1	0	0	0	0.962
Dilated Convolution*Backbone Network*Data Augmentation	1	0.00537	0.00537	3.12	0.081
Box Loss*Anchor Dimensions*Backbone Network	1	0.00828	0.00828	4.8	0.031
Box Loss*Anchor Dimensions*Data Augmentation	1	0.01123	0.01123	6.52	0.012
Box Loss*Backbone Network*Data Augmentation	1	0.01721	0.01721	9.99	0.002
Anchor Dimensions*Backbone Network*Data Augmentation	1	0.00339	0.00339	1.97	0.164
4-Way Interactions	25	0.06498	0.0026	1.51	0.081
Image Resolution*Dilated Convolution*Box Loss*Anchor Dimensions	2	0.00085	0.00042	0.25	0.782
Image Resolution*Dilated Convolution*Box Loss*Backbone Network	2	0.00604	0.00302	1.75	0.179
Image Resolution*Dilated Convolution*Box Loss*Data Augmentation	2	0.00019	0.0001	0.06	0.946
Image Resolution*Dilated Convolution*Anchor Dimensions*Backbone Network	2	0.00957	0.00478	2.78	0.067
Image Resolution*Dilated Convolution*Anchor Dimensions*Data Augmentation	2	0.00034	0.00017	0.1	0.906

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Image Resolution*Dilated Convolution*Backbone Network*Data Augmentation	2	0.0043	0.00215	1.25	0.292
Image Resolution*Box Loss*Anchor Dimensions*Backbone Network	2	0.0098	0.0049	2.84	0.063
Image Resolution*Box Loss*Anchor Dimensions*Data Augmentation	2	0.00036	0.00018	0.1	0.901
Image Resolution*Box Loss*Backbone Network*Data Augmentation	2	0.0005	0.00025	0.15	0.864
Image Resolution*Anchor Dimensions*Backbone Network*Data Augmentation	2	0.00683	0.00342	1.98	0.143
Dilated Convolution*Box Loss*Anchor Dimensions*Backbone Network	1	0.01215	0.01215	7.05	0.009
Dilated Convolution*Box Loss*Anchor Dimensions*Data Augmentation	1	0.00196	0.00196	1.14	0.289
Dilated Convolution*Box Loss*Backbone Network*Data Augmentation	1	0	0	0	0.966
Dilated Convolution*Anchor Dimensions*Backbone Network*Data Augmentation	1	0	0	0	0.993
Box Loss*Anchor Dimensions*Backbone Network*Data Augmentation	1	0.01208	0.01208	7.01	0.009
5-Way Interactions	11	0.02273	0.00207	1.2	0.298
Image Resolution*Dilated Convolution*Box Loss*Anchor Dimensions*Backbone Network	2	0.01317	0.00658	3.82	0.025
Image Resolution*Dilated Convolution*Box Loss*Anchor Dimensions*Data Augmentation	2	0.00647	0.00324	1.88	0.159
Image Resolution*Dilated Convolution*Box Loss*Backbone Network*Data Augmentation	2	0.00008	0.00004	0.02	0.977
Image Resolution*Dilated Convolution*Anchor Dimensions*Backbone Network*Data Augmentation	2	0.00032	0.00016	0.09	0.911
Image Resolution*Box Loss*Anchor Dimensions*Backbone Network*Data Augmentation	2	0.00055	0.00027	0.16	0.854
Dilated Convolution*Box Loss*Anchor Dimensions*Backbone Network*Data Augmentation	1	0.00214	0.00214	1.24	0.268
6-Way Interactions	2	0.0071	0.00355	2.06	0.133

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Image Resolution*Dilated Convolution*Box Loss*Anchor Dimensions*Backbone Network*Data Augmentation	2	0.0071	0.00355	2.06	0.133
Error	96	0.16543	0.00172		
Total	191	4.87146			

At the end of Table 5.6 two additional values are presented: Error and total:

- **Error** represents the unexplained variation in the model. It shows the difference between the actual results we observed and the results the model predicted. In our model, $SS_{error} = 0.16543$ (Adj SS) indicates the variance in mAP (mean Average Precision) that the model is not able explain.

This value is calculated by finding the sum of the squared differences between each observed result and its corresponding predicted result, as shown by the equation 5.3 below (Kutner, 2005):

$$SS_{error} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (5.3)$$

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 overall variance and is made up of the explained variance from the model and the unexplained variance (error). This value is aiming to be a reference point on evaluating the model's performance. In our model, $SS_{total} = 4.87146$ indicates the total variance in mAP (mean Average Precision). This value is calculated by finding the sum of the squared differences between each observed result and the overall average of the dependent variable, as shown in the equation 5.4 below (Kutner, 2005):

$$SS_{total} = \sum_{i=1}^n (y_i - \bar{y})^2 \quad (5.4)$$

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.

Apart from the ANOVA analysis, we also used MiniTab to generate graphic representations of our results to understand and analyze further the interaction effects between our selected hyperparameters. In our study we focused on:

- Pareto Chart of standardized effects
- Main effects plot for mAP
- Interaction plot for mAP

Pareto Chart of standardized effects

As illustrated in Figure 5.9, the Pareto chart of standardized effects demonstrates the influence and interactions of the different factors on the mAP. The factors labeled from A to F, are the hyperparameters we selected for our study. More specifically:

- A: Image Resolution
- B: Dilated Convolution
- C: Box Loss
- D: Anchor Dimensions
- E: Backbone Network
- F: Data Augmentation

The red-dashed line indicates the standardized effect value of 1.98 when the significance threshold is $\alpha = 0.05$. Hyperparameters or interactions of hyperparameters that have blue-bars extended beyond the standardized effect value (red-dashed line) have a significant impact on mAP. Longer bars illustrate a more important impact on mAP. Nevertheless, we should also mention that there is a 5% risk ($\alpha = 0.05$) of falsely identifying a factor or an interaction of factors as significant when they are not.

Based on the Pareto Chart in Figure 5.9, the significant hyperparameters and interactions of hyperparameters in the training process of our study are the following ones (starting from the most significant to the least significant):

- Backbone Network (E)
- Data Augmentation (F)
- The interaction Backbone Network (E) and Data Augmentation (F)
- Box Loss (C)
- Anchor Dimensions (D)
- Image Resolution (A)
- The interaction of Dilated Convolution (B) and Backbone Network (E)
- Dilated Convolution (B)
- The interaction of Box Loss (C) and Anchor Dimensions (D)
- The interaction of Box Loss (C) and Data Augmentation (F)
- The interaction of Image Resolution (A) and Data Augmentation (F)
- The interaction of Box Loss (C), Backbone Network (E) and Data Augmentation (F)

- The interaction of Image Resolution (A) and Anchor Dimensions (D)
- The interaction of Image Resolution (A), Backbone Network (E) and Data Augmentation (F)
- The interaction of Dilated Convolution (B), Box Loss (C), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Box Loss (C), Anchor Dimensions (D), Backbone Network (E) and Data Augmentation (F)
- The interaction of Dilated Convolution (B), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Box Loss (C), Anchor Dimensions (D) and Data Augmentation (F)
- The interaction of Image Resolution (A) and Box Loss (C)
- The interaction of Image Resolution (A), Dilated Convolution (B), Box Loss (C), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Dilated Convolution (B), Box Loss (C) and Backbone Network (E)
- The interaction of Image Resolution (A), Dilated Convolution (B) and Backbone Network (E)
- The interaction of Box Loss (C), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Dilated Convolution (B) and Data Augmentation (F)

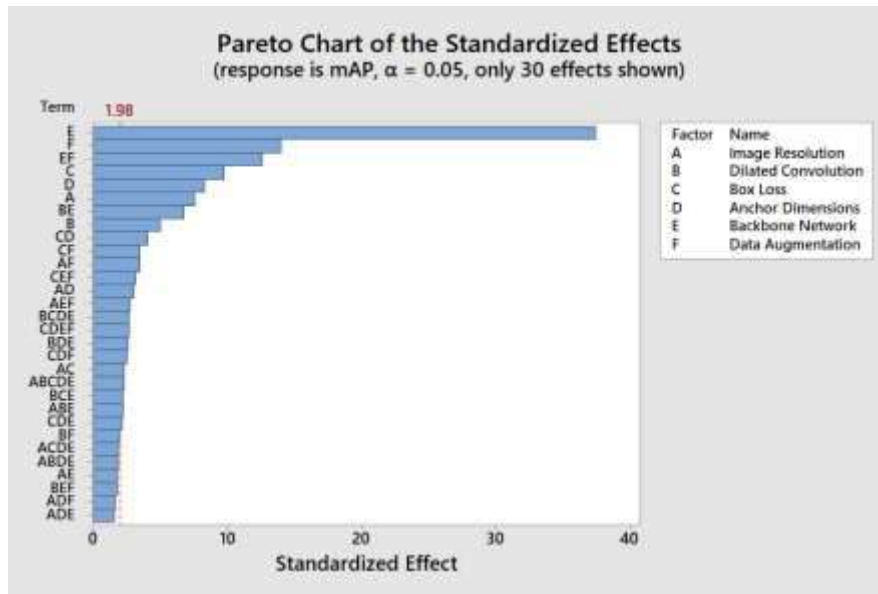


Figure 5.9 Pareto chart of the standardized effects during training process

The six remaining hyperparameter interactions do not have a significant effect on mAP, since their values are less than 1.98 (Navarro Tuch et al., 2019).

Consequently, we conclude that the most significant hyperparameter is the backbone network (E), which has the highest standardized effect, indicating that affects mAP the most. It is followed by data augmentation (F), box loss (C), anchor dimensions (D) and image resolution (A), all of which also exceed

the critical value threshold of 1.98, marked by the red vertical line. Additionally, interactions between these factors can significantly impact performance, suggesting a need for careful tuning and optimization of these hyperparameters in the training process.

Main effects plot for mAP

Figure 5.10 shows the main effects plot for the training process in our study (Kim et al., 2007), which measures the impact on mAP of the main factors A, B, C, D, E and F. By analyzing these six factors in the main effect plot in Minitab, we can conclude to the following observations below:

- Increasing image resolution from 352x352 to 416x416 improves mAP by $31\% - 27\% = 4\%$ and increasing the image resolution further from 416x416 to 832x832 improves mAP by $43.2\% - 31\% = 12.2\%$
- Adding dilated convolution in our model reduces mAP by $35.2\% - 32.2\% = 3\%$
- Changing the box loss function from IoU to DIoU improves mAP by $36.7\% - 30.8\% = 5.9\%$
- Changing anchor box dimensions from the default model values to a new set of values reduces mAP by $36.2\% - 31.2\% = 5\%$.
- Changing the backbone network from Darknet-53 (default) to ResNet-152 (new) reduces mAP by $45\% - 22.5\% = 22.5\%$.
- Adding data augmentation in our model improves mAP by $37.9\% - 29.5\% = 8.4\%$

From the result above, we conclude that the addition of dilated convolution in our model, the selection of new anchor dimensions and the alternation of the backbone network from Darknet-53 to ResNet-152 have negative effect on mAP as they reduce its value. As we can see from the plot below (Figure 5.10) the choice of ResNet-152 over Darknet-53 has the most negative impact on mAP, reducing its value by half (45% to 22.5%).

On the other hand, the increase on image resolution, the alternation of box loss function from IoU to DIoU and the addition of data augmentation improve the value of mAP, with the image resolution having the greatest impact.

Therefore, the best options for the factors/hyperparameters we selected, which positively affect the mAP value, are:

- 832x832 for image resolution
- Absence of dilated convolution
- DIoU for box loss function
- Default values for the anchor dimensions
- Darknet-53 as the backbone network
- Addition of the mosaic data augmentation technique

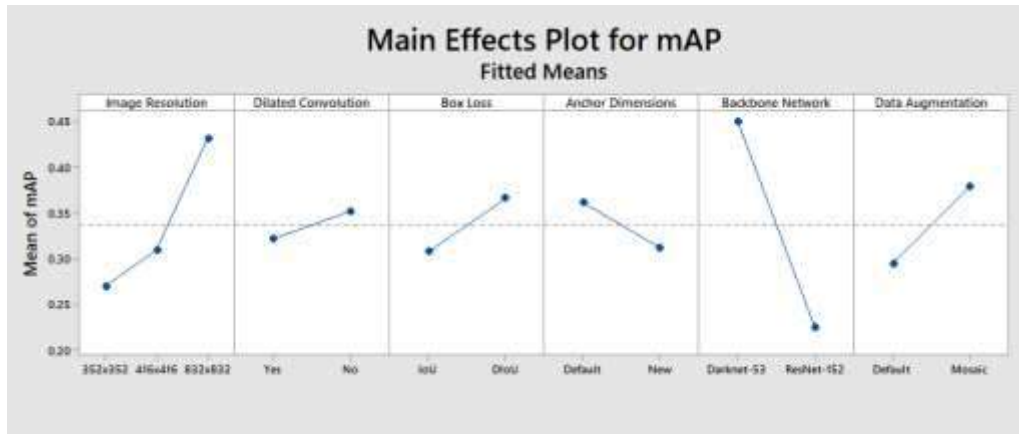


Figure 5.10 Main effects plot for mAP during training process

Interaction plot

The interaction plot illustrates the effects of factor interactions on the Mean Average Precision (mAP) (Figure 5.11). Each graph compares how two factors (like image resolution or data augmentation) interact and influence the model's accuracy. More precisely, the horizontal axis of each sub-plot represents the selected values of one hyperparameter (e.g., Image Resolution, Box Loss), illustrated with the blue and red lines. The vertical axis shows the mean of mAP for each interaction of hyperparameters. The goal is to observe how the hyperparameters interact with each other and their influence on mAP.

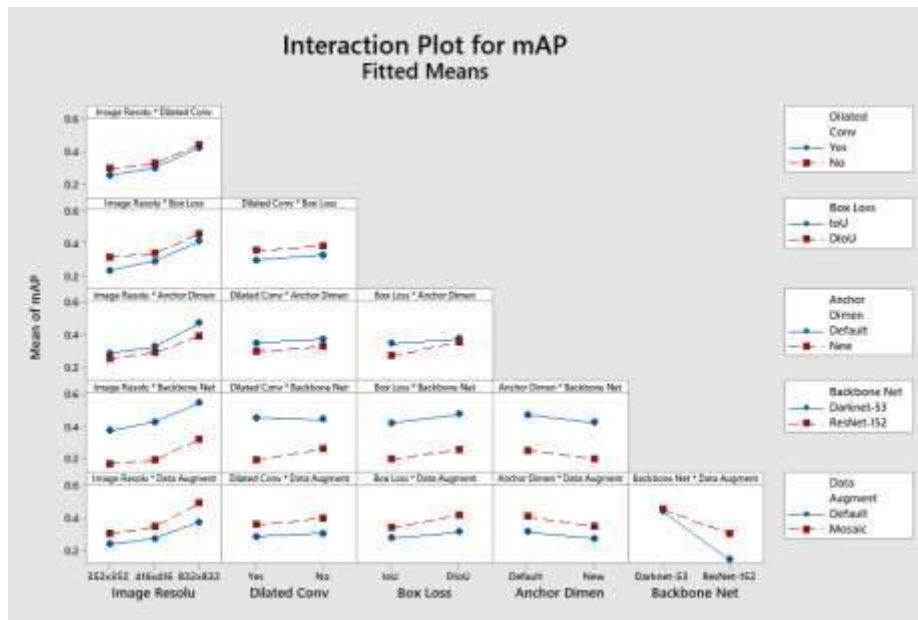


Figure 5.11 Interaction plot for mAP during training process

Table 5.7 illustrates the most optimal (best) and lest optimal (worst) hyperparameter interaction depending on their influence on the mAP.

Table 5.7 The best and worst hyperparameters' interactions on the mAP

Factor 2	Best Interaction
Convolution	832x832 with Dilated Conv (Yes)
Box Loss	with

Factor 2

Best Interaction

832x832 with New Anchor Dimensions

w
i
t
h

Factor 2

Best Interaction

Data Augmentation

Box Loss

Dilated Conv. (Yes) with DIOU

Factor 2

Best Interaction

Dilated Conv (Yes) with New Anchor Dimensions

Dilated Conv (Yes) with ResNet-152

Factor 2

Best Interaction

Data Augmentation

Dilated Conv (Yes) with Mosaic Augmentation

DIoU with New Anchor Dimensions

Factor 2

Best Interaction

with

Data Augmentation

w
i
t
h

Factor 2

Best Interaction

New Anchor Dimensions with ResNet-152

Data Augmentation

New Anchor Dimensions with Mosaic Augmentation

Factor 2

Best Interaction

Data Augmentation

w
i
t
h

5.4.2 ANOVA analysis for the testing process

We followed a similar ANOVA process for the testing process. Table 5.8 below illustrates the ANOVA results of the testing runs.

Table 5.8 Analysis of Variance during testing process

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	96	3.50041	0.03646	26.42	0.000
Blocks	1	0.00004	0.00004	0.03	0.857
Linear	7	2.98687	0.42670	309.16	0.000
Image Resolution	2	0.91496	0.45748	331.46	0.000
Dilated Convolution	1	0.03135	0.03135	22.71	0.000
Box Loss	1	0.07768	0.07768	56.28	0.000
Anchor Dimensions	1	0.04532	0.04532	32.84	0.000
Backbone Network	1	1.69804	1.69804	1230.30	0.000
Data Augmentation	1	0.21952	0.21952	159.05	0.000
2-Way Interactions	20	0.33099	0.01655	11.99	0.000
Image Resolution*Dilated Convolution	2	0.00193	0.00096	0.70	0.500
Image Resolution*Box Loss	2	0.00691	0.00346	2.50	0.087
Image Resolution*Anchor Dimensions	2	0.03032	0.01516	10.98	0.000
Image Resolution*Backbone Network	2	0.02093	0.01047	7.58	0.001
Image Resolution*Data Augmentation	2	0.03326	0.01663	12.05	0.000
Dilated Convolution*Box Loss	1	0.00003	0.00003	0.02	0.876
Dilated Convolution*Anchor Dimensions	1	0.00168	0.00168	1.22	0.273
Dilated Convolution*Backbone Network	1	0.05094	0.05094	36.91	0.000
Dilated Convolution*Data Augmentation	1	0.00618	0.00618	4.48	0.037
Box Loss*Anchor Dimensions	1	0.01187	0.01187	8.60	0.004
Box Loss*Backbone Network	1	0.00001	0.00001	0.01	0.942
Box Loss*Data Augmentation	1	0.00902	0.00902	6.54	0.012
Anchor Dimensions*Backbone Network	1	0.00074	0.00074	0.54	0.466
Anchor Dimensions*Data Augmentation	1	0.00105	0.00105	0.76	0.384
Backbone Network*Data Augmentation	1	0.15612	0.15612	113.12	0.000
3-Way Interactions	30	0.10889	0.00363	2.63	0.000
Image Resolution*Dilated Convolution*Box Loss	2	0.00388	0.00194	1.40	0.251
Image Resolution*Dilated Convolution*Anchor Dimensions	2	0.00121	0.00060	0.44	0.647

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Image Resolution*Dilated Convolution*Backbone Network	2	0.01060	0.00530	3.84	0.025
Image Resolution*Dilated Convolution*Data Augmentation	2	0.00264	0.00132	0.96	0.388
Image Resolution*Box Loss*Anchor Dimensions	2	0.00202	0.00101	0.73	0.484
Image Resolution*Box Loss*Backbone Network	2	0.00112	0.00056	0.40	0.668
Image Resolution*Box Loss*Data Augmentation	2	0.00117	0.00059	0.42	0.655
Image Resolution*Anchor Dimensions*Backbone Network	2	0.00439	0.00220	1.59	0.209
Image Resolution*Anchor Dimensions*Data Augmentation	2	0.00274	0.00137	0.99	0.374
Image Resolution*Backbone Network*Data Augmentation	2	0.02277	0.01139	8.25	0.000
Dilated Convolution*Box Loss*Anchor Dimensions	1	0.00046	0.00046	0.33	0.567
Dilated Convolution*Box Loss*Backbone Network	1	0.00823	0.00823	5.96	0.016
Dilated Convolution*Box Loss*Data Augmentation	1	0.00059	0.00059	0.43	0.515
Dilated Convolution*Anchor Dimensions*Backbone Network	1	0.01571	0.01571	11.38	0.001
Dilated Convolution*Anchor Dimensions*Data Augmentation	1	0.00023	0.00023	0.16	0.686
Dilated Convolution*Backbone Network*Data Augmentation	1	0.00554	0.00554	4.02	0.048
Box Loss*Anchor Dimensions*Backbone Network	1	0.00815	0.00815	5.90	0.017
Box Loss*Anchor Dimensions*Data Augmentation	1	0.00777	0.00777	5.63	0.020
Box Loss*Backbone Network*Data Augmentation	1	0.00826	0.00826	5.99	0.016
Anchor Dimensions*Backbone Network*Data Augmentation	1	0.00142	0.00142	1.03	0.313
4-Way Interactions	25	0.05280	0.00211	1.53	0.074
Image Resolution*Dilated Convolution*Box Loss*Anchor Dimensions	2	0.00127	0.00063	0.46	0.634

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Image Resolution*Dilated Convolution*Box Loss*Backbone Network	2	0.00663	0.00331	2.40	0.096
Image Resolution*Dilated Convolution*Box Loss*Data Augmentation	2	0.00039	0.00020	0.14	0.867
Image Resolution*Dilated Convolution*Anchor Dimensions*Backbone Network	2	0.01226	0.00613	4.44	0.014
Image Resolution*Dilated Convolution*Anchor Dimensions*Data Augmentation	2	0.00090	0.00045	0.33	0.723
Image Resolution*Dilated Convolution*Backbone Network*Data Augmentation	2	0.00357	0.00178	1.29	0.279
Image Resolution*Box Loss*Anchor Dimensions*Backbone Network	2	0.00510	0.00255	1.85	0.163
Image Resolution*Box Loss*Anchor Dimensions*Data Augmentation	2	0.00059	0.00029	0.21	0.808
Image Resolution*Box Loss*Backbone Network*Data Augmentation	2	0.00078	0.00039	0.28	0.756
Image Resolution*Anchor Dimensions*Backbone Network*Data Augmentation	2	0.00260	0.00130	0.94	0.393
Dilated Convolution*Box Loss*Anchor Dimensions*Backbone Network	1	0.00951	0.00951	6.89	0.010
Dilated Convolution*Box Loss*Anchor Dimensions*Data Augmentation	1	0.00083	0.00083	0.60	0.440
Dilated Convolution*Box Loss*Backbone Network*Data Augmentation	1	0.00044	0.00044	0.32	0.575
Dilated Convolution*Anchor Dimensions*Backbone Network*Data Augmentation	1	0.00002	0.00002	0.01	0.909
Box Loss*Anchor Dimensions*Backbone Network*Data Augmentation	1	0.00793	0.00793	5.74	0.019
5-Way Interactions	11	0.01671	0.00152	1.10	0.370
Image Resolution*Dilated Convolution*Box Loss*Anchor Dimensions*Backbone Network	2	0.01044	0.00522	3.78	0.026
Image Resolution*Dilated Convolution*Box Loss*Anchor Dimensions*Data Augmentation	2	0.00361	0.00181	1.31	0.275
Image Resolution*Dilated Convolution*Box Loss*Backbone Network*Data Augmentation	2	0.00012	0.00006	0.04	0.958

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Image Resolution*Dilated Convolution*Anchor Dimensions*Backbone Network*Data Augmentation	2	0.00073	0.00037	0.27	0.767
Image Resolution*Box Loss*Anchor Dimensions*Backbone Network*Data Augmentation	2	0.00074	0.00037	0.27	0.765
Dilated Convolution*Box Loss*Anchor Dimensions*Backbone Network*Data Augmentation	1	0.00106	0.00106	0.77	0.383
6-Way Interactions	2	0.00411	0.00205	1.49	0.231
Image Resolution*Dilated Convolution*Box Loss*Anchor Dimensions*Backbone Network*Data Augmentation	2	0.00411	0.00205	1.49	0.231
Error	95	0.13112	0.00138		
Total	191	3.63153			

Pareto Chart

Figure 5.12 illustrates the Pareto chart of standardized effects for the testing process of our study. It demonstrates the influence and interactions of the different selected factors, on the mAP like Figure 5.9. The factors labeled from A to F, are the hyperparameters we selected for our study. More specifically:

- A: Image Resolution
- B: Dilated Convolution
- C: Box Loss
- D: Anchor Dimensions
- E: Backbone Network
- F: Data Augmentation

In Figure 5.12, the red-dashed line indicates the standardized effect value of 1.99 when the significance threshold is $\alpha = 0.05$.

Based on the Pareto Chart in Figure 5.12, the factors and the interaction of factors in the testing process of our study are the following ones (starting from the most significant to the least significant):

- Backbone Network (E)
- Data Augmentation (F)
- The interaction Backbone Network (E) and Data Augmentation (F)
- Image Resolution (A)
- Box Loss (C)

- The interaction of Dilated Convolution (B) and Backbone Network (E)
- Anchor Dimensions (D)
- Dilated Convolution (B)
- The interaction of Image Resolution (A) and Data Augmentation (F)
- The interaction of Image Resolution (A) and Anchor Dimensions (D)
- The interaction of Image Resolution (A), Backbone Network (E) and Data Augmentation (F)
- The interaction of Image Resolution (A) and Backbone Network (E)
- The interaction of Dilated Convolution (B), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Box Loss (C) and Anchor Dimensions (D)
- The interaction of Dilated Convolution (B), Box Loss (C), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Box Loss (C) and Data Augmentation (F)
- The interaction of Image Resolution (A), Dilated Convolution (B), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Box Loss (C), Backbone Network (E) and Data Augmentation (F)
- The interaction of Dilated Convolution (B), Box Loss (C) and Backbone Network (E)
- The interaction of Box Loss (C), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Box Loss (C), Anchor Dimensions (D), Backbone Network (E) and Data Augmentation (F)
- The interaction of Box Loss (C), Anchor Dimensions (D) and Data Augmentation (F)
- The interaction of Image Resolution (A), Dilated Convolution (B) and Backbone Network (E)
- The interaction of Image Resolution (A), Dilated Convolution (B), Box Loss (C), Anchor Dimensions (D) and Backbone Network (E)
- The interaction of Dilated Convolution (B) and Data Augmentation (F)
- The interaction of Dilated Convolution (B), Backbone Network (E) and Data Augmentation (F)

The remaining hyperparameters or their interactions do not have a significant effect on mAP, since their values are less than 1.99.

In the testing process the most significant hyperparameter is the backbone network (E), followed by data augmentation (F) and image resolution (A). Of course, all factors and factor interactions exceeding the red line (threshold 1.99) are also noted as statistically significant. Interactions such as AE, AF, BE, and AB have significant effects on mAP.

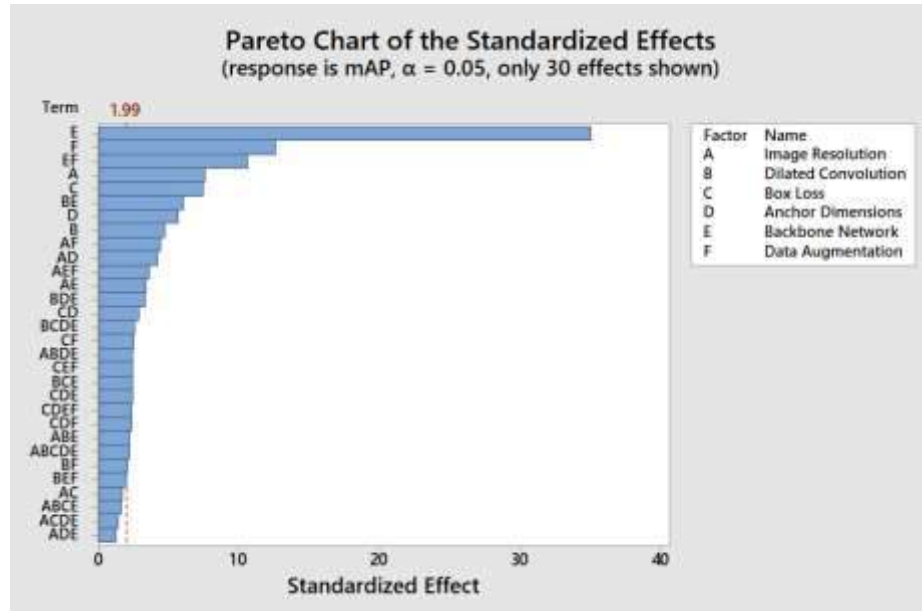


Figure 5.12 Pareto chart of the standardized effects during testing process

Based on Pareto Chart of the training process (Figure 5.9) and the Pareto Chart of the testing process (Figure 5.12), we can conclude that:

1. *Important hyperparameters:*

- Training: The Backbone Network (E) has the largest standardized effect, followed by Data Augmentation (F) and Anchor Dimensions (D). These three hyperparameters are the dominant ones in the training process.
- Testing: Similarly, Backbone Network (E) remains the most important hyperparameter during testing, followed by Data Augmentation (F) and Image Resolution (A).

2. *Important interactions of hyperparameters:*

- Training: Significant interactions include EF, CD, AF, and CF. These interactions suggest that interactions of backbone network with data augmentation and anchor dimensions are important during training.
- Testing: Interactions such as AE, AF, BE, and AB seem to be more important during testing, indicating that factors like image resolution, when combined with data augmentation or the backbone network, have a greater influence on mAP.

Therefore, we can conclude that backbone network (E) and data augmentation (F) consistently remain the most significant hyperparameters for improving mAP in both training and testing processes, indicating their importance in model performance across both phases.

Main effects plot for mAP

Figure 5.13 shows the main effects plot for the testing process in our study (Kim et al., 2007), which measures the impact on mAP for the factors A, B, C, D, E and F. By analyzing these six factors in the main effect plot in Minitab, we can conclude to the following observations below:

- Increasing image resolution from 352x352 to 413x416 improves mAP by $24.6\% - 20.4\% = 4.2\%$. By further increasing the image resolution from 416x416 to 832x832 improves mAP by $36.7\% - 24.6\% = 12.1\%$
- Adding dilated convolution in our model reduces mAP by $28.5\% - 25.9\% = 2.6\%$
- Changing the box loss function from IoU to DIoU improves mAP by $29.2\% - 25.2\% = 4\%$
- Changing anchor box dimensions from the default model values to a new set of values reduces mAP by $28.8\% - 25.7\% = 3.1\%$.
- Changing the backbone network from Darknet-53 (default) to ResNet-152 (new) reduces mAP by $36.6\% - 17.8\% = 18.8\%$.
- Adding data augmentation in our model improves mAP by $30.6\% - 23.8\% = 6.8\%$

The addition of dilated convolution in our model, the selection of new anchor dimensions and the alternation of the backbone network from Darknet-53 to ResNet-152 have negative effect on mAP as they reduce its value. As we can see from the plot below (Figure 5.13) the choice of ResNet-152 over Darknet-53 has the most negative impact on mAP, reducing its value approximately by half (36.6% to 17.8%).

On the other hand, the increase on image resolution, the alternation of box loss function from IoU to DIoU and the addition of data augmentation improve the value of mAP, with the image resolution having the greatest impact, similar to the training process.

Therefore, we conclude that the best options for the factors/hyperparameters we selected, which positively affect the mAP value, are the same with the training process:

- 832x832 for image resolution
- Absence of dilated convolution
- DIoU for box loss function
- Default values for the anchor dimensions
- Darknet-53 as the backbone network
- Addition of the mosaic data augmentation technique

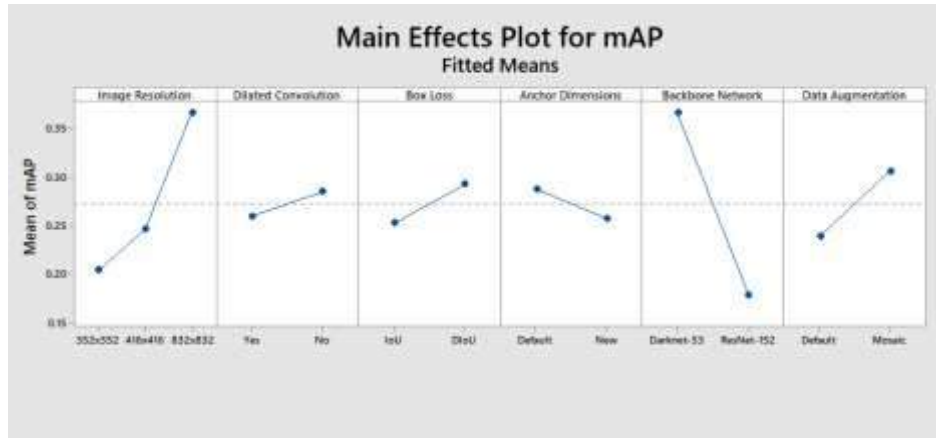


Figure 5.13 Main Effects Plot for mAP during testing process

Therefore, from the above results, we can conclude that Darknet-53 as a backbone offers a better performance when we have a higher image resolution (832x832), the default anchor dimension, the YOLOv3 model suggests, and the dilated convolution before the YOLO heads, is not added. Nevertheless, it seems that the addition of data augmentation is important, as the default YOLOv3 model does not use data augmentation techniques. Additionally, our performance is better when the box loss function is Diou instead of the IoU that is the default box loss function of YOLOv3.

5.4.3 Similarities and differences in the analysis results of validation vs. testing

Table 5.9 compares the effects of the main factors on mAP resulting from the ANOVA of validation vs. testing.

Table 5.9 Comparison of the effects of the main factors on mAP between validation and testing. The values indicate the difference between High and Low

Factor/Interaction	Validation effect on mAP in % (High – Low)	Testing effect on mAP in % (High – Low)
Image resolution (A)	4.0 + 12.2 = 16.2	4.2 + 12.1 = 16.3
Dilated Convolution (B)	3	2.6
Box Loss Function (C)	5.9	4
Anchor Box Dimensions (D)	5	3.1
Backbone Network (E)	22.5	18.8
Data Augmentation (F)	8.4	6.8

More specifically, we can conclude that the effect of Image Resolution (A) on mAP resulting from either the validation or testing ANOVA is consistent. Similar conclusions hold for almost all other Factors. In general, though the effects resulting from the ANOVA of the testing results have slightly lower values as those resulting from the ANOVA of the Testing results. This may be attributed to the lower mAP values obtained during testing.

Overall, the analysis highlights that hyperparameters, such as image resolution and the backbone network, have consistently very significant effects on mAP across both validation and testing phases. The effects of the rest of the factors are still statistically significant but lower in value.

5.4.4 Evaluation Metrics for YOLOv3: Performance

As discussed in Chapter 3, key evaluation metrics such as Precision, Recall, F1-Score, Average Precision (AP), and mean Average Precision (mAP) are used to assess model performance. As a reminder, Precision measures how accurately the model identifies objects, while Recall indicates its ability to detect all relevant objects. Thus, Precision is the ratio of True Positives to True Positives + False Positives, while Recall is the ratio of True Positives to total objects in the image (True Positives + False Negatives). A True Positive occurs when the model correctly detects an object with appropriate Intersection over Union (IoU) and the right classification, while a False Positive occurs when the model misclassifies, that is, it detects a non-existent object or predicts multiple boxes for the same object. A False Negative occurs when the model may not detect an object that is present in the image. The F1-score provides a balance between Precision and Recall, and AP evaluates the Precision across different Recall values for a single class, whereas mAP calculates the average AP across all object classes. A higher mAP suggests improved detection performance across multiple categories. Consequently, these metrics are important in evaluating the model’s performance.

The 90th experiment results (see Table 5.10) give a detailed evaluation of the model’s performance using the above metrics. Based on these results, it is observed that while the model detects cars accurately, it has difficulty identifying bikes and people, which leads to a higher number of false positives.

Table 5.10 Evaluation metrics of YOLOv3 90th experiment

Table 5.10 Evaluation metrics of YOLOv3 90 th experiment									
Overall model performance									

Specifically,

Person - Class ID: 0

- TP (True Positives): 19,033 correctly detected people with sufficient IoU and correct classification
- FP (False Positives): 9,750 incorrect detections due to misclassification, ghost detections, or overlapping bounding boxes
- FN (False Negatives): 23,263 undetected objects, although they existed in the image

Car - Class ID: 1

- TP: 33,299 cars correctly detected with proper IoU and classification
- FP: 8,054 incorrect detections, likely including misclassified objects or duplicate boxes

- FN (False Negatives): 6,820 actual cars were not identified by the model despite being in the image

Long Vehicle - Class ID: 2

- TP: 6,702 long vehicles correctly identified
- FP: 3,180 false detections, likely due to incorrect object classification or improper bounding box placement.
- FN (False Negatives): 3,936 long vehicles were present in the image but remained undetected by the model

Bike - Class ID: 3

- TP: 9,823 bikes correctly detected.
- FP: 5,757 incorrect detections, possibly caused by confusion with other objects or duplicate detections.
- FN (False Negatives): 17,463 bicycles were missing by the model, leading to missing detections.

Total Performance

- TP: 68,857 objects correctly detected.
- FP: 26,741 incorrect detections due to misclassification, poor bounding box placement, or multiple detections of the same object.
- FN (False Negatives): 51,482 objects were not detected, reducing the model's recall performance

Therefore, the model demonstrates strong performance in car detection, achieving the highest AP (85.16%), with high precision (0.81) and recall (0.83), making it the most reliable class. However, bike detection is the weakest, with low AP (41.88%) and recall (0.36), indicating that the model struggles to correctly identify bikes, often missing actual bikes and producing many false positives. Person detection is also challenging, having a moderate AP (49.11%), low recall (0.45), high false positives (9,750), and high false negatives (23,263) indicating frequent misclassification of objects as people. The high false positive rate (FP = 26,741) indicates that the model frequently detects objects incorrectly.

Based on our experimental results, we can conclude from the last row of Table 5.10:

1. Precision (0.72) showcases a good accuracy, but false positives exist. The model correctly classifies 72% of detected objects, meaning that 28% of detections are false positives.
2. Recall (0.57) indicates many missed detections. A recall score of 0.57 suggests that while the model detects many objects, a large portion remains undetected, missing 43% of actual objects. This imbalance between precision and recall indicates that the model avoids making too many incorrect detections but still fails to detect many actual objects.

3. F1-Score (0.64) indicates a moderate balance between precision and recall. A higher F1-score would indicate a more effective relationship between detecting all objects and reducing misclassifications.
4. mean Average Precision (60.99%) could indicate good but not optimal performance. This score of 60.99% suggests that across all object classes, the model is reliable but not highly accurate, especially at detecting objects across different recall levels. The mAP is largely influenced by strong performance in car detection (AP = 85.16%) and weaker performance in person and bike detection.

Previous studies report YOLOv3 mAP values ranging from 39.7% to 40.3% on similar UAV datasets (Pebrianto et al., 2023). Another study based on UAV imagery, achieved a test mAP of 31.4% on the VisDrone dataset using YOLOv3 (Zhang et al., 2023). Our model achieved a mAP of 60.99% in the validation process and a mAP of 52.37% (see Table 5.3) in the test process. Consequently, our model outperforms these benchmarks, demonstrating improved robustness in detecting various object classes. However, it is important to note that different UAV datasets were used in our research and in the studies mentioned.

It should be mentioned that mAP is the most significant evaluation metric. The mAP value is computed at a specific Intersection over Union (IoU) threshold, commonly set at 0.5 (mAP@0.5). A higher IoU threshold (e.g., 0.75) requires stricter overlap between predicted and ground-truth bounding boxes, often reducing mAP, while a lower threshold (e.g., 0.25) allows less strict detections, potentially increasing mAP. This means a model with high mAP (0.75) is more precise in localization. A helpful way to understand mAP is to imagine it as a measure of confidence in both detecting and correctly classifying objects. For instance, if the model detects a person but mistakenly classifies it as a bicycle, this impacts AP and ultimately reduces mAP. A model with a high mAP not only finds most objects but also classifies them correctly with high confidence.

In simple terms, mAP does not indicate the percentage of objects detected in an image. A common mistake is that if mAP = 60%, the model detects 60% of the objects in an image. However, mAP is a measure of both detection accuracy and classification correctness across multiple recall thresholds. For example, if a model predicts 80 bounding boxes but 40 of them have poor localization ($IoU \leq 0.5$) or incorrect class labels ($Class\ threshold \leq 0.25$), the precision and recall values will be affected, leading to a lower mAP score. Thus, a model with mAP = 60% does not mean it identifies 60% of objects, but rather that it achieves an average precision of 60% over different recall values across multiple object categories. In practice, mAP serves as a holistic performance metric that evaluates both how many objects are detected and how correctly they are localized and classified rather than just the proportion of objects found in an image.

Chapter 6 Conclusions

This thesis focuses on optimizing training of the YOLOv3 model for object detection using UAV-captured imagery. It showcases the importance of hyperparameter selection in training effectiveness. Specifically, through extensive analysis, we identified important hyperparameters that influence the trained model's performance. By adjusting these hyperparameters we fine-tuned training of the model to achieve higher precision in detecting objects.

The study utilized annotated UAV datasets that were preprocessed to align with YOLOv3's requirements. The selected datasets were the UA Vehicle Detection Dataset, Stanford Dataset and VisDrone2019DET dataset. The consolidated dataset consists of 16,303 images with 602,240 annotations and it was split into training (80%), validation (10%), and testing (10%) subsets.

Training optimization was approached by dividing hyperparameters into two categories. The first one included hyperparameters that were set according to the characteristics of the training subset and were kept invariant throughout the analysis. These included max batches, number of classes, filters, and steps. The second category contained the hyperparameters we selected to adjust; i.e., image resolution, backbone network, anchor box dimensions, dilated convolution, box loss and data augmentation techniques.

A Full-Factorial experimental design was employed to generate 96 ($2^5 \times 3$) distinct combinations of these key hyperparameters. The training process was executed twice for each combination of the selected hyperparameters, resulting in a total of 192 trained models. During training, validation was performed every 100 iterations. Finally, after training, we conducted the testing process to evaluate model performance.

Each of the 192 experiments produced outputs consisting of the highest mAP achieved during validation and testing. The results of these experiments were analyzed using ANOVA, which revealed that all hyperparameters significantly influence model's performance. Among them, the most impactful hyperparameters on performance are: the backbone network, data augmentation and image resolution. Additionally, two significant two-way interactions were observed: a) between the backbone network and data augmentation, and b) between the backbone network and dilated convolution.

The best-performing model achieved mAP values of 60.99% during training/validation and 52.51% during testing process. The model that achieved this performance corresponds to the following hyperparameter combination:

- Image Resolution: 832x832
- Dilated Convolution: No
- Box Loss: DIoU
- Anchor Dimensions: Default
- Backbone: Darknet-53
- Data Augmentation: Mosaic

On the other hand, the performance of the lowest performing models was very low, indicating that the hyperparameter selection and tuning plays an important role and could lead to significant improvements in YOLOv3's detection performance.

The thesis contributed in revealing:

- The important role of hyperparameter selection and tuning in optimizing YOLOv3's training performance
- The significant and quantifiable impact of hyperparameters and their interactions on the precision of object detection.

Future research investigations could include:

- Exploring hyperparameters of newer YOLO versions to analyze how they could affect YOLOv3 performance
- Integrating another backbone network to validate that the default one (Darknet-53) offers the best performance
- Incorporating other types of data augmentation apart from "mosaic" that was selected in our study
- Testing the performance of the model in a greater variety of datasets
- Creating balance in class representation to lower the mAP differences between training/validation and testing.

References

- Aksu, G., Güzeller, C.O., Eser, M.T., 2019. The Effect of the Normalization Method Used in Different Sample Sizes on the Success of Artificial Neural Network Model. *Int. J. Assess. Tools Educ.* 6, 170–192. <https://doi.org/10.21449/ijate.479404>
- Alexey, 2024. AlexeyAB/darknet.
- Alexey, 2020. Mosaic Augmentation Paper? · Issue #8 · WongKinYiu/CrossStagePartialNetworks [WWW Document]. GitHub. URL <https://github.com/WongKinYiu/CrossStagePartialNetworks/issues/8> (accessed 9.1.24).
- 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).
- Atik, M.E., Duran, Z., Özgünlük, R., 2022. Comparison of YOLO Versions for Object Detection from Aerial Images. *IJEGEO* 9, 87–93. <https://doi.org/10.30897/ijegeo.1010741>
- Bi, Y., Xue, B., Mesejo, P., Cagnoni, S., Zhang, M., 2023. A Survey on Evolutionary Computation for Computer Vision and Image Analysis: Past, Present, and Future Trends. *IEEE Transactions on Evolutionary Computation* 27, 5–25. <https://doi.org/10.1109/TEVC.2022.3220747>
- Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y.M., 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. <https://doi.org/10.48550/arXiv.2004.10934>
- Bodla, N., Singh, B., Chellappa, R., Davis, L.S., 2017. Soft-NMS -- Improving Object Detection With One Line of Code. Presented at the Proceedings of the IEEE International Conference on Computer Vision, pp. 5561–5569.
- Brock, A., Lim, T., Ritchie, J.M., Weston, N., 2017. FreezeOut: Accelerate Training by Progressively Freezing Layers. <https://doi.org/10.48550/arXiv.1706.04983>
- Buczowski, M., Stasiński, R., 2019. Convolutional Neural Network-Based Image Distortion Classification, in: 2019 International Conference on Systems, Signals and Image Processing (IWSSIP). Presented at the 2019 International Conference on Systems, Signals and Image Processing (IWSSIP), pp. 275–279. <https://doi.org/10.1109/IWSSIP.2019.8787212>
- Cai, Z., Fan, Q., Feris, R.S., Vasconcelos, N., 2016. A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection, in: Leibe, B., Matas, J., Sebe, N., Welling, M. (Eds.), *Computer Vision – ECCV 2016*. Springer International Publishing, Cham, pp. 354–370. https://doi.org/10.1007/978-3-319-46493-0_22
- Chetlur, S., Woolley, C., Vanderersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E., 2014. cuDNN: Efficient Primitives for Deep Learning. <https://doi.org/10.48550/arXiv.1410.0759>
- Crasto, N., 2024. Class Imbalance in Object Detection: An Experimental Diagnosis and Study of Mitigation Strategies. <https://doi.org/10.48550/arXiv.2403.07113>

- Cruz Martinez, J., 2021. Real-time Object Detection in Video (with intro to Yolo v3) [WWW Document]. URL <https://infotech.report/guest-articles/real-time-object-detection-in-video-with-intro-to-yolo-v3> (accessed 8.25.24).
- Culjak, I., Abram, D., Pribanic, T., Dzapo, H., Cifrek, M., 2012. A brief introduction to OpenCV, in: 2012 Proceedings of the 35th International Convention MIPRO. Presented at the 2012 Proceedings of the 35th International Convention MIPRO, pp. 1725–1730.
- Díaz-Cel, J., Arce-Lopera, C., Mena, J.C., Quintero, L., 2019. The Effect of Color Channel Representations on the Transferability of Convolutional Neural Networks | SpringerLink [WWW Document]. URL https://link.springer.com/chapter/10.1007/978-3-030-17795-9_3 (accessed 9.23.24).
- Diwan, T., Anirudh, G., Tembhurne, J.V., 2023. Object detection using YOLO: challenges, architectural successors, datasets and applications. *Multimed Tools Appl* 82, 9243–9275. <https://doi.org/10.1007/s11042-022-13644-y>
- Dubey, A.K., Jain, V., 2019. Comparative Study of Convolution Neural Network’s Relu and Leaky-Relu Activation Functions, in: Mishra, S., Sood, Y.R., Tomar, A. (Eds.), *Applications of Computing, Automation and Wireless Systems in Electrical Engineering*. Springer, Singapore, pp. 873–880. https://doi.org/10.1007/978-981-13-6772-4_76
- 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).
- Gashi, D., Pereira, M., Vterkovska, V., 2017. Multi-Scale Context Aggregation by Dilated Convolutions *Machine Learning - Project*.
- Gilbert, T., 2020. YOLO Object Detection with OpenCV [WWW Document]. Gilbert Tanner. URL <https://gilberttanner.com/blog/yolo-object-detection-with-opencv/> (accessed 6.28.24).
- Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. Presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587.
- Hakala, T., Honkavaara, E., Saari, H., Mäkynen, J., Kaivosoja, J., Pesonen, L., Pölonen, I., 2013. SPECTRAL IMAGING FROM UAVS UNDER VARYING ILLUMINATION CONDITIONS. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XL-1-W2*, 189–194. <https://doi.org/10.5194/isprsarchives-XL-1-W2-189-2013>
- He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep Residual Learning for Image Recognition. Presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778.
- He, K., Zhang, X., Ren, S., Sun, J., 2015. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. https://doi.org/10.1007/978-3-319-10578-9_23
- Heffels, M., Vanschoren, J., 2020. Aerial Imagery Pixel-level Segmentation. <https://doi.org/10.48550/arXiv.2012.02024>
- 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. Presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4507–4515.
- Huang, C.-Y., Lin, I.-C., Liu, Y.-L., 2022. Applying Deep Learning to Construct a Defect Detection System for Ceramic Substrates. *Applied Sciences* 12, 2269. <https://doi.org/10.3390/app12052269>
- Igiri, C., Uzoma, A., Silas, A., 2021. Effect of Learning Rate on Artificial Neural Network in Machine Learning. *International Journal of Engineering Research* 4.
- iguazio, 2022. What is Recall [WWW Document]. Iguazio. URL <https://www.iguazio.com/glossary/recall/> (accessed 2.21.24).
- Ioffe, S., Szegedy, C., 2015a. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. <https://doi.org/10.48550/arXiv.1502.03167>
- Ioffe, S., Szegedy, C., 2015b. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, in: Proceedings of the 32nd International Conference on Machine Learning. Presented at the International Conference on Machine Learning, PMLR, pp. 448–456.
- Jiang, P., Ergu, D., Liu, F., Cai, Y., Ma, B., 2022. A Review of Yolo Algorithm Developments. *Procedia Computer Science, The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19* 199, 1066–1073. <https://doi.org/10.1016/j.procs.2022.01.135>
- Kamal, A., 2021. 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 2.2.24).
- Khan, S., 2023. IMAGE ANNOTATION USING DEEP LEARNING. Medium. URL <https://medium.com/@salmanskhan/image-annotation-using-deep-learning-115158fb4931> (accessed 9.15.24).
- Kirk, D., 2007. NVIDIA cuda software and gpu parallel computing architecture, in: Proceedings of the 6th International Symposium on Memory Management. Presented at the ISMM07: International Symposium on Memory Management, ACM, Montreal Quebec Canada, pp. 103–104. <https://doi.org/10.1145/1296907.1296909>
- Li, Y., Cheng, R., Zhang, C., Chen, M., Liang, H., Wang, Z., 2023. Dynamic Mosaic algorithm for data augmentation. *MBE* 20, 7193–7216. <https://doi.org/10.3934/mbe.2023311>
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft COCO: Common Objects in Context, in: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (Eds.), *Computer Vision – ECCV 2014*. Springer International Publishing, Cham, pp. 740–755. https://doi.org/10.1007/978-3-319-10602-1_48
- Liu, H.-T.D., Kim, V.G., Chaudhuri, S., Aigerman, N., Jacobson, A., 2020. Neural Subdivision. <https://doi.org/10.48550/arXiv.2005.01819>
- Liu, S., Qi, L., Qin, H., Shi, J., Jia, J., 2018. Path Aggregation Network for Instance Segmentation. Presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8759–8768.

- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C., 2016. SSD: Single Shot MultiBox Detector, in: Leibe, B., Matas, J., Sebe, N., Welling, M. (Eds.), *Computer Vision – ECCV 2016, Lecture Notes in Computer Science*. Springer International Publishing, Cham, pp. 21–37. https://doi.org/10.1007/978-3-319-46448-0_2
- Ma, H., Liu, Y., Ren, Y., Yu, J., 2020. Detection of Collapsed Buildings in Post-Earthquake Remote Sensing Images Based on the Improved YOLOv3. *Remote Sensing* 12, 44. <https://doi.org/10.3390/rs12010044>
- Montesinos López, O.A., Montesinos López, A., Crossa, J., 2022. Overfitting, Model Tuning, and Evaluation of Prediction Performance, in: Montesinos López, O.A., Montesinos López, A., Crossa, José (Eds.), *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. Springer International Publishing, Cham, pp. 109–139. https://doi.org/10.1007/978-3-030-89010-0_4
- Mu, S., Wang, C., Liu, M., Li, D., Zhu, M., Chen, X., Xie, X., Deng, Y., 2011. Evaluating the potential of graphics processors for high performance embedded computing, in: *2011 Design, Automation & Test in Europe*. Presented at the 2011 Design, Automation & Test in Europe, pp. 1–6. <https://doi.org/10.1109/DATE.2011.5763120>
- Mukhoti, J., Kulharia, V., Sanyal, A., Golodetz, S., Torr, P., Dokania, P., 2020. Calibrating Deep Neural Networks using Focal Loss, in: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., pp. 15288–15299.
- 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>
- 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>
- Öztürk, Ş., Özkaya, U., Akdemir, B., Seyfi, L., 2018. Convolution Kernel Size Effect on Convolutional Neural Network in Histopathological Image Processing Applications | IEEE Conference Publication | IEEE Xplore [WWW Document]. URL <https://ieeexplore.ieee.org/abstract/document/8742484> (accessed 9.23.24).
- Papageorgiou, C.P., Oren, M., Poggio, T., 1998. A general framework for object detection, in: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. Presented at the Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271), pp. 555–562. <https://doi.org/10.1109/ICCV.1998.710772>
- Pathak, A.R., Pandey, M., Rautaray, S., 2018. Application of Deep Learning for Object Detection. *Procedia Computer Science, International Conference on Computational Intelligence and Data Science* 132, 1706–1717. <https://doi.org/10.1016/j.procs.2018.05.144>
- Pebrianto, W., Mudjirahardjo, P., Pramono, S.H., Rahmadwati, Setyawan, R.A., 2023. YOLOv3 with Spatial Pyramid Pooling for Object Detection with Unmanned Aerial Vehicles. <https://doi.org/10.48550/arXiv.2305.12344>

- Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. Presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788.
- Redmon, J., Farhadi, A., 2018. YOLOv3: An Incremental Improvement. <https://doi.org/10.48550/arXiv.1804.02767>
- Redmon, J., Farhadi, A., 2017. YOLO9000: Better, Faster, Stronger, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Honolulu, HI, pp. 6517–6525. <https://doi.org/10.1109/CVPR.2017.690>
- Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, in: Advances in Neural Information Processing Systems. Curran Associates, Inc.
- Riad, R., Teboul, O., Grangier, D., Zeghidour, N., 2022. Learning strides in convolutional neural networks. <https://doi.org/10.48550/arXiv.2202.01653>
- Ribani, R., Marengoni, M., 2019. A Survey of Transfer Learning for Convolutional Neural Networks, in: 2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T). Presented at the 2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T), pp. 47–57. <https://doi.org/10.1109/SIBGRAPI-T.2019.00010>
- 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
- Santos, P.P., Carvalho, D.S., Sardinha, A., Melo, F.S., 2024. The impact of data distribution on Q-learning with function approximation. Mach Learn 113, 6141–6163. <https://doi.org/10.1007/s10994-024-06564-5>
- Sharma, Siddharth, Sharma, Simone, Athaiya, A., 2020. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. IJEAST 04, 310–316. <https://doi.org/10.33564/IJEAST.2020.v04i12.054>
- Shorten, C., Khoshgoftaar, T.M., 2019. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60. <https://doi.org/10.1186/s40537-019-0197-0>
- Shuai, Q., Wu, X., 2020. Object detection system based on SSD algorithm, in: 2020 International Conference on Culture-Oriented Science & Technology (ICCST). Presented at the 2020 International Conference on Culture-oriented Science & Technology (ICCST), IEEE, Beijing, China, pp. 141–144. <https://doi.org/10.1109/ICCST50977.2020.00033>
- Sujee, R., Sudharsun, L., Shanthosh, D., 2020. Fabric Defect Detection Using YOLOv2 and YOLO v3 Tiny | SpringerLink [WWW Document]. URL https://link.springer.com/chapter/10.1007/978-3-030-63467-4_15 (accessed 9.17.24).
- Teptaris, 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?fbclid=IwAR0aMRFETCX8zDIJcnF_ly5AzhPhvMwYZBlk14EaZIF5sNH_K_oIl2R8tCpl (accessed 2.8.24).

- Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T., Smeulders, A.W.M., 2013. Selective Search for Object Recognition. *Int J Comput Vis* 104, 154–171. <https://doi.org/10.1007/s11263-013-0620-5>
- Vakili, M., Ghamsari, M., Rezaei, M., 2020. Performance Analysis and Comparison of Machine and Deep Learning Algorithms for IoT Data Classification.
- van Dyk, D.A., Meng, X.-L., 2001. The Art of Data Augmentation. *Journal of Computational and Graphical Statistics* 10, 1–50. <https://doi.org/10.1198/10618600152418584>
- Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E., 2018. Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience* 2018, e7068349. <https://doi.org/10.1155/2018/7068349>
- Wang, 2024. [jwangjie/UAV-Vehicle-Detection-Dataset](https://github.com/jwangjie/UAV-Vehicle-Detection-Dataset).
- Wenkel, S., Alhazmi, K., Liiv, T., Alrshoud, S., Simon, M., 2021. Confidence Score: The Forgotten Dimension of Object Detection Performance Evaluation [WWW Document]. URL <https://www.mdpi.com/1424-8220/21/13/4350> (accessed 9.23.24).
- Wu, Y., He, K., 2018. Group Normalization. Presented at the Proceedings of the European Conference on Computer Vision (ECCV), pp. 3–19.
- Wu, Z., Shen, C., van den Hengel, A., 2019. Wider or Deeper: Revisiting the ResNet Model for Visual Recognition. *Pattern Recognition* 90, 119–133. <https://doi.org/10.1016/j.patcog.2019.01.006>
- Xiong, C., Zayed, T., Abdelkader, E.M., 2024. A novel YOLOv8-GAM-Wise-IoU model for automated detection of bridge surface cracks. *Construction and Building Materials* 414, 135025. <https://doi.org/10.1016/j.conbuildmat.2024.135025>
- Xu, H., Yao, L., Li, Z., Liang, X., Zhang, W., 2019. Auto-FPN: Automatic Network Architecture Adaptation for Object Detection Beyond Classification, in: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). Presented at the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), IEEE, Seoul, Korea (South), pp. 6648–6657. <https://doi.org/10.1109/ICCV.2019.00675>
- Yang, F., Choi, W., Lin, Y., 2016. Exploit All the Layers: Fast and Accurate CNN Object Detector with Scale Dependent Pooling and Cascaded Rejection Classifiers, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2129–2137. <https://doi.org/10.1109/CVPR.2016.234>
- Yang, K., Yau, J.H., Fei-Fei, L., Deng, J., Russakovsky, O., 2022. A Study of Face Obfuscation in ImageNet, in: Proceedings of the 39th International Conference on Machine Learning. Presented at the International Conference on Machine Learning, PMLR, pp. 25313–25330.
- Zhang, L., Xiong, N., Pan, X., Yue, X., Wu, P., Guo, C., 2023. Improved Object Detection Method Utilizing YOLOv7-Tiny for Unmanned Aerial Vehicle Photographic Imagery. *Algorithms* 16, 520. <https://doi.org/10.3390/a16110520>
- Zhang, W., Fu, C., Xie, H., Zhu, M., Tie, M., Chen, J., 2021. Global context aware RCNN for object detection. *Neural Computing and Applications* 33, 1–13. <https://doi.org/10.1007/s00521-021-05867-1>

Zhang, X., Zou, Y., Shi, W., 2017. Dilated convolution neural network with LeakyReLU for environmental sound classification, in: 2017 22nd International Conference on Digital Signal Processing (DSP). Presented at the 2017 22nd International Conference on Digital Signal Processing (DSP), pp. 1–5.

<https://doi.org/10.1109/ICDSP.2017.8096153>

Zhang, Y., Wallace, B., 2015. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification [WWW Document]. arXiv.org. URL

<https://arxiv.org/abs/1510.03820v4> (accessed 9.10.24).

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., Ren, D., 2020. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. Proceedings of the AAAI Conference on Artificial Intelligence 34, 12993–

13000. <https://doi.org/10.1609/aaai.v34i07.6999>

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