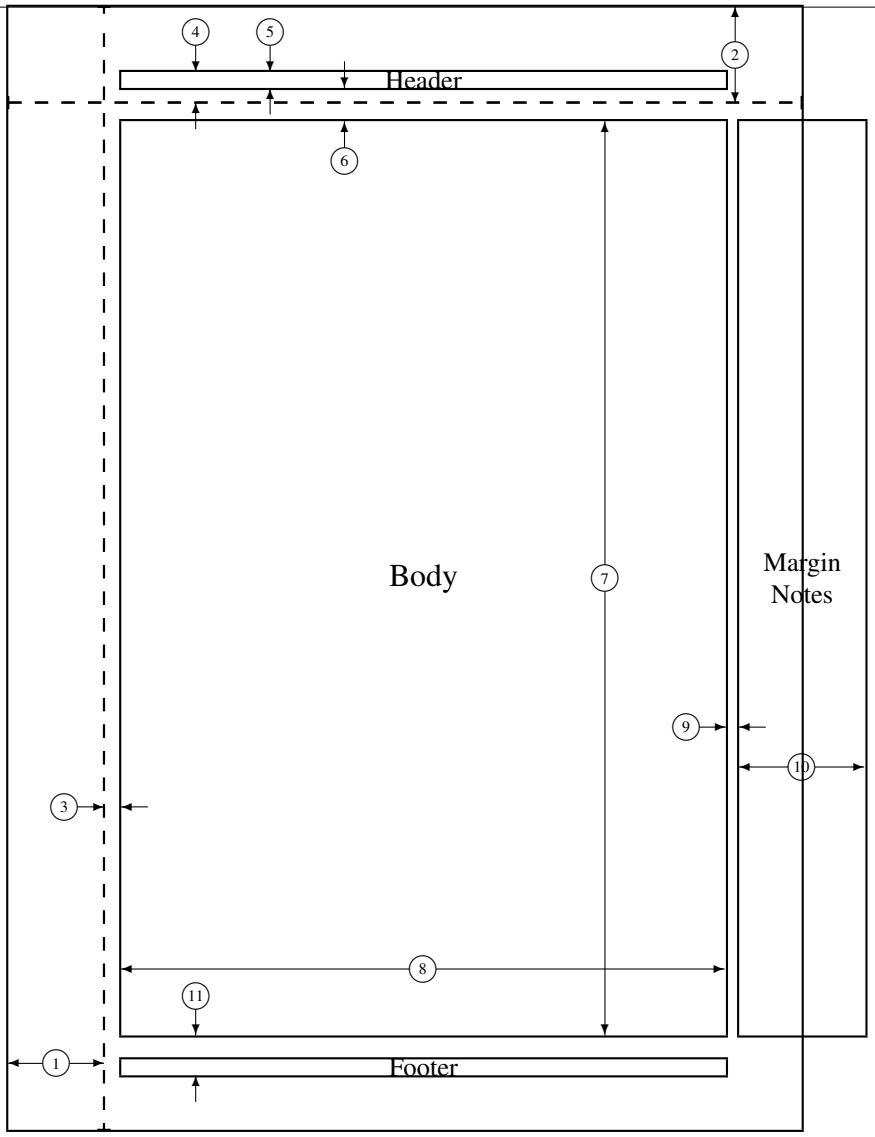
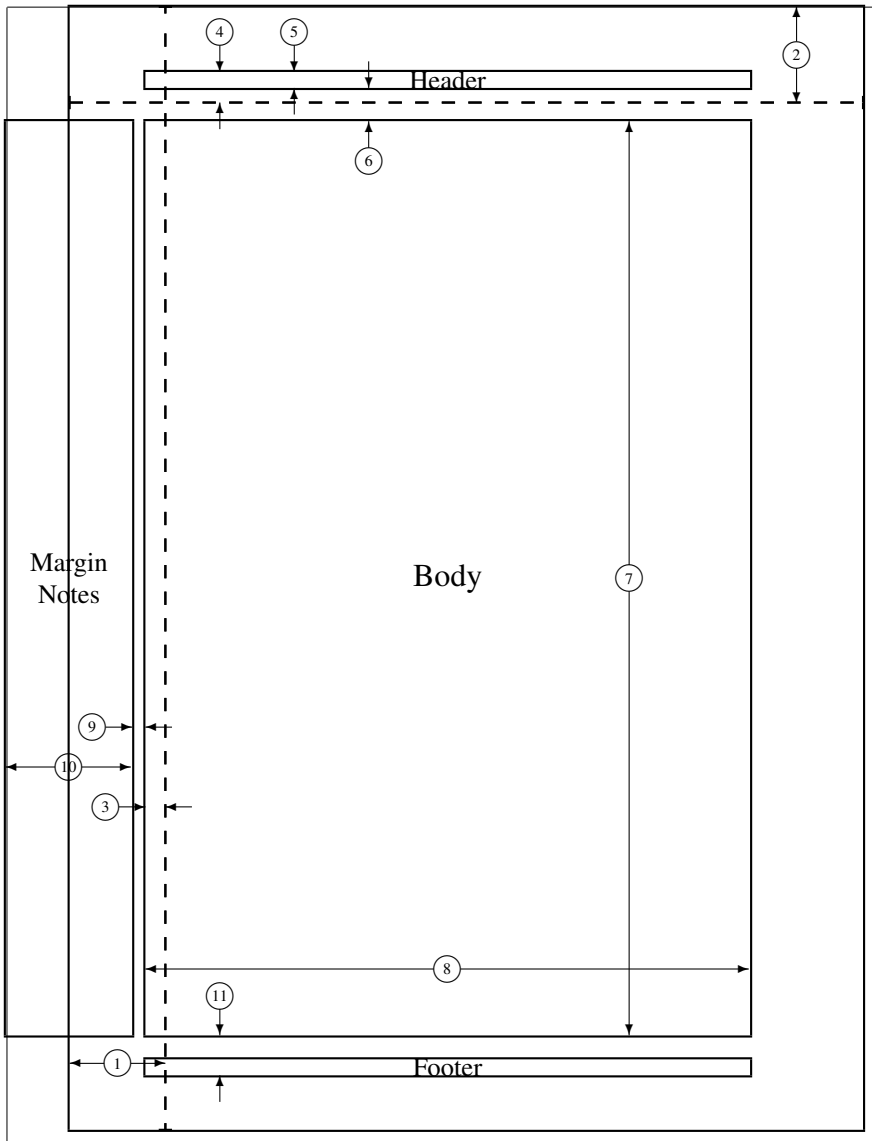


1	one inch + \hoffset	2	one inch + \voffset
3	\oddsidemargin = 13pt	4	\topmargin = -23pt
5	\headheight = 12pt	6	\headsep = 25pt
7	\textheight = 688pt	8	\textwidth = 455pt
9	\marginparsep = 10pt	10	\marginparwidth = 95pt
11	\footskip = 30pt		\marginparpush = 7pt (not shown)
	\hoffset = 0pt		\voffset = 0pt
	\paperwidth = 597pt		\paperheight = 845pt



1	one inch + \hoffset	2	one inch + \voffset
3	\evensidemargin = -15pt	4	\topmargin = -23pt
5	\headheight = 12pt	6	\headsep = 25pt
7	\textheight = 688pt	8	\textwidth = 455pt
9	\marginparsep = 10pt	10	\marginparwidth = 95pt
11	\footskip = 30pt		\marginparpush = 7pt (not shown)
	\hoffset = 0pt		\voffset = 0pt
	\paperwidth = 597pt		\paperheight = 845pt

TUGAS AKHIR - EC184801

OPTIMISASI PENDETEKSIAN OBJEK KECIL YOLOv7 UNTUK MENDETEKSI OBJEK-OBJEK *AIRBORNE*

Dion Andreas Solang
NRP 0721 19 4000 0039

Dosen Pembimbing

Reza Fuad Rachmadi, S.T., M.T., Ph.D

NIP 19850403201212 1 001

Dr. I Ketut Eddy Purnama, S.T., M.T.
NIP 19690730199512 1 001

Program Studi Strata 1 (S1) Teknik Komputer
Departemen Teknik Komputer
Fakultas Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2023

[This page intentionally left blank]

TUGAS AKHIR - EC184801

OPTIMISASI PENDETEKSIAN OBJEK KECIL YOLOv7 UNTUK MENDETEKSI OBJEK-OBJEK *AIRBORNE*

Dion Andreas Solang
NRP 0721 19 4000 0039

Dosen Pembimbing

Reza Fuad Rachmadi, S.T., M.T., Ph.D

NIP 19850403201212 1 001

Dr. I Ketut Eddy Purnama, S.T., M.T.
NIP 19690730199512 1 001

Program Studi Strata 1 (S1) Teknik Komputer
Departemen Teknik Komputer
Fakultas Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2023

[This page intentionally left blank]

FINAL PROJECT - EC184801

YOLOv7 SMALL OBJECT DETECTION OPTIMIZATION TO DETECT AIRBORNE OBJECTS

Dion Andreas Solang

NRP 0721 19 4000 0039

Advisor

Reza Fuad Rachmadi, S.T., M.T., Ph.D

NIP 19850403201212 1 001

Dr. I Ketut Eddy Purnama, S.T., M.T.

NIP 19690730199512 1 001

Undergraduate Study Program of Computer Engineering

Department of Computer Engineering

Faculty of Faculty of Intelligent Electrical and Informatics Technology

Sepuluh Nopember Institute of Technology

Surabaya

2023

[This page intentionally left blank]

ABSTRAK

Nama Mahasiswa : Dion Andreas Solang

Judul Tugas Akhir : OPTIMISASI PENDETEKSIAN OBJEK KECIL YOLOv7 UNTUK MENDETEKSI OBJEK-OBJEK *AIRBORNE*

Pembimbing : 1. Reza Fuad Rachmadi, S.T., M.T., Ph.D
2. Dr. I Ketut Eddy Purnama, S.T., M.T.

Pada penelitian ini, kami menunjukan percobaan kami untuk meningkatkan kapabilitas YOLOv7 untuk mendeteksi objek airborne. Objek airborne tampak sangat kecil pada kamera karena jarak yang cukup jauh dari kamera. Oleh karena itu, YOLOv7 harus dioptimisasi untuk dapat mendeteksi objek-objek kecil. Beberapa modifikasi diajukan dan diuji pada penelitian ini. Modifikasi-modifikasi ini meliput perubahan arsitektur (Menambah layer deteksi, mengubah sumber feature-map, dan mengganti layer deteksi menjadi anchor-free), dan pada bag-of-freebies (rekalkulasi anchor, dan augmentasi mosaik). Hingga saat ini, kami menemukan bahwa kombinasi augmentasi mosaik, rekalkulasi anchor, dan mengubah sumber feature-map memberikan skor mAP yang paling tinggi 14,09% dibandingkan dengan YOLOv7 yang biasa (mAP=0%).

Kata Kunci: ***Deteksi Objek Kecil, YOLOv7, Modifikasi Arsitektur, Modifikasi Bag-of-Freebies, Objek Airborne***

[This page intentionally left blank]

ABSTRACT

Name : Dion Andreas Solang
Title : *YOLOv7 SMALL OBJECT DETECTION OPTIMIZATION TO DETECT AIRBORNE OBJECTS*
Advisors : 1. Reza Fuad Rachmadi, S.T., M.T., Ph.D
2. Dr. I Ketut Eddy Purnama, S.T., M.T.

In this research, we present an attempt to improve the capability of YOLOv7 to detect airborne objects. Airborne objects appear very small on cameras due to their distance to the camera. For that reason, YOLOv7 needs to be optimized to detect small objects. Several modification proposal was made and tested in this research. These modifications include changes in the architecture (Adding extra detection head, modifying feature-map source, and replacing detection head to a detached anchor-free head), and some bag-of-freebies applications (anchor recalculation, mosaic augmentation). At the current state, we found the combination of mosaic augmentation, anchor recalculation, and modifying feature-map source produces the greatest score in mAP, a 14.09% increase compared to the plain YOLOv7 (mAP=0%).

Keywords: ***Small Object Detection, YOLOv7, Architecture Modification, Bag-of-Freebies Modification, Airborne Object***

[This page intentionally left blank]

PREFACE

Puji dan syukur kehadirat Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque.

Penelitian ini disusun dalam rangka Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Keluarga, Ibu, Bapak dan Saudara tercinta yang telah Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero.
2. Bapak Nikola Tesla, S.T., M.T., selaku Quisque ullamcorper placerat ipsum. Cras nibh.
3. Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl.

Akhir kata, semoga Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus.

Surabaya, Juli 2023

Dion Andreas Solang

[This page intentionally left blank]

TABLE OF CONTENTS

[This page intentionally left blank]

LIST OF FIGURES

[This page intentionally left blank]

LIST OF TABLES

[This page intentionally left blank]

CHAPTER I
INTRODUCTION

1.1 Background

Autonomous Aerial Vehicle (AAV) have the potential to significantly impact industries, particularly in commercial delivery. One notable example is Amazon Prime Air, which is currently under development. Prime Air aims to deliver goods from Amazon warehouses directly to customers (Amazon, 2022). To accomplish this, AAV require a reliable and efficient autonomy system.

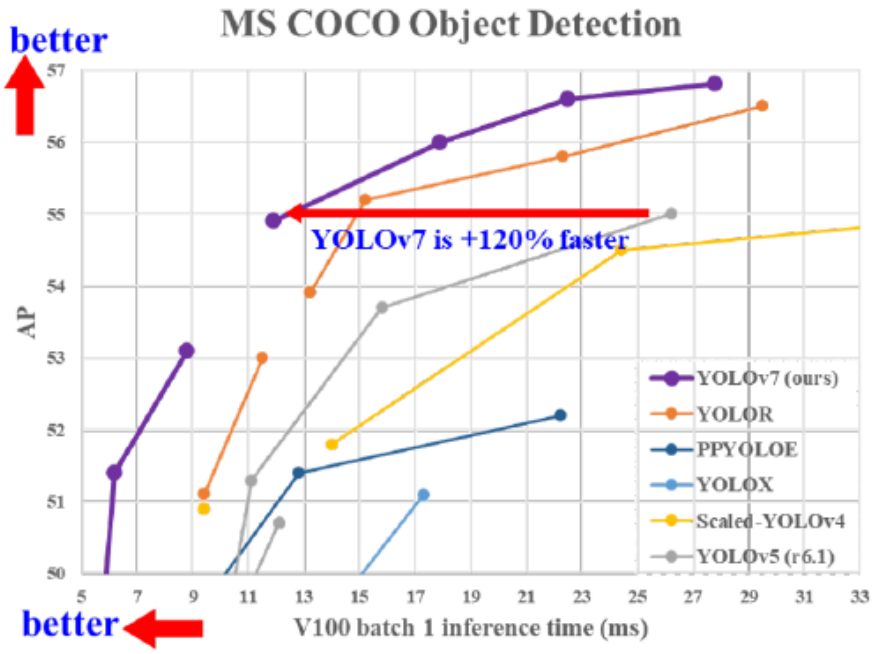
One of the most critical challenges in designing an AAV system is the ability to sense and avoid (SAA) obstacles. While the airspace in which AAVs operate may be relatively sparse, there is still risk of encountering static obstacles or airborne objects such as birds or drones. In commercial applications, such encounters can have consequences not only for the AAV itself but also for the items it carries. Ensuring effective SAA capabilities is essential to mitigate these risks and safeguard both the AAV and the valuable cargo it transports.

Most SAA system of AAVs includes camera as their primary sensor. Cameras provide visual perception to the AAV, real-time, in the form of images. These images must be processed by a computer vision algorithm to identify and localize the obstacles in it so that the AAV can estimate their position and plan actions it needs to do to avoid them. There are other onboard sensing options such as LiDAR or radar, but cameras are more favored due to their lighter weight, cheaper price, and relatively lower energy consumption compared to active sensors.

Airborne objects present a particularly challenging problem in SAA as they can appear unexpectedly and approach rapidly from long distances due to their inherent need for speed in flight. For this reason, it is important to detect airborne objects while they are still far away. Unfortunately, their far distance cause them to appear very small on images. Figure 1.1 show an

Source: “Airborne Object Tracking Dataset” (2021) under CDLA-Permissive 1.0

Figure 1.1: An example of airborne object dataset



Source: C.-Y. Wang, Bochkovskiy, et al. (2022) with permission (see Appendix C)

Figure 1.2: YOLOv7 performance on COCO dataset compared to other object detectors.

example of how airborne objects appear on images. In "Airborne Object Tracking Challenge" (2021), the airborne objects can appear in the range of 4 to 1000 px on a 2048×2448 px image, that is, around 0.00008 - 0.01 % of the image area.

For reasons listed above, the computer vision algorithm to be implemented to detect these airborne objects, must be able to detect small objects accurately. And not only that, it also must be able to do it in real-time. The fact that AAVs operating environments are outdoors, brings more trouble for the computer vision. Outdoor environment brings large complexity and additional variance to the image distribution, making it almost impossible to handcraft feature extractor for it. This is where non-handcrafted feature extractors come in handy, particularly the deep learning approaches. Deep learning models have shown incredible abilities in handling complex data. Given enough training data, a deep learning model can craft itself feature kernels that can extract the objects in an image, despite being subjected to complex outdoor environment. For this reason, deep learning models gained prominence in addressing these challenges.

Introducing YOLOv7, a state-of-the-art convolutional neural network based real-time object detector (C.-Y. Wang, Bochkovskiy, et al., 2022). At the time of the proposal for this research was made (November 2022), YOLOv7 outperform both in speed and accuracy of all known real-time object detectors with inference speed in the range of 5-160 FPS as seen on Figure 1.2. It also has the highest accuracy (56.8% AP) among object detectors with inference speed greater than 30 FPS on a V100 GPU. The capabilities of this cutting-edge architecture makes it well-suited for AAV computer vision system. However, all the performance metrics of YOLOv7 mentioned before are obtained by training the model using COCO 2017 dataset. A dataset which consist of general objects that people see in their daily life. COCO dataset is going to have very distinct distribution compared to airborne objects. As such, there would be a need for some modification to YOLOv7 so that it could detect airborne objects well.

The topic of this research is about modifying YOLOv7 with objective of optimizing it to detect small objects, which extends to airborne objects. We experimented with some modification to YOLOv7's bag-of-freebies, bag-of-specials, and network architecture. Then we benchmarked the modified models against the "Airborne Object Tracking Dataset" (2021), and picked the model with the highest AP score that can perform inference in real-time.

1.2 Problem Statement

YOLOv7 is not an object detection model specifically designed to detect small objects or airborne objects. Therefore, this research ask the following question.

- What modification can be done to YOLOv7 to improve its abilities in detecting airborne objects?

1.3 Purpose

The purpose of this research is to find modifications of YOLOv7 that would improve its ability in detecting airborne objects.

1.4 Problem Scope

In this research, we want to formulate the scope of the problem such that the modifications applied on YOLOv7 would not lose its real-time detection ability and is still reasonably computable/trainable. Otherwise, we can just run a Solomonoff induction on airborne object data and call the result a modification of YOLOv7. Thus, we state the following scopes for the problem.

- YOLOv7 is used as the baseline for modifications.
- The result of modification must be able to perform inference in real-time.
- The modified YOLOv7 must be trainable within a reasonable duration using the computational resource available, which is a computer which uses consumer GPU Nvidia RTX 2080 Ti.

[This page intentionally left blank]

CHAPTER II
LITERATURE REVIEW

2.1 Theoretical Basis

2.1.1 Object Detection Task

Object detection is a fundamental task in computer vision. It is a task that have 2 objective, localization and classification. Localization is about determining the spatial properties of an object in an image. E.g. by enclosing objects in bounding boxes, determining which part of the image is an object and which part is not. On the other hand, classification is about determining the class of objects. An object detection algorithm has to determine the label of an object whether they are an animal, human, necromancer, or anything in the set of classes.

To measure how well an object detection algorithm or model perform, computer vision researchers defined metrics to measure how fit a model prediction to an object and to measure how well the model perform across all objects. Example of those metrics are IoU and mAP.

Intersection Over Union (IoU)

Intersection over union (IoU) is a widely used metrics to determine how fit a predicted bounding box against the true bounding box. It is done by calculating the area of the intersection between the predicted and dividing it by the area of the union of those 2 boxes. This way, IoU is robust to objects of different sizes and aspect ratio.

Bounding box of an object can be large, small, wide, or tall. Since IoU consider the intersection of bounding boxes in relation to the union, IoU provides a normalized measure of the fitness of boxes, making it robust to the bounding boxes size. For example, if 2 boxes A and B intersect, there would be three possible cases, A fits B perfectly, A is inside B or vice versa, and some of A in B or vice versa as shown in Figure 2.1. When bounding box A fits B perfectly, the value of IoU would be 1 as the area of intersection is the same as the area of union. For the case where a bounding box is inside another, IoU can lower the score if the enclosed box's size is smaller compared to the enclosing box due to the normalization by the union area. For the case when the boxes partly intersect, the area of intersection and area of union would be suboptimal, leading to a lower IoU score. We can say that the intersection part of the IoU provide a score for the bounding boxes position similarities and the union provide score for size similarities.

To calculate the IoU of two bounding boxes A and B, the area of intersection and union must first be calculated. The following shows the process of calculating IoU.

- Calculate the area of intersection of A and B. Let (x_A, y_A) and (x_B, y_B) be coordinates of the center of the boxes A and B respectively, and let (w_A, h_A) and (w_B, h_B) be widths and heights of boxes A and B, as demonstrated in Figure 2.2.

The calculation for the area of intersection can be done like this:

$$\begin{aligned} x_{\text{inter}} &= \max(x_A, x_B) \\ y_{\text{inter}} &= \max(y_A, y_B) \\ w_{\text{inter}} &= \min(x_A + w_A, x_B + w_B) - x_{\text{inter}} \\ h_{\text{inter}} &= \min(y_A + h_A, y_B + h_B) - y_{\text{inter}} \\ \text{Area}_{\text{inter}} &= w_{\text{inter}} \times h_{\text{inter}} \end{aligned}$$

- Calculate the area of union.

  From set theory, we know that $|A \cup B| = |A| + |B| - |A \cap B|$. This equation also applies on calculating the area of union.

$$\begin{aligned} \text{Area}_{union} &= \text{Area}_A + \text{Area}_B - \text{Area}_{\text{inter}} \\ \text{Area}_{union} &= w_A \times h_A + w_B \times h_B - \text{Area}_{\text{inter}} \end{aligned}$$

- Finally, calculate IoU

$$IoU = \frac{\text{Area}_{\text{inter}}}{\text{Area}_{\text{union}}} \tag{2.1}$$

Mean Average Precision (mAP)

Average Precision (AP) is a popular metrics used to measure the capability of an object detection model for a given dataset. The main advantage of using AP are its ability to capture the precision recall tradeoff and its independence towards confidence threshold. AP has these 2 advantage as an effect of the way it is calculated.

Calculating AP is the same as calculating area under the curve (AUC) of precision and recall curve. Precision is defined as

$$P(\mathbb{S}_c, h, \tau, \epsilon) = \frac{\left|\left\{(\hat{B}, x) \in \mathbb{S} : \ \exists B \in h(x, \tau)_c, \ IoU(\hat{B}, B) > \epsilon\right\}\right|}{|\{B \in h(x, \tau)_c\}|} \tag{2.2}$$

Where h = hypothesis/model that predict bounding boxes
$\mathbb{S}_c$ = set of bounding boxes of class c in dataset paired with their respective image
$h(x, \tau)_c$ = bounding boxes predicted by h with class c
τ = confidence threshold for h
ϵ = IoU threshold for a positive prediction
B = Bounding box
x = image

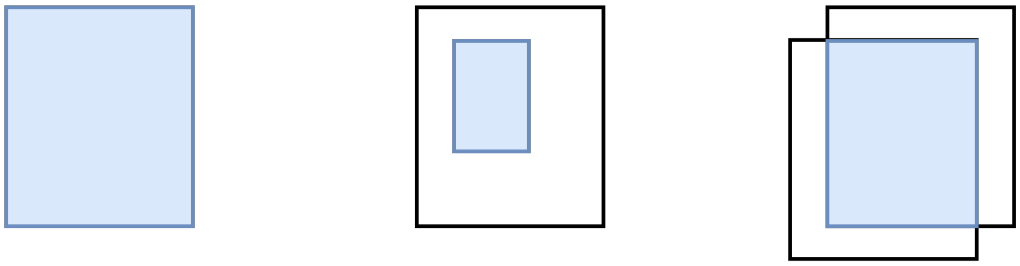


Figure 2.1: Three Cases of Bounding Boxes Intersection

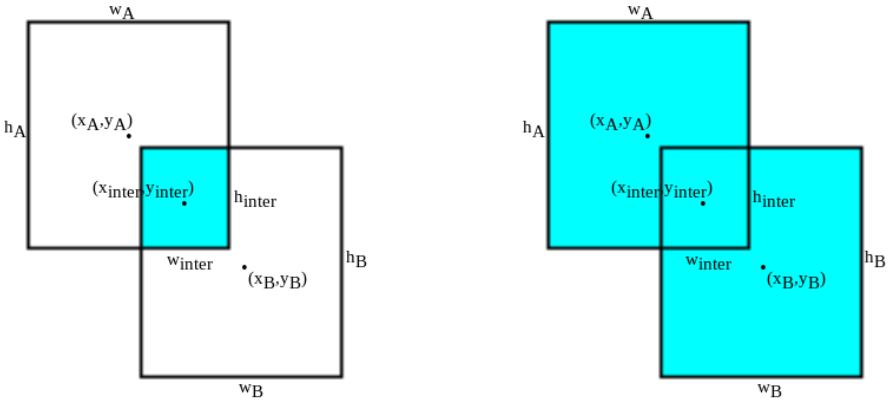


Figure 2.2: Intersection and Union of 2 Bounding Boxes

and for recall, it is defined as

$$R(\mathbb{S}_c, h, \tau, \epsilon) = \frac{\left|\left\{(\hat{B}, x) \in \mathbb{S} : \ \exists B \in h(x, \tau)_c, \ IoU(\hat{B}, B) > \epsilon\right\}\right|}{|\mathbb{S}_c|} \tag{2.3}$$

Then we define the precision recall curve as a parametric function

$$PR(\tau) = (R(\mathbb{S}_c, h, \tau, \epsilon), P(\mathbb{S}, h, \tau, \epsilon)) \tag{2.4}$$

Using equation 2.4, *PR* points in Figure 2.3 can be generated using $0 \leq \tau \leq 1$ Before calculating the AP however, the curve in Figure 2.3 is usually interpolated using Equation 2.5, which relies on Equation 2.6 that transformed *PR* curve to a functional relation of R to P. The interpolated curve can be seen on 2.4

$$p_{inter}(r) = \max_{\bar{r}>r} p(\bar{r}) \tag{2.5}$$

$$\text{Where } p(\bar{r}) = \max \{p : \forall (r, p) \in \{PR(\tau) : 0 \leq \tau \leq 1\} \ \wedge r = \bar{r}\} \tag{2.6}$$

The AP, which is the area under the curve then can be calculated using the following integral:

$$AP = \int_0^1 p_{inter}(r)\, dr \tag{2.7}$$

When calculating AP, the *IoU* threshold for positive detection is usually predefined. As example, for AP@50, we set the ϵ in Equation 2.2 and 2.3 to 50%. And for AP@75, the ϵ is set to 75%.

The AP calculation so far only works for a single class. To have a multiclass metric, we use mAP which is calculated as the average AP across all classes in the dataset.

$$mAP@X = \frac{1}{|\text{classes}|} \sum_{c \in \text{classes}} AP_c@X \tag{2.8}$$

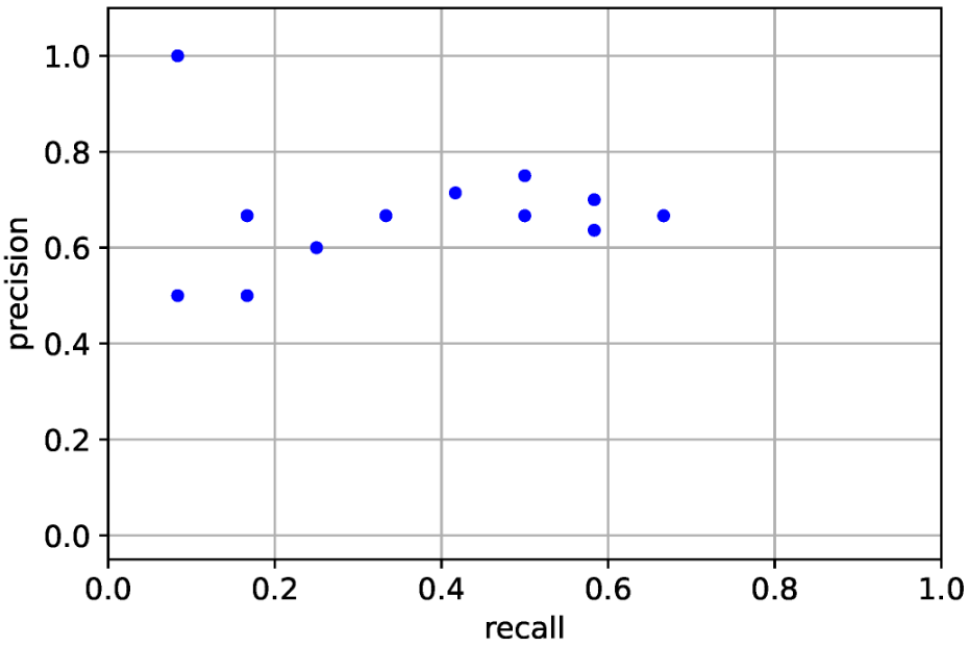
2.1.2 YOLO Family Architecture

YOLO is an abbreviation of "You Only Look Once" which describes what kind of neural network YOLO is, a single stage object detector. It means that this architecture predicts regions and classes both at once. In contrast, two-stage detector predicts objects' regions first and then predicts their classes. Detecting objects in a single-stage manner is what gave YOLO architecture the ability to infer in real-time. This is possible due to how YOLO architecture was designed. YOLO architecture consist of 3 main part, the head, the neck, and the backbone.

Backbone

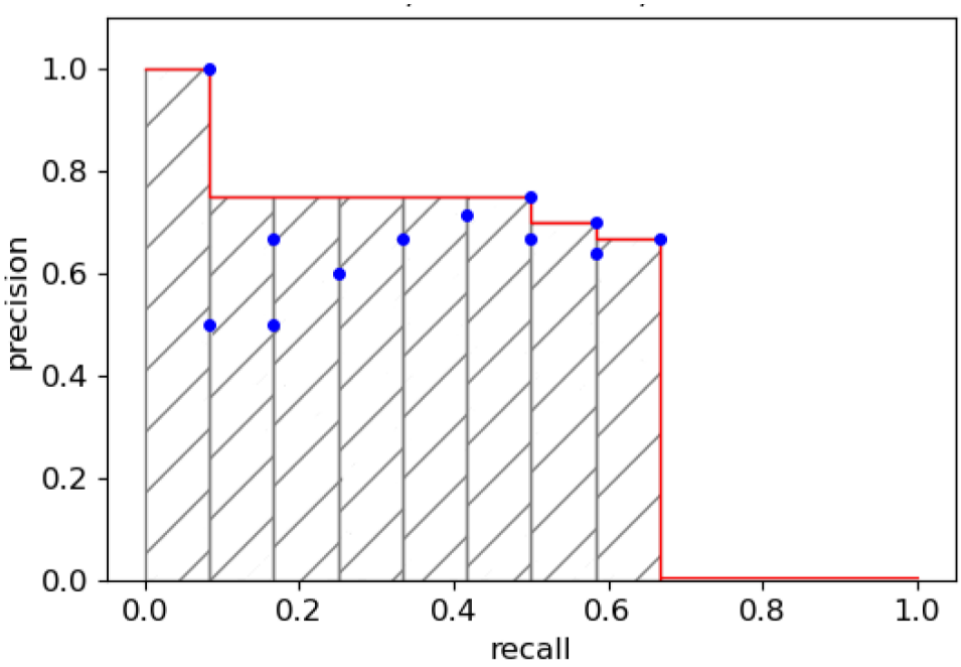
The backbone is the network that extract features from the inputted image. Typically, the backbone is composed of deep neural network layers that progressively down sample the spatial dimensions of the input while increasing the number of learned features or meaningful abstraction of the data.

The implementation of backbone in YOLO usually varies from one version to another. As example, Redmon and Farhadi (2016)'s YOLOv2 implemented Darknet-19 network as back-



Source: Padilla et al. (2021) under CC BY 4.0

Figure 2.3: PR Points Generated by Varying τ



Source: Padilla et al. (2021) under CC BY 4.0

Figure 2.4: PR Curve Interpolated

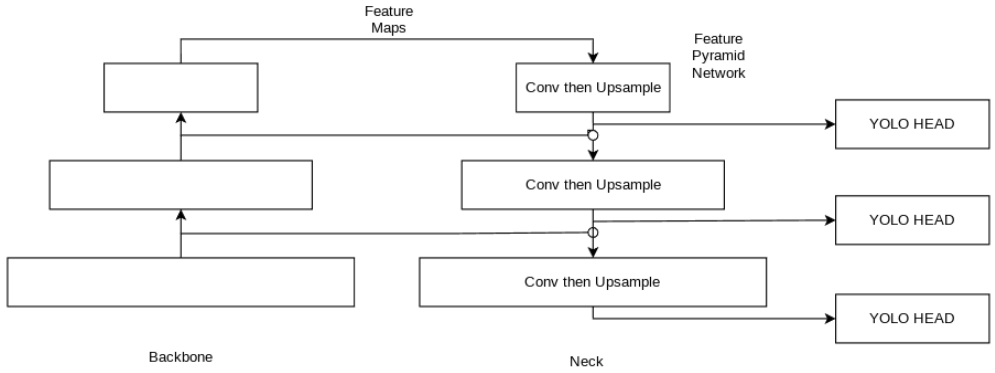


Figure 2.5: Feature Pyramid Network in YOLOv3

bone, Redmon and Farhadi (2018)'s YOLOv3 implemented Darknet-53, Bochkovskiy et al. (2020)'s YOLOv4 with their CSP-Darknet-53, and Zhang et al. (2021) with their non-CNN (Transformer) backbone. Each of these network has their own has their own advantages and disadvantage when it comes to accuracy, memory requirement, or latency.

The Neck

The neck is the intermediate network between backbone and head and its main function of the neck is to enhance features extracted by the backbone. The specific implementation of neck for each YOLO architecture is different one to the other, however. One common approach is to combine feature maps from different prediction scales of the YOLO network.

In YOLOv3, Redmon and Farhadi (2018) introduced YOLO prediction in multiple scale. To utilize information across scales, YOLOv3 fuses features from multiple parts of the backbone before up sampling them as seen on Figure 2.5 by using Feature Pyramid Network (FPN) as its neck.

A further improvement was made by Bochkovskiy et al. (2020) with their YOLOv4 by introducing Path Aggregation Network (PANet) for the neck. With PANet, feature are fused back to higher scale by adding another FPN-like layer after the original FPN but with reverse direction. This results in a better integration of multiscale information and increased the object detection accuracy.

Overall, the neck's role in YOLO architectures is to facilitate feature enhancement and fusion across different scales to improve detection performance.

The Head and The Anchors

The head is where the object detection happens. Extracted and enhanced features of the image is fed to the head in multiple different scales. For each scale, the head will predict a box for each $N_k \times N_k$ lattice point on feature map's grid. In total, each layer will output a tensor with size $N_k \times N_k \times [A_k \times (4 + 1 + C)]$ where N_k is the size of feature map grid at the k-th scale, A_k is the number of anchors for that scale, 4 is for the four offsets $[t_x, t_y, t_w, t_h]$ (see Figure 2.6), 1 is for the objectness score for the grid, and C is for the number of classes it has to predict.

Most of YOLO family architecture head utilizes anchor boxes to assist bounding box prediction. This technique is used in Redmon and Farhadi (2016), Redmon and Farhadi (2018), Bochkovskiy et al. (2020), C.-Y. Wang et al. (2020), Jocher et al. (2022), C.-Y. Wang et al.

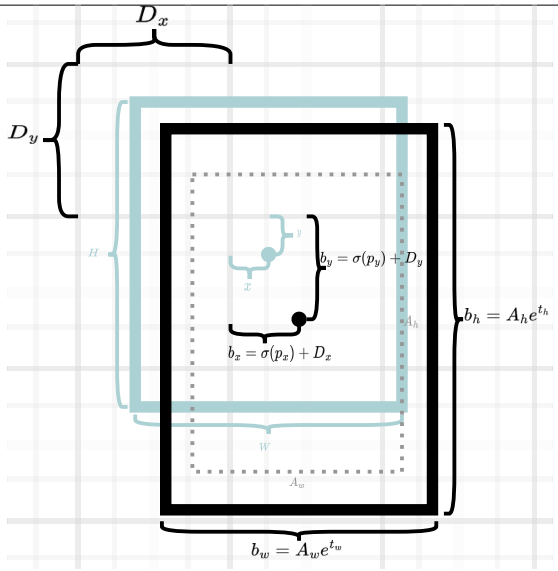


Figure 2.6: Head Layer Prediction for Anchor Box and Its Offsets from Lattice of The Feature Grid

(2021), and C.-Y. Wang, Bochkovskiy, et al. (2022). Instead of directly predicting the size and position of the bounding box, YOLO head predicts the offsets for each anchor boxes assigned to the head, then it utilizes the objectness score to pick which result of these anchor boxes to be used. Using anchor boxes allows the neural network to converge more quickly because it provides a prior knowledge of the dataset before training.

Loss Function

The goal of a YOLO architecture is to (1) predict if an object exist or not, (2) predict the bounding box of such object, and (3) predict the class of the object. These 3 loss functions that correspond to those objectives are called objectness loss, box loss, and class loss respectively. To update the weights on training, the total loss is calculated as the weighted sum of those 3 losses. In original YOLO, the loss functions were defined like the following. For localization loss, it is described by equation 2.9.

$$L_{box} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \tag{2.9}$$

Where S^2 = the total number of grid cells in the output,

B = the total number of anchor box per grid cells,

$$\mathbb{1}_{ij}^{obj} = \begin{cases} 1, & \text{if object present in grid} \\ 0, & \text{otherwise,} \end{cases}$$

(x_i, y_i) = the predicted coordinates of the center of object i

$(\hat{x}_i, \hat{y}_i)$ = the ground truth coordinates of the center of object

(w_i, h_i) = the predicted width and height of object i

$(\hat{w}_i, \hat{h}_i)$ = the ground truth width and height of object

For objectness loss, described by equation 2.10

$$L_{obj} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \alpha \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \tag{2.10}$$

$$Where\ \mathbb{1}_i^{\text{noobj}} = \begin{cases} 1, & \text{if object assigned to anchor j} \\ 0, & \text{otherwise} \end{cases}$$

$(C_i, \hat{C}_i)$ = predicted and ground truth confidence score for objectness of anchor

α = weight to punish false positives

And for class loss, described by 2.11

$$L_{class} = \sum_{i=0}^{S^2} \mathbb{1}_{ij} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{2.11}$$

Where $(p_i(c), \hat{p}_i(c))$ are the predicted and ground truth class probabilities for class c for object i.

The three losses combined to the final loss function in equation 2.12

$$L = \lambda_{box} L_{box} + \lambda_{obj} L_{obj} + \lambda_{class} L_{class} \tag{2.12}$$

$Where\ \lambda_{box}$ = the weight for localization,

λ_{obj} = weight for objectness

λ_{class} = weight for class

These 3 λ-s can be tuned to optimize the performance of a YOLO network.

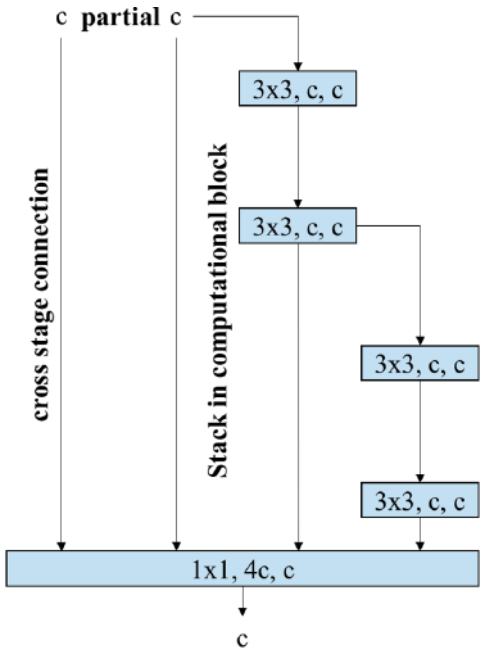
2.1.3 YOLOv7

As said in section 1.1, YOLOv7 is the state-of-the-art real-time object detector. It was made by the authors of YOLOv4, and by the time it was published (July 2022), it surpassed all known real-time object detectors both in speed and accuracy. To achieve this, YOLOv7 implemented some new changes and addition to the neural network.

Backbone

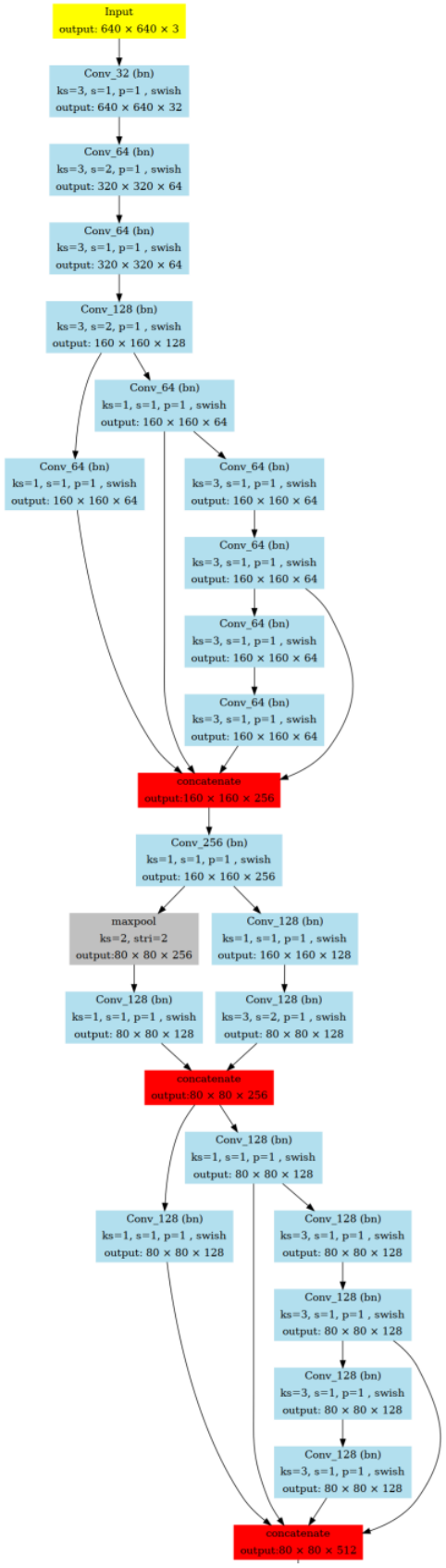
YOLOv7 implemented the Efficient Layer Aggregation Network (ELAN) and Extended-ELAN (E-ELAN) as the backbone of its network. ELAN is a convolutional neural network that was designed to extract features efficiently by controlling the shortest longest gradient path in the network (C.-Y. Wang, Liao, & Yeh, 2022). This choice of backbone allows YOLOv7 to perform prediction more accurately despite having fewer number of parameters.

Figure 2.7b shows how ELAN block in Figure 2.7a implemented in YOLOv7. First, the $640 \times 640 \times 3$ input x propagate through 3 convolution layers, down-sampling the dimension to 160×160 and increased the channels to 128. Then it encounter the first ELAN block. ELAN block did not change the dimension, but it doubled the number of channels, making the



Source: C.-Y. Wang, Bochkovskiy, et al. (2022)

(a) ELAN block



(b) First two ELAN blocks in YOLOv7

Figure 2.7: ELAN in YOLOv7

dimension of x to be $160\times160\times256$. Before propagating x to the next ELAN layer however, x is down-sampled to $80\times80\times256$ using max-pooling and convolution layers. Prior to each ELAN layers in the backbone, the feature will be down-sampled in its width and height dimension by half.

Label Assignment Strategy and Auxiliary Head

SimOTA, first introduced in YOLOX, is an algorithm to approximate Optimal Transport Assignment (OTA) in a faster way. Ge et al. (2021) introduced SimOTA in YOLOX because OTA was deemed too slow to compute as it was increasing the training time by 25%. YOLOv7 also implemented SimOTA for its dynamic label assigner.

YOLOv7 deep supervised its training process by attaching auxiliary heads to its neural network as seen on Figure 2.8a. These auxiliary heads are only used on training, on inference, they are removed from the neural network to improve latency, only the lead head is kept. There is a problem however with assigning labels to the auxiliary and lead heads. Most object detection networks that utilizes auxiliary heads have 2 independent label assigners, one for auxiliary heads and one for lead heads. YOLOv7 done things differently.

YOLOv7 proposed 2 way of assigning labels to auxiliary and lead heads. Lead head guided label assignment (Figure 2.8b) and coarse-to-fine lead head guided assignment (Figure 2.8c). For lead head guided label assignment, the assigner gives a copy of lead heads' label assignment to the auxiliary heads. For coarse-to-fine lead head guided assignment, the assigner works like lead head guided assigner but gives coarse label assignment to auxiliary head. Coarse label assignment is done by relaxing the positive sample constraints of the assigner.

Due to the relaxed constraints, coarse label assignment to auxiliary heads assigns more positive labels the auxilary heads' grids. This way, the network will learn more to recall. On inference, this recall ability would be filtered by the lead head to produce accurate prediction. C.-Y. Wang, Bochkovskiy, et al. (2022) find that coarse-to-fine label assignment produces the greatest AP scores.

Reparameterization

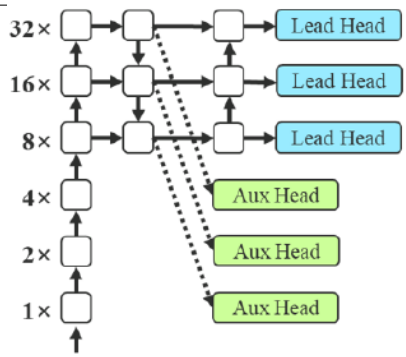
YOLOv7 utilized RepConv and YOLOR implicit layers in its network. In addition to that, YOLOv7 also uses Convolution-BatchNorm layer. These layers can be reparameterized after training to simplify the neural network, thereby reducing latency and memory usage but not hurting inference performance.

The reparameterization of these layers can be done after training by computing some mathematical simplification of some layer combination. These combinations of layers are YOLOR$^+$–Convolution–YOLOR$^+$, YOLOR*–Convolution–YOLOR*, and Convolution-BatchNorm.

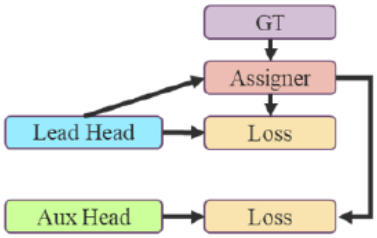
For combination of layers YOLOR$^+$–Convolution–YOLOR$^+$ layers, it can be reparameterized like the following.

$$\begin{aligned} x_{n+1} &= W(x_n + g_1(z_1)) + b + g_2(z_2) \\ &= W(x_n) + (W(g_1(z_1)) + b + g_2(z_2)) \\ &= W(x_n) + b' \end{aligned}$$

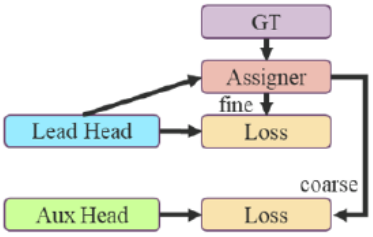
Observe that the reparameterization combined 3 layers into a single convolution layer with bias.



(a) Auxiliary heads attachment in YOLOv7



(b) Lead guided label assignment



(c) Coarse-to-fine lead guided label assignment

Source: C.-Y. Wang, Bochkovskiy, et al. (2022) with permission (see Appendix C)

Figure 2.8: YOLOv7 Label Assignment Strategy with Auxilary Heads

Then, For combination of YOLOR*–Convolution–YOLOR* layers:

$$\begin{aligned} x_{n+1} &= (W(g_1(z_1)x_n) + b)g_2(z_2) \\ &= g_2(z_2)g_1(z_1)W(x_n) + bg_2(z_2) \\ &= W'(x_n) + b' \end{aligned}$$

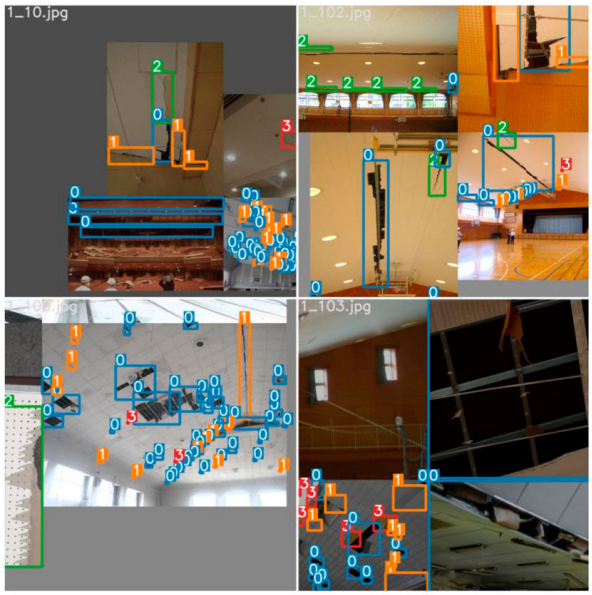
And finally for Convolution-BatchNorm:

$$\begin{aligned} x_{n+1} &= ((W(x_n) + b) - m)/s \\ &= (W(x_n) + (b - m))/s \\ &= (W/s)(x_n) + (b - m)/s \\ &= W'(x_n) + b' \end{aligned}$$

In summary, YOLOR$^+$–Conv–YOLOR$^+$ will be replaced with the original convolutional layer W but with bias $W(g_1(z_1)) + b + g_2(z_2)$. YOLOR*–Conv–YOLOR* layers will be replaced with a convolutional $g_2(z_2)g_1(z_1)W$ and bias $bg_2(z_2)$. And Conv–BatchNorm will be replaced with a convolutional W/s and bias $(b - m)/s$.

2.1.4 Anchor Recalculation

Anchor recalculation is a common method of introducing prior distribution of the dataset to an anchor-based object detection networks. Most of the time, anchors provided by pretrained YOLO weights are optimized for the common metrics dataset such as COCO2017 or VOC2012. Thus, recalculating anchor can help the neural network learn faster.



Source: P. Wang et al. (2022) under CC BY 4.0

Figure 2.9: Four example of mosaic augmentation

There are multiple ways of recalculating anchors. Some of them can be performed before training or during training. Most of pre-training anchor recalculation method involves with clustering the anchors to the dataset. This is done by using clustering algorithm such as K-means, Gaussian Mixture, and many others. Recalculating anchor on training is a little more complex to do as it will involve some loss function or architectural change on the object detection neural network. Zhong et al. (2018) for example add additional layer on detection part of object detectors that is connected to anchor modifiers such that the anchors will also be updated during training. C.-Y. Wang et al. (2021) mentioned that the implicit multiplication layer of their network can be purposed for anchor refinement.

2.1.5 Mosaic Augmentation

Mosaic augmentation was introduced in Ultralytics' implementation of YOLOv3 (Jocher, 2020). This method involves randomly selecting four images from the dataset and tiling them together into a single augmented image, as shown in Figure 2.9. The resulting augmented image has a mosaic-like appearance, thus named "mosaic" augmentation.

Several works (C.-Y. Wang et al., 2019; Bochkovskiy et al., 2020; Jocher et al., 2022) have reported improvements in accuracy when utilizing mosaic augmentation in object detection tasks. By combining multiple images into one, mosaic augmentation provides the model with more diverse training samples, leading to enhanced performance in localizing and classifying objects.

2.2 Related Works

2.2.1 YOLOv4-tiny with added head

Aditya et al. (2022) used YOLOv4-tiny as their Autonomous Surface Vehicle (ASV) object detector due to the computational device constraint. To detect objects that were at least 30

meters away from the ASV, they applied a modification of YOLOv4-tiny, which was YOLOv4-tiny but with additional head layer. The original YOLOv4-tiny only had 2 head layers, thus was only predicting in 2 scales. An addition of head layer allows it to predict in 3 scales. Using this modification, the network was able to detect small object better (as seen on Figure 2.10) and raised the overall mAP score by 4% without significantly reducing latency.

2.2.2 exYOLO

exYOLO is a modification of YOLOv3 to detect small objects (Xiao, 2021). Xiao (2021) thought that the features of small objects in an image would disappear after series stage of down-sampling in the neural network. To solve this, exYOLO added a feature enhancement before feature-fusion in the neck to one of the feature scale as seen in Figure 2.11. This change made exYOLO produce a higher mAP score on VOC2007 compared to its baseline YOLOv3.

2.2.3 YOLO-Z

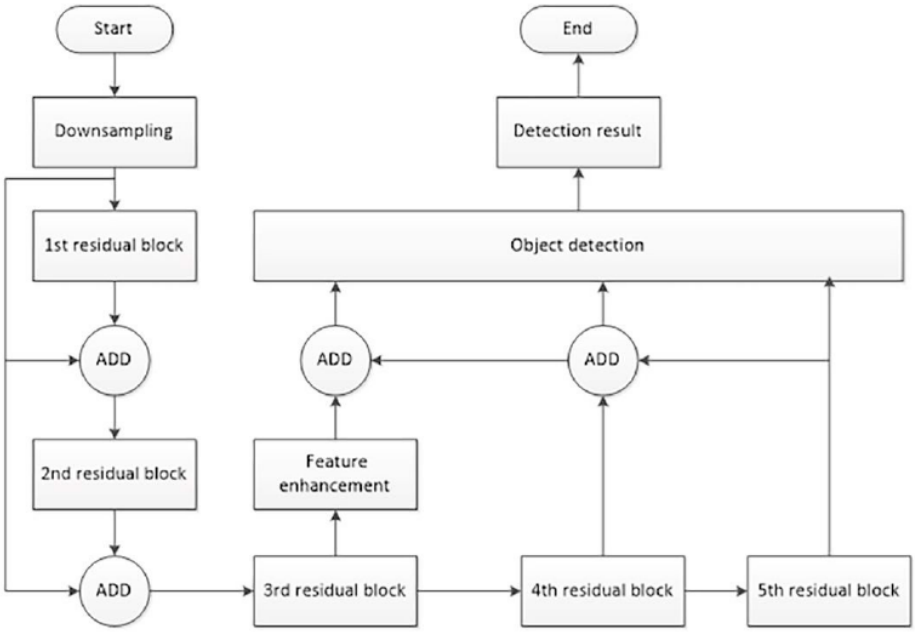
YOLO-Z is a derivative architecture of YOLOv5r5.0. This variant of YOLO modified the backbone, neck, and number of anchors of the original YOLOv5 to enhance its capability of detecting small objects (Benjumea et al., 2021). These changes are backbone change from YOLOv5r5.0 to a down-scaled DenseNet, neck change from FPN to biFPN on some YOLO-Z scales, and increasing the number of anchors used at each scale. These changes successfully improve the small object detection capability as seen on figure 2.12.

YOLO-Z was aimed to be used in autonomous racing car. In this high speed environment, early detection of obstacle is crucial to plan for action. For that reason, the autonomous racing car must detect the cone-shaped obstacles that are far away from it. Since objects that are far away appear small on image captured by camera, YOLO-Z was designed with purpose of small object detection.

(a) Regular YOLOv4-tiny prediction (b) YOLOv4-tiny with added head prediction

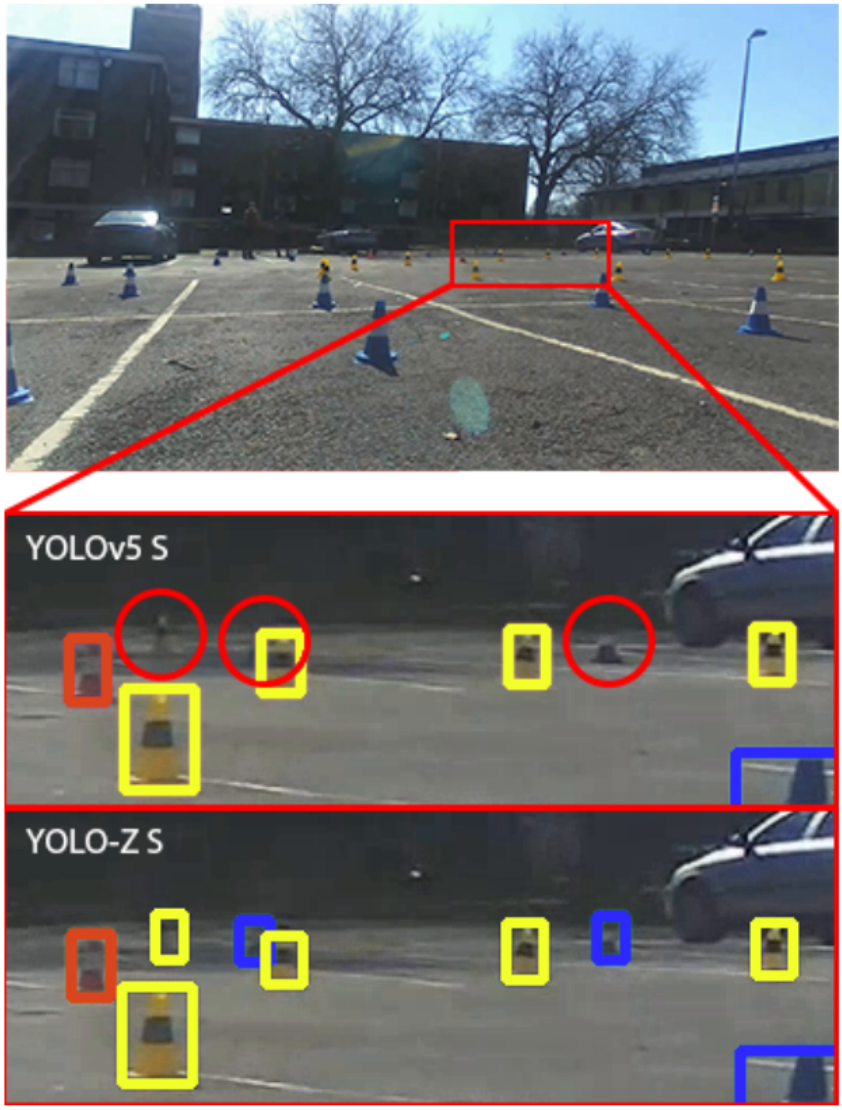
Source: Aditya et al. (2022)

Figure 2.10: Comparison of regular YOLOv4-tiny and YOLOv4-tiny with added head



Source: Xiao (2021) under CC BY-NC-ND 4.0 (see Appendix C)

Figure 2.11: Architecture of exYOLO



Source: Benjumea et al. (2021) with permission (see Appendix C)

Figure 2.12: Small objects in the image. Comparison of YOLOZ-S and YOLOv5-S. YOLOv5-S was not able to detect the circled objects.

[This page intentionally left blank]

CHAPTER III
METHODS

3.1 Model Searching Method

To find the best model for detecting small objects, trial and error will be performed by introducing modifications to YOLOv7. These modifications can include adding or changing the architecture, bag-of-specials, or bag-of-freebies. These modifications will be tested both independently and in combination with others to evaluate their impact on the model's performance.

The best model will be selected based on $mAP@50$ metric. This metric was chosen instead of $mAP@[50 : 95]$ because we don't expect for the model to be able to predict a tightly fit bounding boxes for small object and consider a loose 50% coverage IoU is sufficient.

The model searching method consist of six steps that need to be executed. These steps are dataset preparation, develop training system, create modification model configuration, training the model, analysis, and model selection. Figure 3.1 shows the order these steps will be executed.

To conduct this research, we first have to prepare the dataset. First, the dataset will be downloaded from "Airborne Object Tracking Dataset" (2021). Then, the labels of the dataset will be converted to darknet or COCO format. And finally, the dataset will be sampled into training set, validation set, and test set. The sampling will be done in a way that balances the number classes in each set so that there will be no dominating class during training.

The next thing to do is to develop a training system. The purpose of this system is to make the training process easily conducted and monitored. This system will include features such as train fail notification, train queuing, and overheating alert.

Moving on to the next, we have model configuration creation. Here we will create a model configuration file based on the modification that we want to try. The modification configurations will be made according to the modification candidates listed in section 3.2.

The next step is to train the model. The model configurations that was made in previous step will be built and trained from scratch (no pre-trained weights). In this step, the weights of

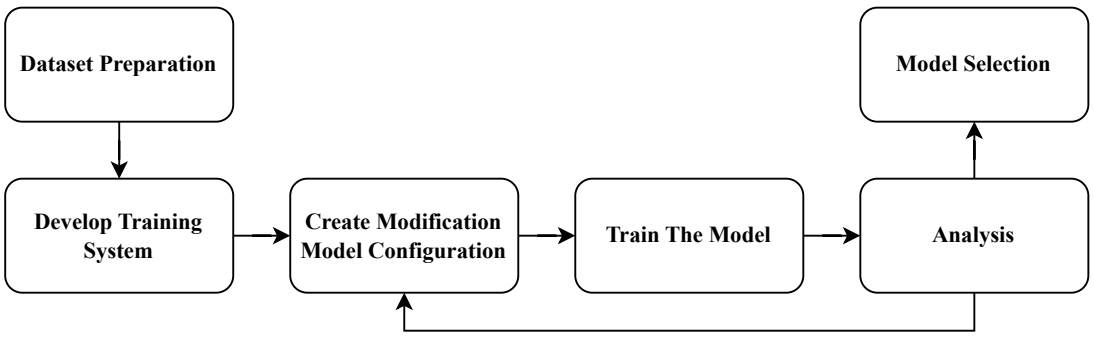
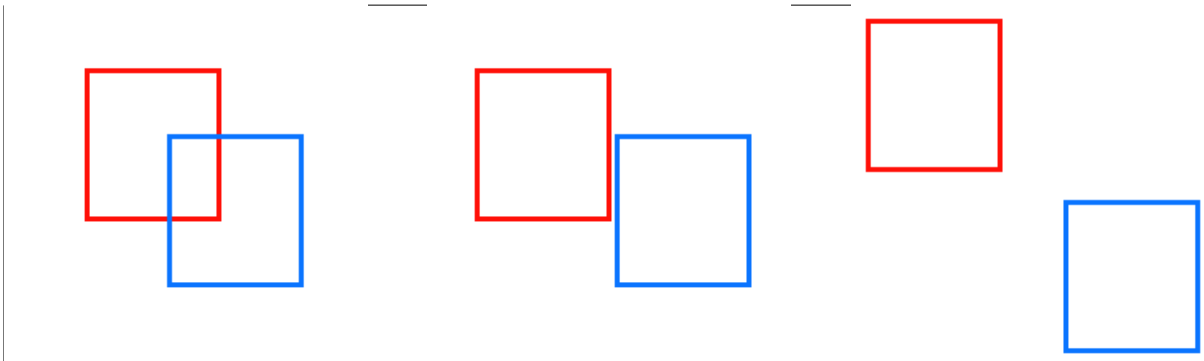


Figure 3.1: Search Steps



(a) $IoU > 0$ when 2 boxes intersect

(b) $IoU = 0$ when 2 boxes does not intersect

(c) $IoU = 0$ when 2 boxes does not intersect and far away

Figure 3.2: Cause of IoU vanishing gradient

the neural network, training history, and performance metrics will be generated for analysis.

In the analysis step, we will analyze the performance of the modified YOLOv7. This step is done to validate the result and to find other candidate of modification that might work. If such modification candidate was found, we will go back to create the model configuration, and then train the modification. This iterative process allows for continuous refinement and enhancement of the modifications capabilities for detecting small objects.

Finally, in the last step we will select the best model. We will select the model with the best performance among the modified YOLOv7 models. The model selection will be done based on $mAP@50$ metric, with constraint in inference latency. The model that qualify for selection must be able to perform inference with speed of atleast 10 FPS in a consumer GPU Nvidia RTX 2080Ti.

3.2 Modification Candidates

3.2.1 Mosaic Augmentation

As discussed in section 2.1.5, mosaic augmentation was able to increase the detection accuracy of small objects on many object detection neural networks. For this reason, we will experiment by training YOLOv7 with and without mosaic augmentation.

3.2.2 Pre-training Anchor Recalculation

In the implementation code of YOLOv7, the anchors provided was calculated based on COCO2017 dataset. We assume that the dataset distribution of airborne objects to be greatly different from COCO2017. As such, the anchors must be recalculated for better learning. This recalculation will be conducted using k-means algorithm. One problem however is that k-means might fail to cluster the dataset into the amount of anchors that we needed. To tackle this, we might have to cluster it in logarithmic coordinates.

3.2.3 Replacing Localization Loss to Extended IoU

Extended-IoU (EIoU) is a modification of IoU that are used in neural networks to tackle the problem of vanishing gradient (Peng & Yu, 2021). IoU are known to cause vanishing gradient problem due to its behavior when two bounding boxes are not intersecting. When 2 boxes A and

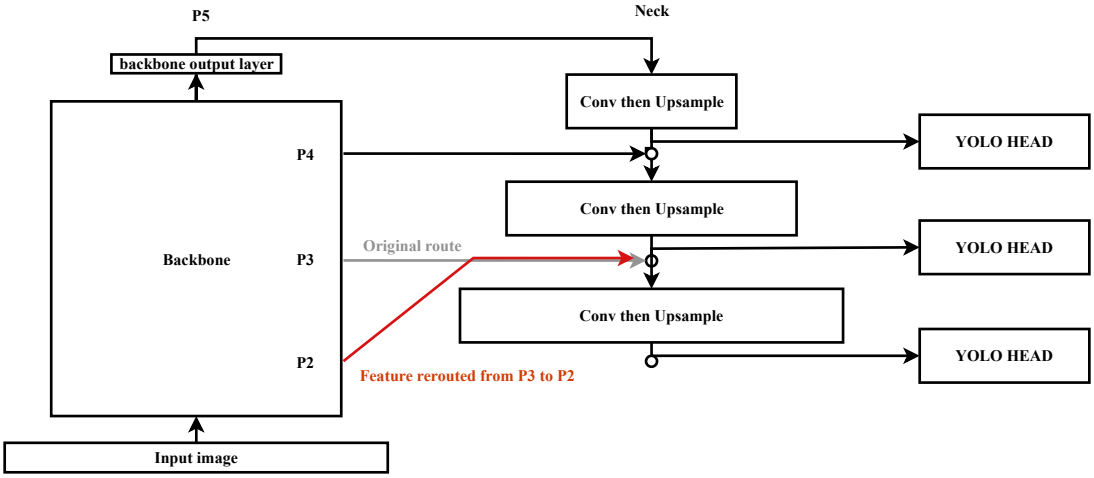


Figure 3.3: Rerouting Neck Connection to Earlier Stage

B are not intersecting, the area of intersection ($A \cap B$) would always be 0. This value doesn't give any information whether the boxes are far apart (Figure 3.2c) or near but not intersecting (Figure 3.2b). To solve this, loss involving IoU are usually paired with some regularization terms. For example Rezatofighi et al. (2019) and Zheng et al. (2019) proposed regularized IoUs named $GIoU$, $DIoU$, and $CIoU$.

YOLOv7 itself is using $CIoU$ for its localization loss. $CIoU$ is just regular IoU that is paired with distance and box aspect ratio regularization terms. The problem of these kinds of regularized IoU is that when the boxes intersect, it does not behave like IoU anymore. $GIoU$, $DIoU$ and $CIoU$ have residue of their regularization when the boxes intersect. Unfortunately, most metrics used to evaluate object detection algorithms depend on IoU (*e.g.* mAP). For this reason Peng and Yu (2021) designed EIoU. The main appeal of EIoU is that it behave exactly like IoU when the boxes intersect and gives non-positive value when the boxes do not intersect. By behaving exactly like IoU, it is hoped that the model would perform better on the IoU-based metrics.

3.2.4 Utilizing Earlier Feature Map Stage

In a deep neural network, the extracted feature/abstraction of the data become more prominent as the input data passes through deeper layers. However, so are the loss of information. For small object detection, every information in an image is crucial. There is a possibility that the features of small objects are lost as the data propagates through the network.

If we view it in a data path network design perspective, the greater the length of the path from input to output, the more information will be lost. Therefore, we propose to reroute the original connection from backbone to the neck to an earlier stage of inference. For example, YOLOv7 take feature maps from P3, P4, and P5 scales of the backbone, we can reroute the connection from P3 to P2 as shown in Figure 3.3. The data path from input image to head routed to P2 is shorter than with P3.

3.2.5 Additional YOLO Head

An additional YOLO head means an extra stage of detection. With more stage of detection, the large variance of the objects in the dataset can be learned. This is especially good for

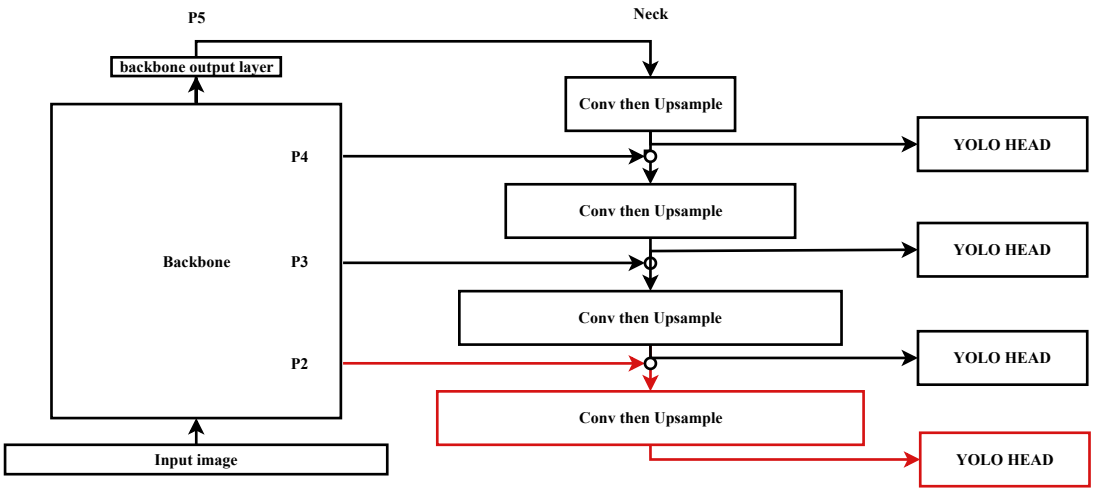


Figure 3.4: Adding an Extra Head Layer

"Airborne Object Tracking Dataset" (2021) where the area of the bounding boxes in the dataset can be orders of magnitude apart as discussed in section 3.5.

3.2.6 Replace YOLO Head to Decoupled Anchor-free Head

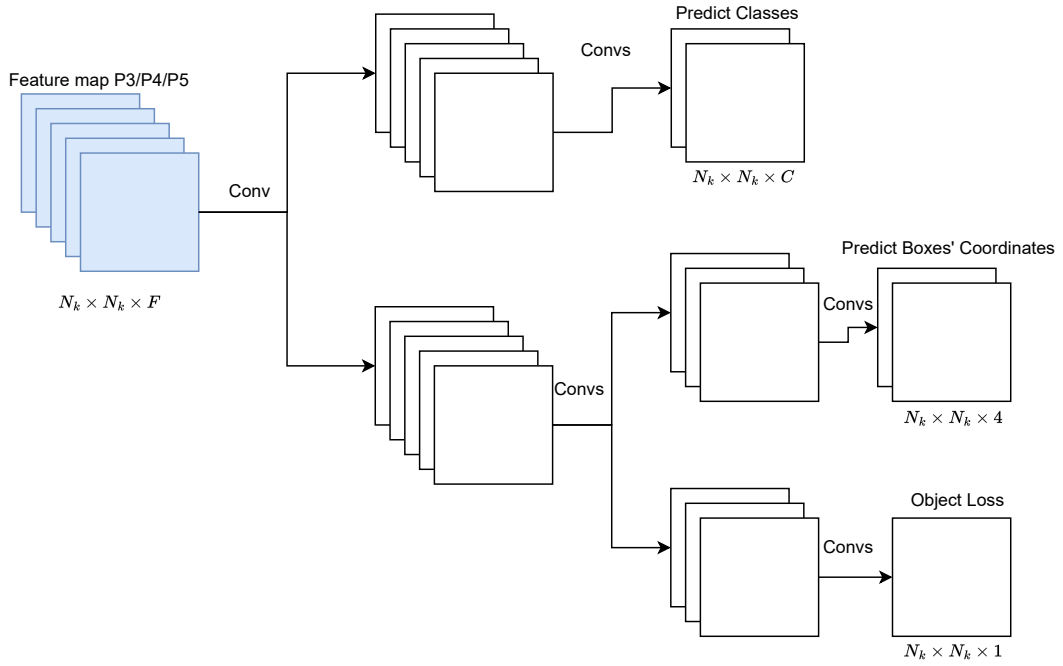
YOLOv6 and YOLOX used a different kind of head layer compared to usual YOLO (Ge et al., 2021; Li et al., 2022). In mainstream YOLO, prediction for bounding box and classes are both calculated on the same layer. With decoupled anchor-free head, the layers are separated for class prediction and bounding box prediction, and bounding box prediction are done using 4 value without anchor as seen on Figure 3.5. Using decoupled anchor-free head gives us 2 advantages. 1) Reducing the amount of design parameters as we don't have to introduce anchor boxes to the model. 2) Reducing the complexity of interpreting prediction result. Advantage (1) is especially enticing. There is a possibility of us introducing bad priors to the neural network by poorly clustering anchor boxes. By having a model that less dependent on prior, we can reduce such possibility.

A thing to consider when applying decoupled anchor-free head to YOLOv7 is the label assigner. YOLOv7 and YOLOX uses SimOTA as its default label assigner. However, Li et al. (2022) YOLOv6 uses Task-aligned Assigner (TAL) for its label assigner. They reported that TAL performs better than SimOTA. Therefore, in applying decoupled anchor-free head, it might be better to use TAL as label assigner.

3.2.7 Partitioning Image

Partitioning the image has the potential to improve the accuracy of small object detection. By partitioning images into multiple smaller images like shown in Figure 3.6, the down scaling factor of image to neural network input size is reduced, thereby reducing the loss of information. This approach allows the network to retain finer details and preserve the integrity of small objects, ultimately leading to more accurate detection results.

One challenge associated with partitioning image is the increase in inference latency. When an image is partitioned into multiple sections, the neural network needs to perform separate detections on each partition. As a result, the total inference time is multiplied by the number of partitions. For instance, if we divide an image into four parts, the network will have



Prediction for Classes and Boxes are decoupled unlike original YOLO head

Figure 3.5: Decoupled Anchor-free Head

Figure 3.6: An Image Partitioned to 4 Images

to carry out four separate detections, leading to a significant increase in latency. Therefore, it is important to find the correct balance between the number of partition, and network input size.

3.3 Computational Resource

To conduct this research, we will be using a computer with the following specification:

- CPU — Intel® Core™ i5-9400F CPU @ 2.90GHz
- GPU — Nvidia Geforce RTX 2080 Ti
  - Memory — 11 GB
  - CUDA Compute Capability — 7.5
- RAM — 12 GB
- Hard Drive Available Memory — 1.3 TB
- Operating System — Ubuntu 20.04
- CUDA Toolkit Version — 11.7
- PyTorch Version — 1.13.1

3.4 Pilot Test

To ensure the reliability of the hardware, and to find the hardware-induced constraint, we performed A pilot test. In this pilot test, we trained several YOLOv7 models with varying input sizes on a dummy dataset for 300 epochs. The dummy dataset have the same dimension as the "Airborne Object Tracking Dataset" (2021). The results are presented in Table 3.1.

From Table 3.1, we want to pick a baseline model to apply our modification. Since we want to persist the features of the 2048×2448 px images as much as possible, we want to pick a model that has the largest input size but still able to give enough for some changes in architecture. For this reason, we picked YOLOv7 (entry number 3) with input size 1600 and batch size 1 as the baseline. This model can be trained in a reasonable time (20 hours) and enough spare room in the VRAM for architecture modifications.

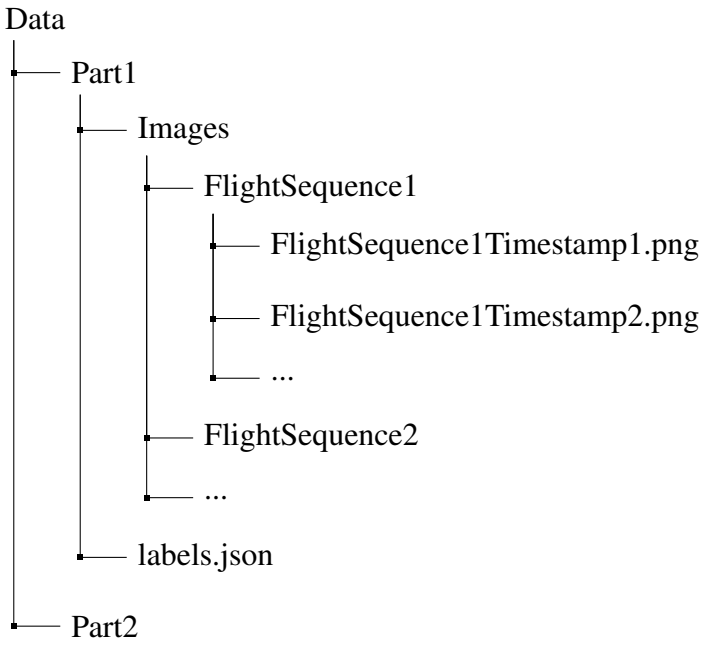
Table 3.1: Pilot Test Training

No	Model	Input Size	Training Time (300 epochs)	VRAM Usage	Average GPU Temperature
0	`YOLOv7`	640	4.0h	4.5GB	82
1	`YOLOv7 batch-size=2`	640	2.5h	6.5GB	82
2	`YOLOv7`	960	4.7h	6.5GB	84
3	`YOLOv7`	1600	20h	9.0GB	84
3	`YOLOv7 batch-size=2`	1600	20h	Out of Memory	85
4	`YOLOv7`	2000	N/A	Out of Memory	N/A
5	`YOLOv7-w6`	1600	N/A	10GB	85
6	`YOLOv7-d6`	1600	N/A	Out of Memory	N/A
7	`YOLOv7-e6`	1600	N/A	Out of Memory	N/A
8	`YOLOv7-e6e`	1600	N/A	Out of Memory	N/A

3.5 Dataset Preparation

3.5.1 Dataset Source

Dataset containing annotated images of airborne objects can be obtained from "Airborne Object Tracking Dataset" (2021) that was hosted in an AWS S3 Bucket. The dataset consist of several Unmanned Aerial Vehicle (UAV) flights' frame-by-frame camera capture. The dataset is structured in the following manner:



And each labels JSON file, `labels.json` on each part of the data, is structured like the following:

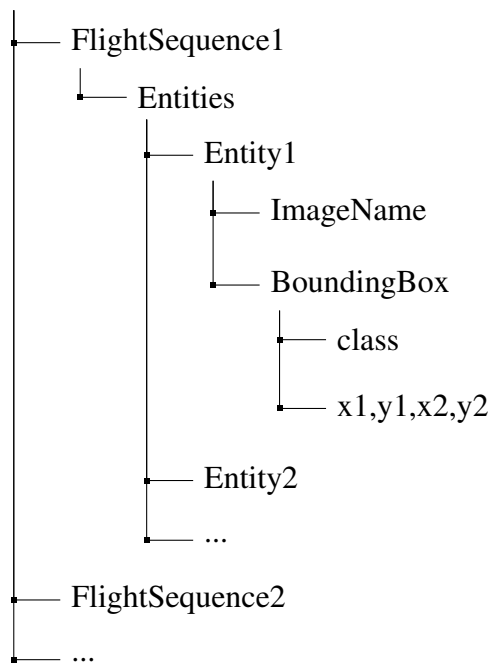


Table 3.2: Original Dataset Splits

Splits	Size (TB)	UAV Flight Sequences	Total Images	Total Labels
Training	11,3	4154	4975765	2891891
Validation + Test	2.1	789	943852	496075
Total	13,4	4943	5919617	3387966

Table 3.3: Dataset Objects' Classes Distribution

Splits	Total Objects	Airplane	Helicopter	Bird	*Other 4 Classes*
Training	2,89 M	0,79 M	1,22 M	0,33 M	0,54 M
Validation + Test	0,50 M	0,13 M	0,17 M	0,06 M	0,14 M
Total	3,39 M	0,92 M	1,39 M	0,39 M	0,69 M

Notice that the labels are structured in an entity based manner. This made the label conversion process more complex. Handling this labelling structure will be discussed more in the next section.

Dataset Statistics

There are 7 classes in the dataset, but there are 3 dominating classes. The size of the dataset and its class distribution can be seen on Table 3.2 and 3.3. In total, the dataset is a 13.4 TB images of 3.39 million objects.

If we look at the area (in px) distribution of the objects' bounding boxes, we will find the histogram in Figure 3.7. Take note that in the figure, the x-axis is using in logarithmic scale, which means the true data is very right skewed. The median of the areas is $314px$, while the 5th and 95th percentile are $36px$ and $5061px$, three orders of magnitude difference.

The widths and heights of the objects themselves are distributed in a way shown in Figure 3.8: The figure uses logarithmic scale for x and y, and the dashed lines represent the median for x and y axis. Meanwhile, The distribution of objects position in the camera can be seen on Figure 3.9. We can see that most of the objects are around the center of the image.

3.5.2 Dataset Sampling

Sampling Distribution

As seen on section 3.5.1, The size of the dataset is massive. For this reason, we only downloaded the planned encounters from the AWS S3 Bucket which in total amounted to 1.1 TB. As this number is still too large due to the limitation of computational resource, we will only sample some of them for training and testing. We will sample in total 700 images from the dataset with splits as shown in Table 3.4.

Sampling and Label Conversion Procedure

Sampling the dataset was not straightforward. Due to how the "Airborne Object Tracking Dataset" (2021) files and labels was structured, we had to make a sampling and converting

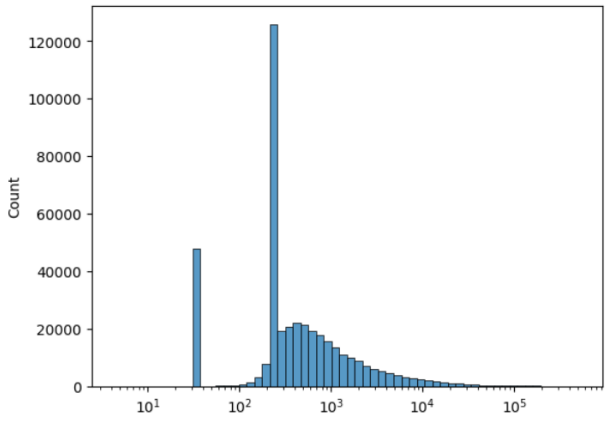


Figure 3.7: Distribution of Bounding Boxes' Area

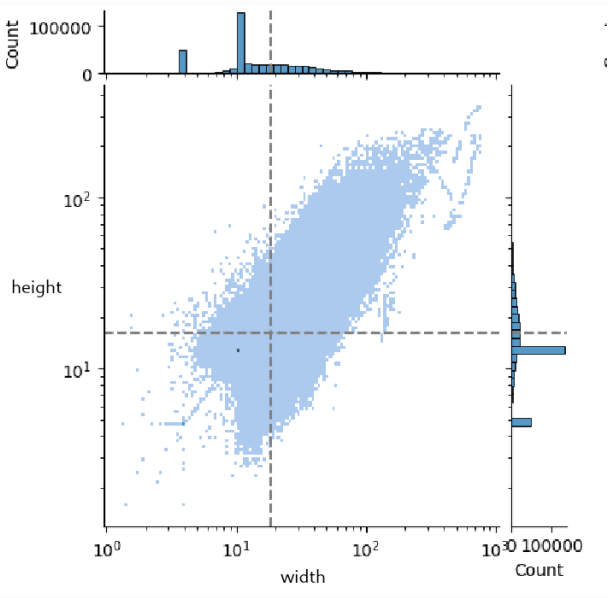
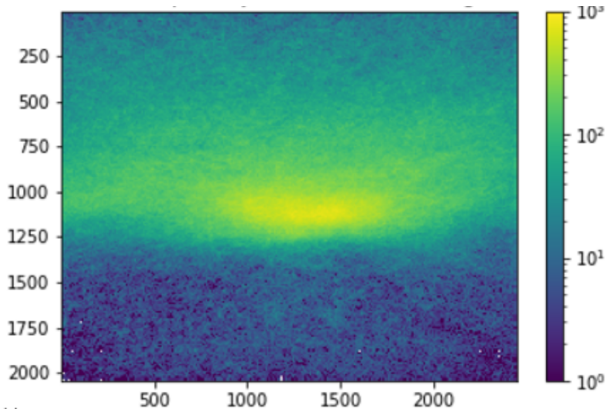


Figure 3.8: Distribution of Bounding Boxes' Widths and Heights



Source: "Airborne Object Tracking Dataset" (2021) under CDLA-Permissive 1.0

Figure 3.9: Distribution of Objects' Position

Table 3.4: Dataset Sampling Distribution

Splits	Total Images	Classes Percentage				
		Airplane	Helicopter	Bird	Drone	Negative
Training	400	23.75%	23.75%	23.75%	23.75%	5%
Validation	100	20%	20%	20%	20%	20%
Test	200	20%	20%	20%	20%	20%

procedure like the following:

1. Open all `labels.json` file and load all entities into a single list called `entities`
2. Let `Img2Boxes` be a dictionary with image names as key and list of bounding boxes in the image as value.
3. Let `Cls2Img` be a dictionary with 4 class names as key (airplane, bird, etc) and set of image name that contain object of those classes as the value.
4. Populate `Img2Boxes` and `Cls2Img` in the following manner:

```
For entity in entities:
  Img2Boxes[entity.ImageName].append(entity.BoundingBox)
  Cls2Img[entity.BoundingBox.class].append(entity.ImageName)
```

5. Calculate the total image needed for each class according to Table 3.4.
   e.g. Total image needed for airplane class is $\frac{400\times23.75+100\times20+200\times20}{100}$.
6. Distribute the images to train, valid and test.

```
For class in list_of_classes:
  Images[class] = Cls2Img[class].random_choice(TotalImageNeeded[↩
     class])

For class in list_of_classes[-1]: // except negative sample
  addtotrain = images[class][:23.75/63.75*TotalImageNeeded[class]]
  images[class] = images[class][23.75/63.75*TotalImageNeeded[class↩
     ]:]
  train.append(addtotrain)

  do the same to valid and test
process the negative sample independently
```

7. Convert `train`, `valid`, `test` to darknet format.

```
For img in train:
  label = []
  For boxes in Img2Boxes[img]
    // to_darknet converts class xyxy to class x y w h normalized
    label.append(to_darknet(boxes.BoundingBox))

  copy(img, train_folder+img)
```

```
    write(label, train_folder+img.with_suffix('.txt'))


repeat this for valid and test
```

[This page intentionally left blank]

CHAPTER IV
EXPERIMENTS

4.1 Baseline Performance

For comparison purposes, we first measure the baseline performance of YOLOv7 with all modification candidates from section 3.2 stripped. We call this model as `YOLOv7-plain`. After 300 epochs of training with batch-size 1 like in the experiment setup, we found that this model was unable to detect anything on the test set (a valid detection is detection with $IoU > 0.5$ to groundtruth).

4.2 Mosaic Augmentation and Anchor Recalculation

In this section, we present the effect of mosaic augmentation and anchor recalulation on the $mAP@50$ score when applied independently and combinatively. For comparison purposes, we assigned names the models as `YOLOv7-M`, `YOLOv7-AR`, and `YOLOv7-MAR`. These names correspond to `YOLOv7-plain` with mosaic augmentation, `YOLOv7-plain` with anchor recalculation, and `YOLOv7-plain` with both mosaic augmentation and anchor recalculation, respectively.

The process of applying mosaic augmentation to the dataset is pretty straightforward as YOLOv7's implementation code already provide the mosaic augmentation tool. A mosaic augmentation result example can be seen in Figure 4.1.

Anchor recalculation is done by clustering the training data's widths and heights to 9 centroid using k-means algorithm. However, due to the skewed distribution of the dataset, regular k-means (with L^2 Distance) actually failed to form 9 clusters. To fix this, we used a distance function

$$L = \sqrt{\ln^2 \frac{w_1}{w_2} + \ln^2 \frac{h_1}{h_2}} \tag{4.1}$$

Distance function in equation 4.1 was actually an L^2 distance but with log-transformed space. With this function, k-means was finally able to cluster the data into 9 centroids. These centroids are used by the head layers as anchors. Each head uses 3. The comparison of the original anchor and recalculated anchor can be seen in Table 4.1 and Figure 4.2.

If we observe the distribution of anchors before and after recalculation in Figure 4.2, we can see that indeed the recalculated anchors cover more of the dataset than the original. 8 out of 9 of the original anchors are placed in the first quadrant of the median axis. It means that

Table 4.1: Anchor Points Before and After Recalculation

Head	Original Anchors	Recalculated Anchors
1	[12,16], [19,36], [40,28]	[4,4], [14,5], [11,11]
2	[36,75], [76,55], [72,146]	[27,8], [18,17], [41,14]
3	[142,110], [192,243], [459,401]	[43,32], [86,27], [149,70]

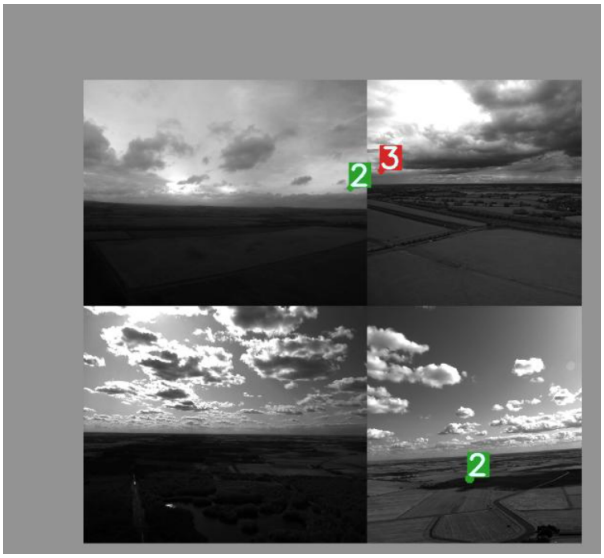


Figure 4.1: An example of mosaic augmented image

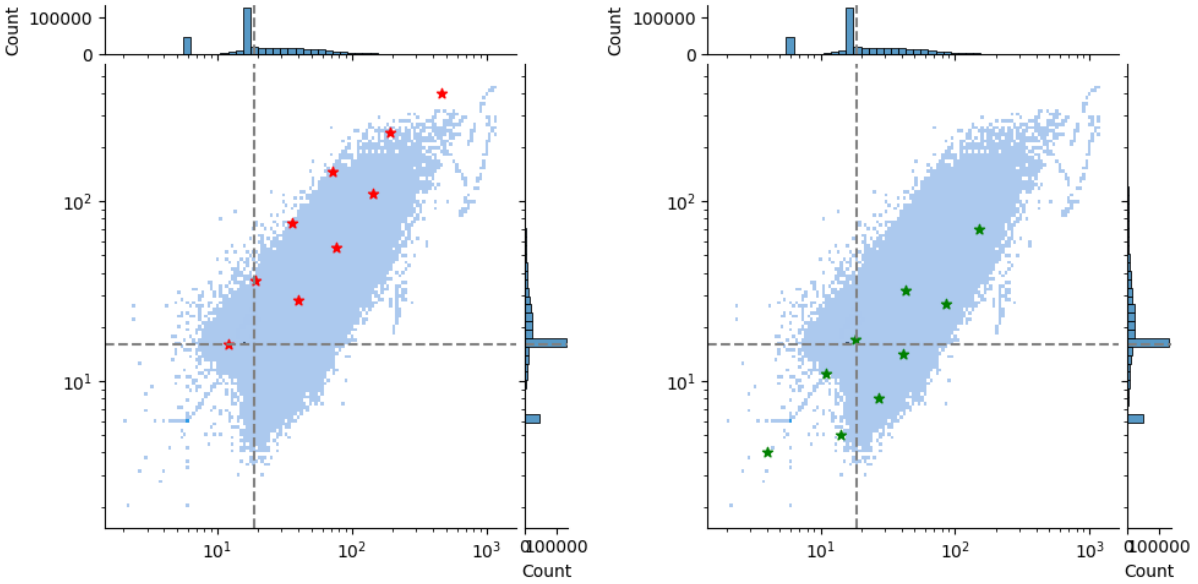


Figure 4.2: Anchor Distribution on Dataset. Left: Original Anchor. Right: Recalulated Anchors

Table 4.2: Mosaic Augmentation and Anchor Recalculation Performance

No	Model	mAP@50
0	`YOLOv7-plain`	0%
1	`YOLOv7-M`	0%
2	`YOLOv7-AR`	0%
3	**`YOLOv7-MAR`**	**11.2%**
	Improvement	**+11.2%**

those 8 anchors are responsible for only 25% of the dataset and leave the rest to 1 anchor. This is very inefficient. In contrast, the recalculated anchors have a more balanced distribution with anchors distributed across each quadrant.

Performance

The performance of the three models, namely `YOLOv7-M`, `YOLOv7-AR`, and `YOLOv7-MAR`, was compared to that of the baseline model YOLOv7-plain. The results are presented in Table 4.2. It can be observed from the table that YOLOv7-plain alone was unable to detect any objects in the test set. However, after applying both mosaic augmentation and anchor recalculation, the model achieved a notable improvement with a score of 11.2% at mAP@50.

Since the model `YOLOv7-MAR` was the only model that was capable of detection in the test set, we henceforth establish this model as the baseline for further modification. Meaning, in the subsequent sections, any modifications mentioned should be presumed to be composed of mosaic augmentation and anchor recalculation, in addition to the respective modification that was applied, unless explicitly stated otherwise.

4.3 Replacing Localization Loss to EIoU

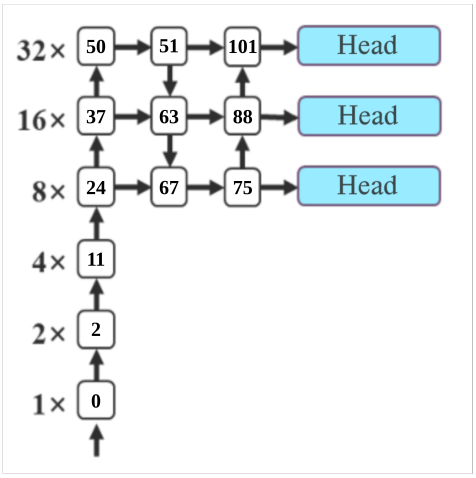
In this section, we experimented with $EIoU$. We replaced the original $CIoU$ loss of YOLOv7 to pure $EIoU$ and convexcified version of $EIoU$. The convexciation was done by modifying the $EIoU$ loss from $-EIoU$ to $(1 - EIoU)^2$ for better gradient dynamics during training.

The performance of these modifications can be seen in Table 4.3

Surprisingly, although $EIoU$ outperformed $CIoU$ when applied to networks like Faster-RCNN+ResNet and RetinaNet as Peng and Yu (2021) claimed, it performed worse when applied to YOLOv7 with "Airborne Object Tracking Dataset" (2021). Even after convexciation, $CIoU$ still outperform $EIoU$ by 6.28%.

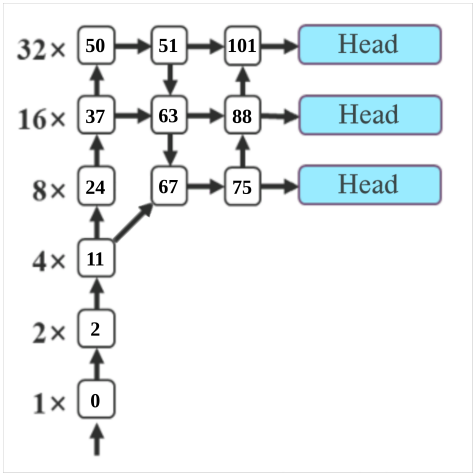
Table 4.3: EIoU Localization Loss Performance

No	Modification	mAP@50
0	**`YOLOv7-MAR +CIoU`** (original)	**11.2%**
1	`YOLOv7-MAR + EIoU`	0%
2	`YOLOv7-MAR + EIoU + Convexication`	4.92%
	Peningkatan	**-6.28%**



Source: C.-Y. Wang, Bochkovskiy, et al. (2022) with permission (see Appendix C)

(a) Before Rerouting



(b) After Rerouting

Figure 4.3: Modifying Connection to Earlier Feature Map Stage

Table 4.4: Performance of The Rerouted Model

No	Modification	mAP@50
0	`YOLOv7-MAR`	11.2%
1	**`YOLOv7-MAR + rerouting`**	**14.09%**
	Improvement	**+2.98%**

4.4 Utilizing Earlier Feature Map Stage

For this modification, we reconfigured the source of the feature map by redirecting it from P3 to P2, as shown in Figure 4.3. Specifically, we adjusted the routing on layer 66, changing it from 24 to 11. Since the output of layer 11 is at a different scale compared to 24 (with a scaling factor of 2^{-2} and 2^{-3} respectively), we modified the upsampling factor of layer 55 from 2 to 4 to ensure size compatibility with layer 11. However, this change in upsampling disrupted the subsequent layers. To solve this, we performed downsampling after layer 75 (connected to the head) by setting layer 77's stride to 2 and layer 79's stride to 4, resolving the size mismatch in the subsequent layers. The complete configuration of the layers can be seen in the appendix.

As presented in Table 4.4, rerouting the source of feature map to an earlier stage produce an improvement of 2.98% compared to `YOLOv7-MAR`.

4.5 Additional YOLO Detection Head

To add additional head layer, we utilized the feature map at the P2 scale of the network. We introduced an upsampling block after layer 75 and fused it with layer 11 at layer 79 by concatenation. To maintain the continuity of the network, we introduced a downsampling block that concludes at layer 87. The subsequent layers in the network retained their original structure, but with layer numbering shifted by 25 as seen on Figure 4.4.

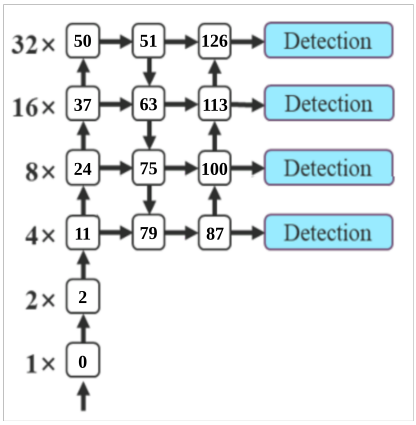


Figure 4.4: Model Architecture After Increasing The Head

Table 4.5: Additional Head Performance

No	Modification	mAP@50
0	`YOLOv7-MAR`	**11.2%**
1	`YOLOv7-MAR + more head`	5.19%
	Improvement	**-6%**

As depicted in Table 4.5, additional head layer hurt the performance of the model. This is counterintuitive. We had seen P2 feature map capable of increasing the mAP in previous section. If we look at the loss at validation set, we can see that `YOLO-MAR + head` is minimizing the localization loss more than `YOLO-MAR + rerouting`. There are several reasons we thought of why this happened:

1. The network was simply producing loose bounding boxes prediction which looks better in loss function but fail when threshold criteria like in $mAP@50$ is introduced.
2. This is an effect of bad choice of hyperparameters.

4.6 Decoupled Anchor-free Head

We experimented with replacing the coupled YOLO head to of a decoupled anchor-free head. Upon replacing the head, we also replaced the label assigner from SimOTA to TAL. The performance of this modification is shown in Table 4.6.

The model was initially unable to converge due to numerical instability caused by having the objects becoming too small upon downscaling. After stabilizing the loss function to handle these cases as shown in B.3, this modification was able to improve the mAP score by 21.78%.

4.7 Partitioning Image

We trained the models using partitioned image of the original dataset. In inference, we expect to partition an image into 4 equal sized images, which will then be fed to the neural network. This way, the network will produce 4 independent output that we can combine to form the final inference.

Table 4.6: Anchor-free Head Performance

No	Model	mAP@50
0	YOLOv7-MAR (anchor head)	11.2%
1	YOLOv7-MAR + anchor-free head	0%
1	**YOLOv7-MAR + anchor-free head** *numerically stabilized	**32.98%**
	Improvement	**+21.78%**

Table 4.7: Partitioning Image Performance

No	Model	Input Size	Partition4 mAP@50
0	YOLOv7-AR	960	2.9%
1	YOLOv7-MAR	640	18.46%
3	YOLOv7-MAR	960	37.69%
4	YOLOv7-MAR + rerouting	960	30.04%
5	YOLOv7-MAR + more head	960	10.53%
6	YOLOv7-M + anchor-free	640	37.57%
7	**YOLOv7-M + anchor-free**	960	**46.18%**

To generate partitioned data for training, we cropped image with size of $(W/2, H/2)$ based on the location of the objects bounding boxes. Therefore, it can be guaranteed that each image has atleast an object except for negative class images.

Since we are partitioning the image to a smaller size, we can finally use a smaller input size to the neural network. We experimented by using 640 and 960 input size on some promising models.

The performace is shown on Table 4.7. YOLOv7-MAR + anchor-free with input size 960 produced the greatest mAP@50 score amongs other model.

4.8 Latency

In this section, the latency of the modifications is presented. The data obtained using computer with specification presented in section 3.3. Table 4.8 shows that all the partition 4 models have > 40 FPS inference speed. This means that they are able to perform at > 10 FPS even without batching to detect a full image, making them all valid candidate for the best model.

Table 4.8: Inference Speed of Modified Models

No	Model	Input Size	Inference Speed	YOLOR Reparameterized
0	YOLOv7-MAR, plain, M, AR, EIoU	1600	29.6 FPS	29.6 FPS
1	YOLOv7-MAR + rerouting	1600	23.28 FPS	23.02 FPS
2	YOLOv7-MAR + more head	1600	15.91 FPS	16.09 FPS
3	YOLOv7-MAR + anchor-free	1600	26.42 FPS	26.01 FPS
4	YOLOv7-MAR	Partition 4, 960	72.45 FPS	73.75 FPS
5	YOLOv7-MAR + rerouting	Partition 4, 960	60.35 FPS	61.64 FPS
6	YOLOv7-MAR + more head	Partition 4, 960	40.7 FPS	41.8 FPS
7	YOLOv7-M + anchor-free	Partition 4, 960	63.17 FPS	64.2 FPS
8	YOLOv7-MAR	Partition 4, 640	69.75 FPS	71.78 FPS
9	YOLOv7-MAR + rerouting	Partition 4, 640	68.61 FPS	72.73 FPS
10	YOLOv7-MAR + more head	Partition 4, 640	57.40 FPS	59.47 FPS
11	YOLOv7-M + anchor-free	Partition 4, 640	65.29 FPS	65.34 FPS

[This page intentionally left blank]

CHAPTER V
CONCLUSION

5.1 Conclusion

Based on the experiments conducted in previous chapter, it can be observed that some modification can improve YOLOv7 ability on detecting airborne objects, while others have negligible or even negative effect. We found that:

- YOLOv7 was only able to perform detection on airborne objects after adding mosaic augmentation to the training data and recalculated its anchor to fit the data. This modification improved the mAP score from 0 to 11.2%.
- EIoU, even with its convexication technique, was not able to outperform YOLOv7's original CIoU. It produced only 4.92% in mAP score.
- Rerouting the detection to use an earlier stage of feature map can improve slightly improve the mAP.
- Additional head on YOLOv7 did not perform better than original 3 head.
- Replacing head to a decoupled anchor free head greatly improve the detection capability.
- Applying image partition to all model greatly improve their detection performance. The greatest performance after applying image partition came from anchor-free head modification which produced 46.18% mAP, a 13.2% improvement compared to without partition.

In conclusion, amongst the proposed modifications applied to YOLOv7, we found that applying image partition, and replacing head to a decoupled anchor free head performed the best on detecting airborne objects.

[This page intentionally left blank]

REFERENCES

Aditya, N., Indaryo, A., Solang, D., Santiung, M., Atmaja, H., Azis, A., Januar, F., Javanica, F., Kautaman, G., Kazaksti, E., Nugraha, D., Permadani, R., Ramadhan, R., Ramadhan, R., Damayanti, Z., Valentia, M., Sundana, P., Shodiq, F., Waisnawa, R., … Dikairono, R. (2022). Roboboat 2022: Technical design report Barunastra ITS roboboat team.

Airborne object tracking challenge. (2021). Retrieved September 1, 2022, from https://www.aicrowd.com/challenges/airborne-object-tracking-challenge

Airborne object tracking dataset. (2021). Retrieved September 1, 2022, from https://registry.opendata.aws/airborne-object-tracking

Amazon. (2022). *Amazon prime air prepares for drone deliveries*. Retrieved September 1, 2022, from https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries

Benjumea, A., Teeti, I., Cuzzolin, F., & Bradley, A. (2021). Yolo-z: Improving small object detection in yolov5 for autonomous vehicles.

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection.

Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). Yolox: Exceeding yolo series in 2021.

Jocher, G. (2020). *by Ultralytics* (Version 7.0). https://doi.org/10.5281/zenodo.3908559

Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., Michael, K., TaoXie, Fang, J., imyhxy, & et al. (2022). Ultralytics/yolov5: V7.0 - yolov5 sota realtime instance segmentation. https://doi.org/10.5281/zenodo.7347926

Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., Li, Y., Zhang, B., Liang, Y., Zhou, L., Xu, X., Chu, X., Wei, X., & Wei, X. (2022). Yolov6: A single-stage object detection framework for industrial applications.

Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L., & da Silva, E. A. B. (2021). A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, *10*(3). https://doi.org/10.3390/electronics10030279

Peng, H., & Yu, S. (2021). A systematic iou-related method: Beyond simplified regression for better localization. *CoRR*, *abs/2112.01793*. https://arxiv.org/abs/2112.01793

Redmon, J., & Farhadi, A. (2016). Yolo9000: Better, faster, stronger.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement.

Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression.

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2020). Scaled-YOLOv4: Scaling cross stage partial network.

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.

Wang, C.-Y., Liao, H.-Y. M., & Yeh, I.-H. (2022). Designing network design strategies through gradient path analysis.

Wang, C.-Y., Liao, H.-Y. M., Yeh, I.-H., Wu, Y.-H., Chen, P.-Y., & Hsieh, J.-W. (2019). Cspnet: A new backbone that can enhance learning capability of cnn.

Wang, C.-Y., Yeh, I.-H., & Liao, H.-Y. M. (2021). You only learn one representation: Unified network for multiple tasks.

Wang, P., Xiao, J., Kawaguchi, K., & Lichen, W. (2022). Automatic ceiling damage detection in large-span structures based on computer vision and deep learning. *Sustainability*, *14*, 3275. https://doi.org/10.3390/su14063275

Xiao, J. (2021). exYOLO: A small object detector based on YOLOv3 object detector. *Procedia Computer Science*, *188*, 18–25. https://doi.org/10.1016/j.procs.2021.05.048

Zhang, Z., Lu, X., Cao, G., Yang, Y., Jiao, L., & Liu, F. (2021). Vit-yolo:transformer-based yolo for object detection. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2799–2808. https://doi.org/10.1109/ICCVW54120.2021.00314

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., & Ren, D. (2019). Distance-iou loss: Faster and better learning for bounding box regression.

Zhong, Y., Wang, J., Peng, J., & Zhang, L. (2018). Anchor box optimization for object detection.

APPENDIX I

A.1 YOLOv7 + more head Architecture Configuration

```
# parameters
nc: 4  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple

# anchors
anchors:
  #- [12,16, 19,36, 40,28]  # P3/8
  #- [36,75, 76,55, 72,146]  # P4/16
  #- [142,110, 192,243, 459,401]  # P5/32
  - [4,4,  10,10,  23,8]
  - [16,15,  41,11,  59,23]
  - [46,34,  47,41,  85,29]
  - [114,27,  112,62,  272,71]

# yolov7 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [32, 3, 1]],  # 0

    [-1, 1, Conv, [64, 3, 2]],  # 1-P1/2
    [-1, 1, Conv, [64, 3, 1]],

    [-1, 1, Conv, [128, 3, 2]],  # 3-P2/4
    [-1, 1, Conv, [64, 1, 1]],
    [-2, 1, Conv, [64, 1, 1]],
    [-1, 1, Conv, [64, 3, 1]],
    [-1, 1, Conv, [64, 3, 1]],
    [-1, 1, Conv, [64, 3, 1]],
    [-1, 1, Conv, [64, 3, 1]],
    [[-1, -3, -5, -6], 1, Concat, [1]],
    [-1, 1, Conv, [256, 1, 1]],  # 11

    [-1, 1, MP, []],
    [-1, 1, Conv, [128, 1, 1]],
    [-3, 1, Conv, [128, 1, 1]],
    [-1, 1, Conv, [128, 3, 2]],
    [[-1, -3], 1, Concat, [1]],  # 16-P3/8
    [-1, 1, Conv, [128, 1, 1]],
    [-2, 1, Conv, [128, 1, 1]],
    [-1, 1, Conv, [128, 3, 1]],
    [-1, 1, Conv, [128, 3, 1]],
    [-1, 1, Conv, [128, 3, 1]],
    [-1, 1, Conv, [128, 3, 1]],
    [[-1, -3, -5, -6], 1, Concat, [1]],
    [-1, 1, Conv, [512, 1, 1]],  # 24

    [-1, 1, MP, []],
    [-1, 1, Conv, [256, 1, 1]],
```

```
   [-3, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 29-P4/16
   [-1, 1, Conv, [256, 1, 1]],
   [-2, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [1024, 1, 1]],  # 37

   [-1, 1, MP, []],
   [-1, 1, Conv, [512, 1, 1]],
   [-3, 1, Conv, [512, 1, 1]],
   [-1, 1, Conv, [512, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 42-P5/32
   [-1, 1, Conv, [256, 1, 1]],
   [-2, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [1024, 1, 1]],  # 50
  ]

# yolov7 head
head:
  [[-1, 1, SPPCSPC, [512]], # 51

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [37, 1, Conv, [256, 1, 1]], # route backbone P4
   [[-1, -2], 1, Concat, [1]],

   [-1, 1, Conv, [256, 1, 1]],
   [-2, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1]], # 63

   [-1, 1, Conv, [128, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [24, 1, Conv, [128, 1, 1]], # route backbone P3
   [[-1, -2], 1, Concat, [1]],

   [-1, 1, Conv, [128, 1, 1]],
   [-2, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
```

```
   [-1, 1, Conv, [128, 1, 1]], # 75

   ################## RECALCULATE INDEX FROM HERE ##################
   [-1, 1, Conv, [64, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [11, 1, Conv, [64, 1, 1]], # route backbone P2
   [[-1, -2], 1, Concat, [1]], # 79

   [-1, 1, Conv, [64, 1, 1]],
   [-2, 1, Conv, [64, 1, 1]],
   [-1, 1, Conv, [32, 3, 1]],
   [-1, 1, Conv, [32, 3, 1]],
   [-1, 1, Conv, [32, 3, 1]],
   [-1, 1, Conv, [32, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [64, 1, 1]], # 87 HEAD 1

   [-1, 1, MP, []],
   [-1, 1, Conv, [64, 1, 1]],
   [-3, 1, Conv, [64, 1, 1]],
   [-1, 1, Conv, [64, 3, 2]],
   [[-1, -3, 75], 1, Concat, [1]], # 92

   [-1, 1, Conv, [128, 1, 1]],
   [-2, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [128, 1, 1]], # 100 HEAD 2



   [-1, 1, MP, []],
   [-1, 1, Conv, [128, 1, 1]],
   [-3, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [128, 3, 2]],
   [[-1, -3, 63], 1, Concat, [1]], #105

   [-1, 1, Conv, [256, 1, 1]],
   [-2, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1]], # 113 HEAD 3

   [-1, 1, MP, []],
   [-1, 1, Conv, [256, 1, 1]],
   [-3, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, -3, 51], 1, Concat, [1]], # 118

   [-1, 1, Conv, [512, 1, 1]],
   [-2, 1, Conv, [512, 1, 1]],
```

```
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [512, 1, 1]], # 126 HEAD 4

   #[75, 1, RepConv, [256, 3, 1]],
   #[88, 1, RepConv, [512, 3, 1]],
   #[101, 1, RepConv, [1024, 3, 1]],

   [87, 1, RepConv, [128, 3,1]], # 127
   [100, 1, RepConv, [256, 3, 1]],
   [113, 1, RepConv, [512, 3, 1]],
   [126, 1, RepConv, [1024, 3, 1]],


   [[127,128,129,130], 1, IDetect, [nc, anchors]],   # Detect(P3, P4, P5)
  ]
```

A.2 YOLOv7 + rerouting Architecture Configuration

```
# parameters
nc: 4  # number of classes
depth_multiple: 1.0  # model depth multiple
width_multiple: 1.0  # layer channel multiple

# anchors
anchors:
  #- [12,16, 19,36, 40,28]  # P3/8
  #- [36,75, 76,55, 72,146]  # P4/16
  #- [142,110, 192,243, 459,401]  # P5/32
  - [4,4,  14,5,  11,11]  # P3/8
  - [27,8,  18,17,  41,14]  # P4/16
  - [43,32,  86,27,  149,70]  # P5/32

# yolov7 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [32, 3, 1]],  # 0

   [-1, 1, Conv, [64, 3, 2]],  # 1-P1/2
   [-1, 1, Conv, [64, 3, 1]],

   [-1, 1, Conv, [128, 3, 2]],  # 3-P2/4
   [-1, 1, Conv, [64, 1, 1]],
   [-2, 1, Conv, [64, 1, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1]],  # 11

   [-1, 1, MP, []],
   [-1, 1, Conv, [128, 1, 1]],
```

```
   [-3, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [128, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 16-P3/8
   [-1, 1, Conv, [128, 1, 1]],
   [-2, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [512, 1, 1]],  # 24

   [-1, 1, MP, []],
   [-1, 1, Conv, [256, 1, 1]],
   [-3, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 29-P4/16
   [-1, 1, Conv, [256, 1, 1]],
   [-2, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [1024, 1, 1]],  # 37

   [-1, 1, MP, []],
   [-1, 1, Conv, [512, 1, 1]],
   [-3, 1, Conv, [512, 1, 1]],
   [-1, 1, Conv, [512, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 42-P5/32
   [-1, 1, Conv, [256, 1, 1]],
   [-2, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [-1, 1, Conv, [256, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [1024, 1, 1]],  # 50
  ]

# yolov7 head
head:
  [[-1, 1, SPPCSPC, [512]], # 51

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [37, 1, Conv, [256, 1, 1]], # route backbone P4
   [[-1, -2], 1, Concat, [1]],

   [-1, 1, Conv, [256, 1, 1]],
   [-2, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
```

```
  [-1, 1, Conv, [256, 1, 1]], # 63

  [-1, 1, Conv, [128, 1, 1]],
  [-1, 1, nn.Upsample, [None, 4, 'nearest']],
  [11, 1, Conv, [128, 1, 1]], # route backbone P3
  [[-1, -2], 1, Concat, [1]],

  [-1, 1, Conv, [128, 1, 1]],
  [-2, 1, Conv, [128, 1, 1]],
  [-1, 1, Conv, [64, 3, 1]],
  [-1, 1, Conv, [64, 3, 1]],
  [-1, 1, Conv, [64, 3, 1]],
  [-1, 1, Conv, [64, 3, 1]],
  [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
  [-1, 1, Conv, [128, 1, 1]], # 75

  # RECALCULATE FROM HERE
  # downsample after use in 75
  [-1, 1, MP, []],
  [-1, 1, Conv, [128, 1, 2]],
  [-3, 1, Conv, [128, 1, 1]],
  [-1, 1, Conv, [128, 3, 4]],
  [[-1, -3, 63], 1, Concat, [1]],  # 80

  [-1, 1, Conv, [256, 1, 1]],
  [-2, 1, Conv, [256, 1, 1]],
  [-1, 1, Conv, [128, 3, 1]],
  [-1, 1, Conv, [128, 3, 1]],
  [-1, 1, Conv, [128, 3, 1]],
  [-1, 1, Conv, [128, 3, 1]],
  [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
  [-1, 1, Conv, [256, 1, 1]], # 88

  [-1, 1, MP, []],
  [-1, 1, Conv, [256, 1, 1]],
  [-3, 1, Conv, [256, 1, 1]],
  [-1, 1, Conv, [256, 3, 2]],
  [[-1, -3, 51], 1, Concat, [1]],

  [-1, 1, Conv, [512, 1, 1]],
  [-2, 1, Conv, [512, 1, 1]],
  [-1, 1, Conv, [256, 3, 1]],
  [-1, 1, Conv, [256, 3, 1]],
  [-1, 1, Conv, [256, 3, 1]],
  [-1, 1, Conv, [256, 3, 1]],
  [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
  [-1, 1, Conv, [512, 1, 1]], # 101

  [75, 1, RepConv, [256, 3, 1]],
  [88, 1, RepConv, [512, 3, 1]],
  [101, 1, RepConv, [1024, 3, 1]],

  [[102,103,104], 1, IDetect, [nc, anchors]],   # Detect(P3, P4, P5)
 ]
```

APPENDIX II

B.1 Compute EIoU

```
def bbox_iou(box1, box2, x1y1x2y2=True, eps=1e-7):
    # Returns the IoU of box1 to box2. box1 is 4, box2 is nx4
    box2 = box2.T

    # Get the coordinates of bounding boxes
    if x1y1x2y2:  # x1, y1, x2, y2 = box1
        b1_x1, b1_y1, b1_x2, b1_y2 = box1[0], box1[1], box1[2], box1[3]
        b2_x1, b2_y1, b2_x2, b2_y2 = box2[0], box2[1], box2[2], box2[3]
    else:  # transform from xywh to xyxy
        b1_x1, b1_x2 = box1[0] - box1[2] / 2, box1[0] + box1[2] / 2
        b1_y1, b1_y2 = box1[1] - box1[3] / 2, box1[1] + box1[3] / 2
        b2_x1, b2_x2 = box2[0] - box2[2] / 2, box2[0] + box2[2] / 2
        b2_y1, b2_y2 = box2[1] - box2[3] / 2, box2[1] + box2[3] / 2


    w1, h1 = b1_x2 - b1_x1, b1_y2 - b1_y1 + eps
    w2, h2 = b2_x2 - b2_x1, b2_y2 - b2_y1 + eps

    x0 = torch.min(b1_x1,b2_x1) # topleftest of top left of the boxes
    y0 = torch.min(b1_y1,b2_y1)
    x1 = torch.max(b1_x1,b2_x1) # bottomrightest of toplefts
    y1 = torch.max(b1_y1,b2_y1)
    x2 = torch.min(b1_x2,b2_x2) # topleftest of bottomrights
    y2 = torch.min(b1_y2,b2_y2)

    xmin = torch.min(x1,x2)
    ymin = torch.min(y1,y2)
    xmax = torch.max(x1,x2)
    ymax = torch.max(y1,y2)

    inter = (x2-x0)*(y2-y0) + \
            (xmin-x0)*(ymin-y0) - \
            (x1-x0)*(ymax-y0) - \
            (xmax-x0)*(y1-y0)

    union = w1 * h1 + w2 * h2 - inter + eps
    return inter/union # using this just for covexication
    #return torch.min(inter/union,torch.Tensor([1.0]).cuda()) # use this ↩
       return for EIoU without covexication
```

B.2 EIoU Loss

```
class ComputeLossOTA_EIoU(ComputeLossOTA):
    def __call__(self, p, targets, imgs):  # predictions, targets, model
        device = targets.device
        lcls, lbox, lobj = torch.zeros(1, device=device), torch.zeros(1, ↩
           device=device), torch.zeros(1, device=device)
```

```
        bs, as_, gjs, gis, targets, anchors = self.build_targets(p, targets, imgs)
        pre_gen_gains = [torch.tensor(pp.shape, device=device)[[3, 2, 3, 2]] for pp in p]


        # Losses
        for i, pi in enumerate(p):  # layer index, layer predictions
            b, a, gj, gi = bs[i], as_[i], gjs[i], gis[i]  # image, anchor, gridy, gridx
            tobj = torch.zeros_like(pi[..., 0], device=device)  # target obj

            n = b.shape[0]  # number of targets
            if n:
                ps = pi[b, a, gj, gi]  # prediction subset corresponding to targets

                # Regression
                grid = torch.stack([gi, gj], dim=1)
                pxy = ps[:, :2].sigmoid() * 2. - 0.5
                #pxy = ps[:, :2].sigmoid() * 3. - 1.
                pwh = (ps[:, 2:4].sigmoid() * 2) ** 2 * anchors[i]
                pbox = torch.cat((pxy, pwh), 1)  # predicted box
                selected_tbox = targets[i][:, 2:6] * pre_gen_gains[i]
                selected_tbox[:, :2] -= grid
                iou = bbox_iou(pbox.T, selected_tbox, x1y1x2y2=False, EIoU=True)  # iou(prediction, target)
                lbox += ((1.0 - iou)**2).mean()  # iou loss # using square for covexication

                # Objectness
                tobj[b, a, gj, gi] = (1.0 - self.gr) + self.gr * iou.detach().clamp(0).type(tobj.dtype)  # iou ratio

                # Classification
                selected_tcls = targets[i][:, 1].long()
                if self.nc > 1:  # cls loss (only if multiple classes)
                    t = torch.full_like(ps[:, 5:], self.cn, device=device)  # targets
                    t[range(n), selected_tcls] = self.cp
                    lcls += self.BCEcls(ps[:, 5:], t)  # BCE

                # Append targets to text file
                # with open('targets.txt', 'a') as file:
                #     [file.write('%11.5g ' * 4 % tuple(x) + '\n') for x in torch.cat((txy[i], twh[i]), 1)]

            obji = self.BCEobj(pi[..., 4], tobj)
            lobj += obji * self.balance[i]  # obj loss
            if self.autobalance:
                self.balance[i] = self.balance[i] * 0.9999 + 0.0001 / obji.detach().item()

        if self.autobalance:
            self.balance = [x / self.balance[self.ssi] for x in self.balance]
```

```
        lbox *= self.hyp['box']
        lobj *= self.hyp['obj']
        lcls *= self.hyp['cls']
        bs = tobj.shape[0]  # batch size

        loss = lbox + lobj + lcls
        return loss * bs, torch.cat((lbox, lobj, lcls, loss)).detach()
```

B.3 Anchor-free Loss Numerical Stabilization

```
class ComputeLossStabilized(ComputeLoss):
    def __call__(self, p, targets, img=None, epoch=0):
        loss = torch.zeros(3, device=self.device)  # box, cls, dfl
        feats, pred_distri, pred_scores = p
        pred_scores = pred_scores.permute(0, 2, 1).contiguous()
        pred_distri = pred_distri.permute(0, 2, 1).contiguous()

        dtype = pred_scores.dtype
        batch_size, grid_size = pred_scores.shape[:2]
        imgsz = torch.tensor(feats[0].shape[2:], device=self.device, ↩
            dtype=dtype) * self.stride[0]  # image size (h,w)
        anchor_points, stride_tensor = make_anchors(feats, self.stride, ↩
            0.5)

        # targets
        targets = self.preprocess(targets, batch_size, scale_tensor=imgsz↩
            [[1, 0, 1, 0]])
        gt_labels, gt_bboxes = targets.split((1, 4), 2)  # cls, xyxy
        mask_gt = gt_bboxes.sum(2, keepdim=True).gt_(0)

        # pboxes
        pred_bboxes = self.bbox_decode(anchor_points, pred_distri)  # ↩
            xyxy, (b, h*w, 4)

        target_labels, target_bboxes, target_scores, fg_mask = self.↩
            assigner(
             pred_scores.detach().sigmoid(),
             (pred_bboxes.detach() * stride_tensor).type(gt_bboxes.dtype),
             anchor_points * stride_tensor,
             gt_labels,
             gt_bboxes,
             mask_gt)

        target_bboxes /= stride_tensor
        target_scores_sum = target_scores.sum()
        target_scores_sum = max(1,target_scores_sum) # numerical ↩
            stabilization
        # cls loss
        # loss[1] = self.varifocal_loss(pred_scores, target_scores, ↩
            target_labels) / target_scores_sum  # VFL way
        loss[1] = self.BCEcls(pred_scores, target_scores.to(dtype)).sum()↩
             / (target_scores_sum)  # BCE

        # bbox loss
        if fg_mask.sum():
            loss[0], loss[2], iou = self.bbox_loss(pred_distri,
```

```
                                                pred_bboxes,
                                                anchor_points,
                                                target_bboxes,
                                                target_scores,
                                                target_scores_sum,
                                                fg_mask)

        loss[0] *= 7.5  # box gain
        loss[1] *= 0.5  # cls gain
        loss[2] *= 1.5  # dfl gain

        return loss.sum() * batch_size, loss.detach()  # loss(box, cls, ↩
            dfl)
```

APPENDIX III

[This page intentionally left blank]

AUTOBIOGRAPHY

Dion Andreas Solang, born on the day he was born. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

[This page intentionally left blank]