

# **IMPROVING THE PERFORMANCE OF YOLO-BASED DETECTION ALGORITHMS FOR SMALL OBJECT DETECTION IN UAV-TAKEN IMAGES**

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE  
OF BILKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR  
THE DEGREE OF  
MASTER OF SCIENCE  
IN  
COMPUTER ENGINEERING

By  
Oyku Sahin  
January 2023

Improving the Performance of YOLO-based Detection Algorithms for  
Small Object Detection in UAV-taken Images

By Oyku Sahin

January 2023

We certify that we have read this thesis and that in our opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

---

İbrahim Körpeoğlu(Advisor)

---

Sedat Özer(Co-Advisor)

---

Özgür Ulusoy

---

M. Furkan Kırac

Approved for the Graduate School of Engineering and Science:

---

Orhan Arıkan  
Director of the Graduate School



## ABSTRACT

# IMPROVING THE PERFORMANCE OF YOLO-BASED DETECTION ALGORITHMS FOR SMALL OBJECT DETECTION IN UAV-TAKEN IMAGES

Oyku Sahin

M.S. in Computer Engineering

Advisor: İbrahim Körpeoğlu

Co-Advisor: Sedat Özer

January 2023

Recent advances in computer vision yield emerging novel applications for camera-equipped unmanned aerial vehicles such as object detection. The accuracy of the existing object detection solutions running on images acquired by Unmanned Aerial Vehicles (UAVs) is limited when compared to the performance of the object detection solutions designed for ground-taken images. Existing object detection solutions demonstrate lower performance on aerial datasets because of the reasons originating from the nature of the UAVs. These reasons can be summarized as: (i) the lack of large drone datasets with different types of objects, (ii) the larger variance in both scale and orientation of objects in drone images, and (iii) the difference in shape and texture of the features between the ground and the aerial images. Due to these reasons, YOLO-based models, a popular family of one-stage object detectors, perform insufficiently in UAV-based applications. In this thesis, two improved YOLO models: YOLODrone and YOLODrone+ are introduced for detecting objects in drone images. The performance of the models are tested on VisDrone2019 and SkyDataV1 datasets and improved results are reported when compared to the original YOLOv3 and YOLOv5 models.

*Keywords:* Object Detection, Drone Vision, Deep Neural Networks, YOLO.

## ÖZET

# KÜÇÜK NESNE TANIMA ÜZERİNE KULLANILAN YOLO TABANLI NESNE TANIMA ALGORİTMALARININ İYİLEŞTİRİLMESİ

Oyku Sahin

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Danışmanı: İbrahim Körpeoğlu

İkinci Tez Danışmanı: Sedat Özer

Ocak 2023

Bilgisayarlı görünün önemli alanlarından biri olan nesne tanıma alanındaki son gelişmeler, kamera donanımlı İnsansız Hava Araçları (İHA) için ortaya çıkan yeni çözümler sağlar. İnsansız Hava Araçları'ndan alınan görüntüler üzerinde çalışan, nesne tanıma çözümlerinin başarımı, yerden alınan görüntüler için nesne tanıma çözümlerinin performansı ile karşılaştırıldığında daha sınırlıdır. Mevcut nesne tanıma çözümleri, İHA'ların doğasından kaynaklanan nedenlerle İHA veri kümelerinde daha düşük performans göstermektedir. Bu nedenler: (i) farklı nesne boyutlarına sahip büyük drone veri setlerinin olmaması, (ii) drone görüntülerinin hem ölçek hem de yönelim olarak daha büyük varyanslara sahip olması (iii) yer ve hava arasındaki şekil ve doku özelliklerindeki farklılık. Bu sebeplerden dolayı, tek kademeli nesne tanıma yöntemlerinin bir parçası olan YOLO tabanlı modeller, İHA tabanlı uygulamalarda daha düşük performans göstermektedir. Bu tezde, YOLO algoritması üzerine YOLODrone ve YOLODrone+, İHA görüntülerindeki nesneleri tanımak için geliştirilmiştir. Modellerin performansı VisDrone2019 ve SkyDataV1 veri kümelerinde test edilip; YOLOv3 ve YOLOv5 ile karşılaştırıldığında sonuçlarda iyileştirme gözlemlenmiştir.

*Anahtar sözcükler:* Nesne Tanıma, İHA Görüntüsü, Derin Sinir Ağları, YOLO.

## Acknowledgement

First, I would like to express my deepest gratitude and thanks to Dr. Sedat Özer. The completion of this thesis would never be possible without his advanced supervision, exceptional tolerance and illuminating guidance.

Secondly, I would like to express my special thanks to Prof. Dr. İbrahim Körpeoğlu, who helped and guided me over the last three years. I would like to thank Prof. Dr. Özgür Ulusoy and Dr. Furkan Kırac for being on my thesis committee.

Also, I would also like to extend my gratitude to my grandparents for encouraging me to be whoever I want to be and showing their love and support in any circumstances Enis-Meral Şahin and Kadir-Özcan Okutan.

Finally, I would also like to express my gratitude to my incredible classmates Arda Tümay, M. Utku Şahin and Hasan Şerefal for their technical support and insightful discussions.



To my beloved parents  
Serap Şahin  
and  
Akın Şahin

# Contents

|          |                                                    |           |
|----------|----------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                                | <b>1</b>  |
| 1.1      | Motivation . . . . .                               | 2         |
| 1.2      | Research Aims and Objectives . . . . .             | 3         |
| 1.3      | Scope of this Thesis . . . . .                     | 4         |
| 1.4      | Structure of This Thesis . . . . .                 | 5         |
| <br>     |                                                    |           |
| <b>2</b> | <b>Literature Review</b>                           | <b>6</b>  |
| 2.1      | Non-Deep Learning Based Object Detection . . . . . | 8         |
| 2.2      | Deep Learning Based Object Detection . . . . .     | 9         |
| 2.2.1    | YOLO Family . . . . .                              | 14        |
| 2.3      | Object Detection Datasets . . . . .                | 19        |
| <br>     |                                                    |           |
| <b>3</b> | <b>Proposed Models: YOLODrone &amp; YOLODrone+</b> | <b>23</b> |
| 3.1      | YOLODrone . . . . .                                | 25        |
| 3.1.1    | YOLOv3 . . . . .                                   | 25        |

|          |                                                        |           |
|----------|--------------------------------------------------------|-----------|
| 3.1.2    | YOLODrone Architecture . . . . .                       | 28        |
| 3.2      | YOLODrone+ . . . . .                                   | 31        |
| 3.2.1    | Related Background: YOLOv5 . . . . .                   | 31        |
| 3.2.2    | Related Background: Vision Transformers . . . . .      | 36        |
| 3.2.3    | YOLODrone+ Architecture . . . . .                      | 37        |
| <b>4</b> | <b>Experiments</b>                                     | <b>39</b> |
| 4.1      | Object Detection Evaluation Metrics . . . . .          | 40        |
| 4.2      | Datasets and Data Augmentation . . . . .               | 43        |
| 4.3      | Experimental Methodology . . . . .                     | 46        |
| 4.3.1    | The Effect of Using Data Augmentation . . . . .        | 46        |
| 4.3.2    | The Effect of Number of Anchor Boxes . . . . .         | 47        |
| 4.3.3    | Varying the Total Number of Detection Layers . . . . . | 48        |
| 4.3.4    | The Effect of Adding Vision Transformers . . . . .     | 53        |
| <b>5</b> | <b>Discussion and Concluding Marks</b>                 | <b>57</b> |
| <b>A</b> | <b>Object Detection Datasets</b>                       | <b>59</b> |
| A.1      | Datasets That Are Not Obtained by Drones . . . . .     | 60        |
| A.2      | Datasets That Are Obtained by Drones . . . . .         | 64        |
| <b>B</b> | <b>Data Labeling</b>                                   | <b>68</b> |

|                                           |           |
|-------------------------------------------|-----------|
| B.1 SkyData Dataset . . . . .             | 69        |
| B.2 CVAT . . . . .                        | 72        |
| <b>C Multi-Object Tracking</b>            | <b>77</b> |
| C.1 Deep Learning Embedded MOT . . . . .  | 78        |
| C.2 Tracker Algorithm: DeepSort . . . . . | 80        |
| C.3 HOTA Metrics . . . . .                | 82        |
| C.4 Tracking Results . . . . .            | 83        |

# List of Figures

|     |                                                                                                                                                                                                                                                                                                                                                              |    |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | Shows the timeline of object detection solutions that were explained in this chapter. The blue line represents the traditional object detection methods. After AlexNet [1] was discovered then deep learning based solutions were split into two. The green line represents two-stage detectors whereas, purple line represents one-stage detectors. . . . . | 7  |
| 2.2 | Essential parts of the object detection task are visualized step by step. An object (dog) was first classified, then localized and all that information is shown on the final step. The sample image is taken from the COCO [2] dataset. . . . .                                                                                                             | 10 |
| 2.3 | Running object detection for dog detection on a sample image from COCO [2]. The bounding boxes and the class name(s) are the output of object detection and they are visualized around the detected objects. . . . .                                                                                                                                         | 11 |
| 2.4 | The main stages that are generally used while designing either two-stage or single-stage object detection models. . . . .                                                                                                                                                                                                                                    | 13 |
| 2.5 | A sample figure to represent the idea of the grid cells. Purple point in the center of the cell is the center point (X, Y) of the bounding box to be proposed and the green point (x, y) is the coordinate of the grid cells' itself. The image taken from COCO dataset [2] . . .                                                                            | 15 |



|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |    |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.6 | Shows the data collection setup which was used to collect data for the KITTI dataset [3]. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                | 21 |
| 2.7 | Sample images from VisDrone dataset. The recording time as well as angle and altitude of the drone is varied in this dataset as can be seen from the samples. . . . .                                                                                                                                                                                                                                                                                                                                            | 22 |
| 3.1 | Shows the number of papers published related to “object detection” over each year, data used in the graph was obtained from Google Scholar. . . . .                                                                                                                                                                                                                                                                                                                                                              | 24 |
| 3.2 | Shows various scales of the anchor boxes in YOLOv3. The values of these anchor boxes are calculated by the K-Means algorithm. There are three different sizes because the detection layers of YOLO split the input image into grid cells in three different sizes. In this figure, the sizes of the anchor boxes are defined by using MS COCO [2]. The following shows the (height, width) pairs in: (a) (10,13), (16,30), (33,23); (b) (30,61), (62,45), (59,119); (c) (116,90), (156,198), (373,326) . . . . . | 26 |
| 3.3 | Shows the original YOLOv3 architecture. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 27 |
| 3.4 | Two sample images demonstrate how the viewing angle and the object scales change an object’s appearance in an UAV-taken image. Both of the sample images are taken from the VisDrone2019 [4] dataset. . . . .                                                                                                                                                                                                                                                                                                    | 28 |
| 3.5 | Shows the architecture of YOLODrone. Different colored blocks in the architecture represent different blocks. The meaning of each color is given below the architectures. YOLODrone architecture has total of five detection layer . . . . .                                                                                                                                                                                                                                                                     | 29 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |    |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.6  | DenseNet is a type of convolutional neural network (CNN) designed to address the problem of vanishing gradients in deep learning. It does this by introducing connections between all of the layers in the network, allowing information to flow more easily between them and improving the flow of gradients during training as shown. . . . .                                                                                                                                                                                                                                                                                                                | 32 |
| 3.7  | DenseNet with the Cross Stage Partial part added which makes CSPDenseNet . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 33 |
| 3.8  | Modified SPP divides the features along the depth dimension and applies SPP on each divided part. Then combines them again to create an output feature map. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 34 |
| 3.9  | Shows YOLOv5 architecture with all blocks are given in color-coded. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 35 |
| 3.10 | The visual representation of the vision transformers. Instead of computing relationships among all pixels in an image, Vision Transformers uses small sections of the image such as e.g., 3x3 pixels in this figure and computes the relationships among the pixels in these sections. The sections of the sample sections of the image are placed in a sequence and given position embeddings, that are learnable vectors. The sections are arranged into a linear sequence and multiplied by an embedded matrix. The resulting sequence, along with the position embedding, is fed into a transformer for further processing as the figure suggests. . . . . | 37 |
| 3.11 | Shows YOLODrone+ architecture in a color-coded form. Not only are the number of detection layers different from YOLOv5 but also, the Vision Transformers were added to the neck part of the model. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 38 |
| 4.1  | Show the visual representation of the intersection over union for two bounding boxes where one represents the prediction and the other represents the ground truth. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 41 |

|     |                                                                                                                                                                                                                                                                                                               |    |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 4.2 | Shows the sample outputs from various data augmentation techniques. . . . .                                                                                                                                                                                                                                   | 46 |
| 4.3 | Shows a sample image from VisDrone dataset. The black regions represent the ignored regions. . . . .                                                                                                                                                                                                          | 47 |
| 4.4 | Shows outputs from the proposed model. In Figure 4.4b red circles represent the objects that are missed by YOLOv3 and in Figure 4.4c green circles represent the objects that were successfully detected by YOLODrone model. . . . .                                                                          | 51 |
| 4.5 | Shows outputs from the proposed model. In Figure 4.5b red circles represent the objects that are missed by YOLOv3 and in Figure 4.5c green circles represent the objects that were successfully detected by YOLODrone model. . . . .                                                                          | 52 |
| 4.6 | Shows outputs from the proposed YOLODrone+ model. In Figure 4.6b red circles represent the objects that are missed by YOLOv5s and in Figure 4.6c green circles represent the objects that were successfully detected by YOLODrone+ model. . . . .                                                             | 56 |
| A.1 | Sample images from the COCO dataset [2]. As can be seen in both Figure A.1a and Figure A.1b the classes include most of the common objects including a tennis racket and the way the images are obtained vary. Therefore, the lighting and the angle is different as well as the size of the objects. . . . . | 60 |
| A.2 | Sample images from the PASCAL VOC [5]. Both Figure A.2a and Figure A.2b show different classes. . . . .                                                                                                                                                                                                       | 61 |
| A.3 | Sample images from the ILSVRC dataset [6]. The images are obtained from ImageNet however, the annotations are re-labeled for object detection task. . . . .                                                                                                                                                   | 61 |

|      |                                                                                                                                                                                                                                                                                                     |    |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| A.4  | Shows two sample images from KITTI [3] dataset. As also explained in the beginning of this section in order to obtain the KITTI dataset a car is used as shown in Figure 2.6. Therefore, the angle of the way the footage is obtained, is the same which provides an advantage in training. . . . . | 62 |
| A.5  | Shows two samples from MOT [7] dataset. Figure A.5a shows a recording during the night time and Figure A.5b shows the footage obtained during day time. . . . .                                                                                                                                     | 63 |
| A.6  | Figure A.6a and Figure A.6b UAV Detrac [8] shows two samples from UA-DETRAC dataset. These two figures represent the variant nature of the UAV datasets. Although most of the objects are similar in these figures, the difference in their angle makes the UAV datasets challenging. . . . .       | 64 |
| A.7  | Two sample images from the Stanford dataset [9] are shown. As it can be seen from these two samples, the recorded objects are quite small. . . . .                                                                                                                                                  | 65 |
| A.8  | Two samples from UAV123 [10] drone dataset. Figure A.8a is a car and Figure A.8b is a wake-board. This dataset is designed for single-image tracking. However, because of the variety in class, it can also be used for object detection. . . . .                                                   | 66 |
| A.9  | Sample figures from UAVDT dataset [11]. As can be seen from Figure 2.7 it has similar characteristics with VisDrone dataset. . . . .                                                                                                                                                                | 67 |
| A.10 | Shows two samples from an unique UAV dataset SeaDronesSee [12]. As can be seen from the given figures the objects are so small that they look similar therefore, requires very careful architectural design of deep learning models. . . . .                                                        | 67 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |    |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| B.1 | Raw and annotated sample images from the SkyData Challenge are visualized. The left-hand side shows the original images and the right-hand side shows the objects with their bounding boxes on the original image. Image source: <a href="http://www.skydatachallenge.com">www.skydatachallenge.com</a> .                                                                                                                                                                         | 71 |
| B.2 | The main screen after selecting the task and the job that needs to be annotated. . . . .                                                                                                                                                                                                                                                                                                                                                                                          | 73 |
| B.3 | The zoomed in version of the main screen. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                 | 74 |
| B.4 | The red box shows the number of the frame which is opened for annotation. . . . .                                                                                                                                                                                                                                                                                                                                                                                                 | 75 |
| B.5 | The red box shows the number of the frame which is opened for annotation. . . . .                                                                                                                                                                                                                                                                                                                                                                                                 | 75 |
| B.6 | The red box shows the number of the frame which is opened for annotation. . . . .                                                                                                                                                                                                                                                                                                                                                                                                 | 76 |
| C.1 | DeepSORT uses a deep learning-based object detection model to identify and locate objects in each frame of a video sequence. The tracking algorithm then uses the information provided by the object detection model, along with appearance and motion information, to associate the same object across different frames and maintain its identity. This allows DeepSORT to continue tracking objects even when they are temporarily occluded or leave the field of view. . . . . | 81 |
| C.2 | Shows the sample output of tracking with DeepSort as a tracking algorithm and YOLODrone+ as the object detector. Figure C.2c shows that DeepSort lost the track id for black car because other cars covered it. Then as Figure C.2d suggests the tracker finds the car again and gives it, its previous id which was 38 once again. . .                                                                                                                                           | 85 |

|                                                                                                                                                                                                                                                                                           |    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| C.3 Shows the sample output of tracking with DeepSort and YOLO-Drone+. As it can be seen from the given figures the tracker successfully tracks a person with id 20 even though both the drone and the target person are on the move and the scene is crowded with small objects. . . . . | 88 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|



# List of Tables

|     |                                                                                                                                                                                                                                                                                       |    |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.1 | Shows the details of YOLODrone architecture. . . . .                                                                                                                                                                                                                                  | 30 |
| 3.2 | Comparison of YOLOv3 to YOLODrone. . . . .                                                                                                                                                                                                                                            | 30 |
| 4.1 | Shows the results of YOLODrone on VisDrone2019 dataset with and without using the data augmentation. . . . .                                                                                                                                                                          | 47 |
| 4.2 | Shows results of using different numbers of anchor boxes in YOLOv3 Algorithm. . . . .                                                                                                                                                                                                 | 48 |
| 4.3 | Compare the test results of using 3 detection layers, 4 detection layers and 5 detection layers in the YOLO architecture which was trained on VisDrone2019 dataset. . . . .                                                                                                           | 49 |
| 4.4 | Comparison of YOLOv3 to YOLODrone on the visDrone2019 dataset (mAP values are shown for each class). . . . .                                                                                                                                                                          | 50 |
| 4.5 | Shows the names and the values of the hyperparameters used for training the models unless otherwise stated. . . . .                                                                                                                                                                   | 53 |
| 4.6 | In order to understand how the architectural structures perform on the dataset, both the accuracy score and the sensitivity score are shown for both SkyDataV1 and VisDrone datasets on all models. The last four columns provide class-based results for the sample classes. . . . . | 55 |

|     |                                                                                                                                                                                                                                                                                                |    |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 5.1 | Shows the performance comparison of the proposed models for conclusion. . . . .                                                                                                                                                                                                                | 58 |
| B.1 | Shows various details of the first version of the SkyData dataset. Source: <a href="http://www.skydatachallenge.com">www.skydatachallenge.com</a> . . . . .                                                                                                                                    | 69 |
| B.2 | This table shows additional information for each scene of the SkyData. each scene's name, its duration, altitude and the total number of frames of five scenes from the SkyData dataset are provided. Source: <a href="http://www.skydatachallenge.com">www.skydatachallenge.com</a> . . . . . | 70 |
| C.1 | Shows the tracking results of different detectors of DeepSort on HOTA metrics. . . . .                                                                                                                                                                                                         | 83 |



# Chapter 1

## Introduction

With the recent developments in hardware performance, the state-of-the-art deep learning based methods are providing satisfactory results that are used to solve many complex computer vision tasks such as object classification and recognition [1], semantic segmentation [13, 14], object detection [15] and object tracking [16, 17]. The input data can not only be rtsp streams but also be infrared images [18, 19]. Since they outperform the previous generation (classical) machine learning solutions that do not rely on deep learning based solutions, the deep learning methods are used commonly in many fields. It is commonly accepted that the deep learning based models extract representative features better than many classical methods because of their multiple processing layers, which are able to represent the input data with more than one level of abstraction. Thus, implicitly capturing and learning the small details of large-scale data, which can be any type of data, such as time series, image, sound or text. It is possible to extend the deep learning models according to our needs and, because of that, there is a rich family of methods and numbers of unsupervised and supervised learning models [20] that are available for different computer vision tasks.

With fast processing of the large-scale data and high prediction accuracy scores from deep learning methods, it is possible to even detect objects in a video. Object detection, essentially, is a computer vision problem where the goal is finding the location of the bounding box (a rectangular box that can be used to identify the location and size of the object) for each existing object of interest in a given image. Accessibility of the unmanned aerial vehicles and high resolution cameras gives the opportunity to record challenging frames in which the objects may appear at different angles and sizes. The use of cameras on aerial devices, such as drones, increased the demand for using reliable object detection algorithms. However, the detection performance of the existing detection algorithms is not close enough to the detection performance of the detection algorithms that run on ground taken (natural) images.

In this thesis, we introduce techniques to improve the performance of the well known YOLO object detection algorithms [21, 22] to detect the objects of interest in drone-taken images. We study different aspects of object detection and show their effect on detecting objects in drone-taken images. In particular, we study the effect of changing the number of prediction layers at the end of YOLO architecture, the effect of changing the anchor box sizes and detecting them with clustering, the effect of using transformers, the effect of data augmentation and adding vision transformers to the detection layer of the YOLO model. This thesis, particularly, focuses on improving the YOLO-based single stage object detectors' detection performance.

## 1.1 Motivation

Unmanned Aerial Vehicles (UAVs) give us the opportunity to take risks that people should not take during natural disasters such as wildfires. According to the United Nations Environment Program [23], the reason behind the increasing number of fires is climate change. Since the globe keeps warming, the researchers are expecting more wildfires globally. With the detection technology of UAVs, it will be possible to detect not only humans but also animals and then, rescue

them during wildfires. Finally, last but not least, with the proper use of these detection systems, the streets can be more secure for females, which, today, is still a concerning problem in various locations on the globe.

Today, almost all the UAVs are equipped with cameras and as the use of UAVs increases, the need for computer vision algorithms, performing accurate object detection also increases. This thesis aims to provide potential solutions to design better YOLO-based object detection algorithms in the motivation for inner-city surveillance.

## 1.2 Research Aims and Objectives

This thesis considers YOLOv3 and YOLOv5 implementations that are commonly used for object detection as of the concluding period of this thesis. YOLO as a solution to the object detection problem was originally introduced in 2016 [24] and since then, it has been used in many computer vision applications. Over the time, it has multiple iterations of the original YOLO version, by following the number convention such as: YOLO, YOLOv2, YOLOv3, YOLOv4, YOLOv5. This family of YOLO implementations mainly focused on improving the detection accuracy. There are other naming strategies followed as well, where the part that comes after the YOLO name, suggests what the algorithm is designed for. For example, YOLO-Lite [25] (as the name suggests it is designed to be faster and more efficient than the original YOLO algorithm. YOLO-Lite achieves this by simplifying the architecture of the network, which makes it possible to run the algorithm in real-time on low-power devices such as smartphones and Raspberry Pi.), a YOLO version that focuses on the computational aspects of running the algorithm and provides a smaller version with less numbers of parameters. However, the main goal in this thesis is to improve the detection accuracy in drone-taken images (drones and UAVs are considered essentially the same in this thesis.) All such YOLO originated algorithms are called YOLO-based algorithms, YOLO family or YOLO-derivatives in this thesis.

YOLO family is mainly introduced for ground taken images and they show satisfactory performance on many ground taken images. However, their performance is relatively poor when compared to the ground taken images, see [26]. This thesis studies the multiple reasons for yielding poor results in aerial images and then introduces improvements from the architectural aspects of the YOLO family for object detection in drone images.

### 1.3 Scope of this Thesis

The following items summarize the main novelties covered by this thesis:

1. introducing new YOLO based object detection algorithms for small object detection in drone images based on the YOLOv3 and YOLOv5 architectures;
2. studying the performance of using additional detection blocks in YOLO architecture;
3. studying the performance of using vision transformers in YOLOv5 architecture;
4. studying the object detection performance of multiple YOLO based algorithms on state-of-the-art UAV-taken datasets.

Our findings are collected in two architectures as a part of this thesis, namely YOLODrone (includes improvements for YOLOv3 family) and YOLODrone+ (includes improvements for YOLOv5 family) architectures.

## 1.4 Structure of This Thesis

This thesis contains five consecutive chapters. Chapter 1 serves as an introduction which, essentially, states the motivation of this thesis and lists the main contributions of this thesis. Chapter 2 presents relevant literature as a brief review for object detection. Chapter 3 introduces our proposed deep architectures. It explains the architectural details of both algorithms: YOIODrone and YOLO-Drone+. Chapter 4 is devoted to reporting our experimental results, both qualitative and quantitative, obtained via the use of the evaluation metrics. Chapter 5 concludes this thesis.

In addition those five main chapters, Appendix A devoted to summarize existing object detection datasets that are obtained by UAVs. In Appendix B, the details of a data labeling tool and a novel dataset are explained. Appendix C summarizes our preliminary results on multi-object tracking tasks where we used our proposed object detection algorithms and compared their performances to the other existing YOLO based algorithms.

## Chapter 2

# Literature Review

This chapter provides a review of the related literature and introduces some of the key object detection related terms. The aim of the object detection task is to determine the classes of the existing objects as well as the coordinates of a bounding box of those objects in a given image. There is a vast amount of research done on object detection in the literature. Although, recent studies provide promising results on the task, object detection has been a task in computer vision for a long time. Therefore, it is appropriate to categorize the methods under two main categories including: (i) non-deep learning based (classical methods) and (ii) deep learning based methods. When we are talking about deep learning based methods, it becomes extra important to discuss the existing datasets, since data is a crucial part of the performance of the deep learning based models. Consequently, this section contains three subtopics: (i) A summary of the classical methods: non-deep learning based object detection methods, (ii) a summary of the deep learning based object detection methods, (iii) a summary of the main object detection dataset that used in this work.

Object detection has many important real world applications and because of that, computer vision researchers from all over the world are designing various computational models and techniques to detect visual objects that belong to specific classes such as face, pedestrian or cars in images. The main task of object detection is to answer the question: “What objects are in the image and where are they located in the image?”.

It is possible to categorize the progress of object detection solutions as: ‘traditional object detection’ and ‘deep learning based object detection’ solutions. The Figure 2.1 shows the important developments in object detection solutions in a timeline. Since object detection is one of the oldest tasks in computer vision, there are many conventional approaches (also known as traditional methods) for this task. Most of the time, object detection has two main steps. The first one is using features to find foreground entities which are the possible areas for the object in an image and the second step is verifying these candidates by using a classification algorithm. That is the case for deep learning based and classical solutions.

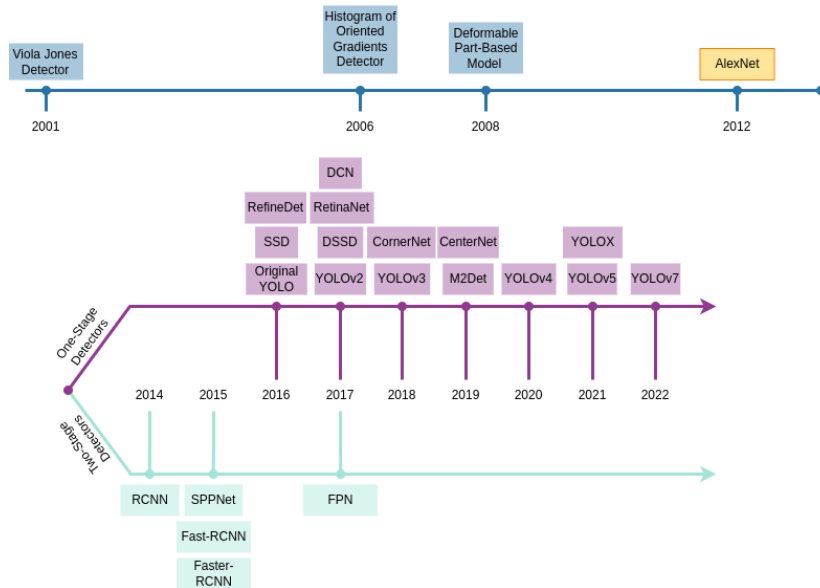


Figure 2.1: Shows the timeline of object detection solutions that were explained in this chapter. The blue line represents the traditional object detection methods. After AlexNet [1] was discovered then deep learning based solutions were split into two. The green line represents two-stage detectors whereas, purple line represents one-stage detectors.

## 2.1 Non-Deep Learning Based Object Detection

A summary of conventional object detection algorithms, i.e., the techniques which do not utilize deep learning, is provided in this section. Such techniques are also called classical techniques, traditional techniques or conventional techniques. The majority of the traditional object detection solutions were based on sophisticated representations of features. Viola-Jones (VJ) detector [27, 28] was one of the most popular conventional object detectors which provided real-time human face detection for the first time including considering different skin colors. The idea of sliding windows was used in Viola-Jones detector. In this approach, the sliding windows walk through possible locations and try to match the patch in the window, if the windows with any human face by scaling the sliding window constantly.

Another important conventional method for object detection is Histogram of Oriented Gradients (HOG) [29] which is a well known feature descriptor and used as an alternative to the scale-invariant feature transform (SIFT) features in many applications. To balance feature invariance such as translation, scale or illumination and the non-linearity such as discrimination of different objects categories, HOG computes a dense grid of uniformly spaced cells to improve accuracy. HOG was motivated by the pedestrian detection problem. However, it was not only used for pedestrian detection but also for a number of different classes. HOG was a very important tool and became an essential part of many object detection algorithms such as [30, 31, 32].

Deformable Part-Based Model (DPM) was one of the most popular and effective approaches among the conventional object detection algorithms. In its earlier stages, DPM was introduced in [30] as an extension to the HOG based object detectors; then followed by improvements in [31, 33, 34, 35]. Divide and conquer idea was used in DPM where the training is the learning of decomposed objects, and the inference is an ensemble of detection of different object parts. The configurations of filters in DPM are trained by weakly supervised learning.



Today, many deep learning based object detectors are influenced by DPM’s valuable insights such as bounding box regression and mixture models. It is known to be more effective to construct detectors for special purposes in a way to find similar areas or regions where the expected objects may occur. These special purposes can be face detection, pedestrian detection or vehicle detection. Therefore, the traditional object detection solutions are also focused on those applications.

Object detection tasks are followed by object tracking. Object tracking has two aims: i) predicting the location of objects in videos and ii) associate the previously detected objects and trajectories in the current frame. Optical flow is used to track objects by measuring the difference between new detections and the trajectory [36]. Possible trajectory of newly detected objects in a new frame are estimated by Kalman Filter in [37]. Additions to the Kalman Filter are made because of the non-linearity of the real life in “Extended Kalman Filter” [38] and “Particle Filter” [39]. The relation between object detection and multi-object tracking were given in Appendix C.

## 2.2 Deep Learning Based Object Detection

With the recent developments in Deep Learning, Convolutional Neural Networks (CNNs) have replaced the hand crafted feature computation part of the classical techniques in many object detection algorithms.

Today, object detection has many applications in various fields including robotic vision, autonomous driving [40], monitoring security and tumor detection [41, 42, 39]. It is also one of the most important parts of the complete object tracking algorithms that are designed based on the tracking-by-detection approach. This approach can be explained as: deep learning based solutions where an end-to-end system takes a previously detected set of objects and develops a unique identification (ID) number for each object. Not only identifying, but also as the frames proceed, the previously detected objects can be associated with the objects in the new frames in these kinds of end-to-end systems.

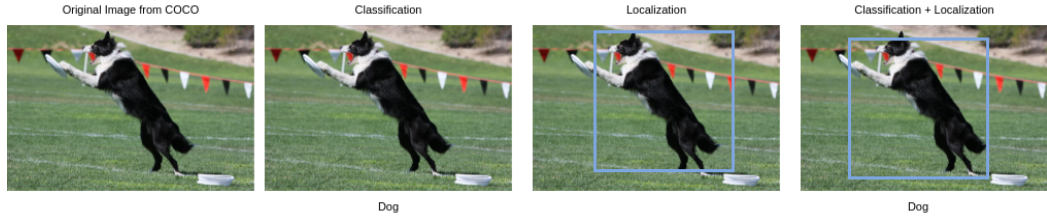


Figure 2.2: Essential parts of the object detection task are visualized step by step. An object (dog) was first classified, then localized and all that information is shown on the final step. The sample image is taken from the COCO [2] dataset.

Object Detection (OD) task essentially includes both classification of the certain objects and localizing them by using bounding boxes. Therefore, we can state that an OD task does not only deal with finding the classes of objects, but also localizing them in the given image by surrounding each object with a tight bounding box. The OD models are supposed to predict a bounding box for every detected object as also shown in Figure 2.2 and in Figure 2.3. The Figure 2.2 shows the computer vision tasks only for the single-object. However, since most of real-life scenarios and object detection benchmark datasets that were given in Appendix A, consists of multiple-objects, the image classification problem can be solved by training a multi-label classifier.

Object Detection task can be used to detect very specific objects in an image such as detecting only faces, detecting only pedestrians or detecting only cars, besides being a more generic object detection application and by supporting too many object classes. Basically, the problem is to answer the question of “Which objects are located and where are they located in the image?” [43]. The generic object detection has the aim to locate and classify a wide variety of objects in an image and put a label on each of those objects along with a rectangular bounding box, while assigning the confidence score for the class type, i.e., the confidence of containing an object inside the bounding box.

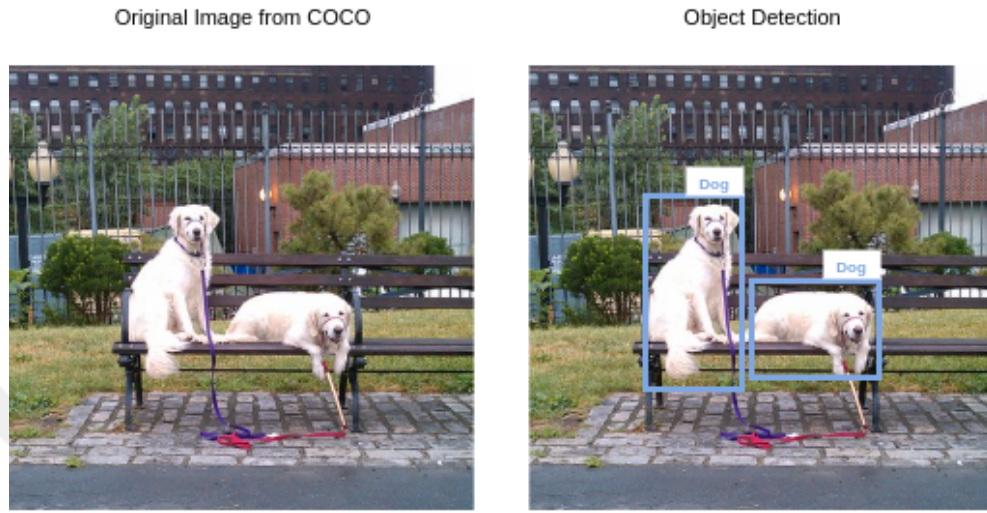


Figure 2.3: Running object detection for dog detection on a sample image from COCO [2]. The bounding boxes and the class name(s) are the output of object detection and they are visualized around the detected objects.

After CNNs are introduced into the computer vision community as in [44], the methods using CNNs to solve object detection problems started to emerge at an increasing pace because of the superior learning capability of CNNs over the classical methods. As also stated in [43], with all the developments in deep learning research, the algorithms for object detection problem can be grouped into two sub-categories [43]:

- **Two-Stage Detection:** Generation of the coarse region proposals are completed in the first stage, then those region proposals are used as input for the second stage to get the final predictions for detection.
- **One-Stage Detection:** Classification and the bounding box regression are completed without making any preliminary generated region proposals so it can be completed in one stage .

Early, proposed solutions were categorized as CNN based two-stage detectors. The Regions with Convolutional Neural Networks (RCNN) family including RCNN [45], Fast-RCNN [46], Faster-RCNN [47]) are the most known members of this two-stage category. Not only the members of RCNN family, but also the Spatial Pyramid Pooling Networks (SPPNet) [48] and Feature Pyramid Networks (FPN) [49] are very popular members of the two-stage detection group of the object detection models.

The first ever published RCNN [45] starts with the extraction of object proposals. The proposed objects are re-scaled and then these re-scaled images fed into a CNN model which was previously trained on ImageNet [50]. Finally, SVM classifiers [51] are used to classify the objects inside those proposed regions. Because of the two-stage processing of the image, RCNN has high time complexity. To overcome this issue, Fast-RCNN [46], which is capable of simultaneously training a regressor for predicting bounding boxes while training a detector within the same network configurations, proposed. However, the regional proposal is still part of the Fast-RCNN that also causes a time limitation. After that, the end-to-end, Faster-RCNN [47] with Region Proposal Network (RPN), that provides cost-free region proposals, proposed to improve the complexity.

The major improvement in SPPNet [48] is the proposed Spatial Pyramid Pooling (SPP) layer which helps the CNN model with generating a fixed-length representation of the feature map without considering the size of image and without the need of rescaling it. Also, the feature maps are computed by using the entire image only once which avoids computing the same features all over again. Another major milestone in the two-stage object detection field is FPN [49]. Most of the CNN models detect the objects by only using the top layer of the model. This creates a limitation for detecting objects of different scales. The FPN is a top-down model with lateral connections. So with this improvement, it is possible for the models to capture the high-level semantics at all scales and detect objects with a wide variety of different scales.

The deep learning models which belong to the single-stage detectors, classify and localize objects in only one process using dense sampling. The difference between two and single stage object detectors can be seen from the Figure 2.4. In single-stage models, the predefined bounding boxes of various scale and aspect ratio were used to localize objects and the bounding boxes predicted the need of region proposal step as it did in two-stage models. Thus, the detection speed increased with the single-stage object detectors.

The computer vision community has been focused on the single-stage object detection models because of the salient improvements in the area. Therefore, various state-of-the-art models are proposed and many of them are mentioned in one of the most recent surveys on object detection [52]. The most popular of single-stage detectors include YOLO Family [53, 22, 54, 21, 55, 24], SSD [56], DSSD [57], RetinaNet [58], M2Det [59], RefineDet [60], CenterNet [61], CornerNet [62] and DCN [63] which will be explained in this section.

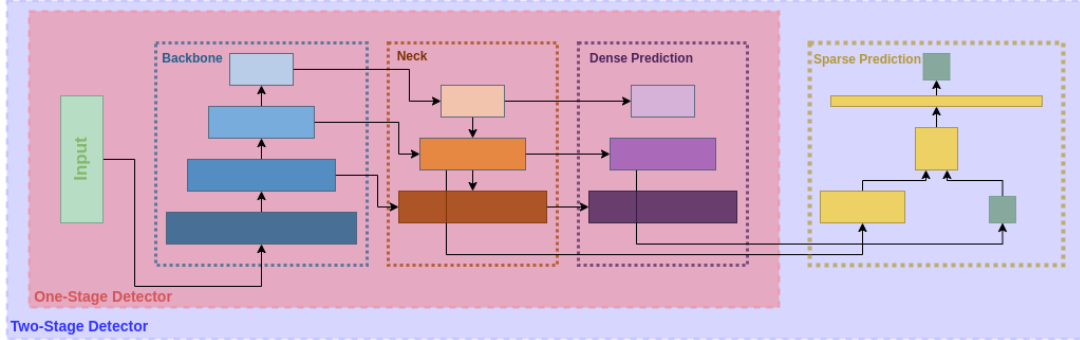


Figure 2.4: The main stages that are generally used while designing either two-stage or single-stage object detection models.

### 2.2.1 YOLO Family

YOLO family is one of the most popular object detection models and because of its open source community the performance of the model improved throughout the years. In this part of the thesis, the development of the YOLO family is explained in detail. YOLO Family belongs to the one-stage networks in an unified model [64]. Over the years, there were five versions of YOLO, each of them upgraded with state-of-the-art ideas. In this section they will be explained. Since the proposed models YOLODrone and YOLODrone+ are based on YOLOv3 [21] and YOLOv5 [22], they will be explain in Chapter 3.

#### 2.2.1.1 Original YOLO

The concept of “Grid Cells” introduced with the original YOLO paper [24]. The main purpose of these “Grid Cells” is to split the input image into cells where each grid cell is responsible for proposing the center point of a single bounding box with a confidence score as shown in Figure 2.5 shows. The concept of “Confidence Score” is to decide if an object exists or not in that proposed bounding box and then calculate the value of the proposed bounding box contains an object with the following equation, where the  $Pr(Object)$  is the probability if an object exists in that bounding box or not and  $IOU_{pr}^{gt}$  is the intersection over union result of the predicted bounding box and ground truth:

$$ConfidenceScore = Pr(Object) * IOU_{pr}^{gt}$$

The center of the proposed bounding box should be between 0 and 1 with this way it has been made sure that the center point of the proposed bounding box is definitely inside the grid cell. It is vital to make sure that these values are between 0 and 1 because each cell is responsible for proposing one center point.

When the top-left corner of the cell is considered as the starting point, if the value is larger than 1 than the cell next to the starting cell or the one below will be the one to propose the center point and the whole unified idea will collapse. Not only the  $(x, y)$  pairs but also the width and height are calculated for each proposed bounding box and they are proportionally to the size of the whole image.



Figure 2.5: A sample figure to represent the idea of the grid cells. Purple point in the center of the cell is the center point  $(X, Y)$  of the bounding box to be proposed and the green point  $(x, y)$  is the coordinate of the grid cells' itself. The image taken from COCO dataset [2]

In order to decide the class of the object each grid cell predicts  $C$  conditional class probabilities:

$$\Pr(Class_i) | Object$$

With the unified approach YOLO uses the concept of “Non-Max Suppression” to eliminate the extensively proposed bounding boxes whose confidence score is lower than a certain threshold. The oldest member of YOLO Family uses Google’s LeNet [65] as the network architecture and has 24 convolutional layers which is followed by 2 fully connected layers. The structure of the layers are 1x1 reduction layers followed by 3x3 convolutional layers. With this paper also another network named as Fast YOLO are introduced.

Loss Function:

$$\begin{aligned}
Loss = & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - x'_i)^2 + (y_i - y'_i)^2] \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{w'_i})^2 + (h_i - h'_i)^2] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - C'_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (C_i - C'_i)^2 \\
& + \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in classes} (p_i(c) - p'_i(c))^2
\end{aligned} \tag{2.1}$$

### 2.2.1.2 YOLOv2

One of the major improvements to the original YOLO, in YOLOv2 [53] is the concept of “Batch Normalization” [66]. This method of normalization provides stability to the distribution for the input layers while training the network. The major contribution of this improvement is to reduce the training time while generalizing the network.

Secondly, the idea of “Anchor Boxes” were introduced to YOLO in this paper. Anchor boxes are the preliminary decided box size ratios by clustering the width-height combinations of all the bounding boxes in the training dataset. With the help of the anchor boxes, each grid cell is able to propose more than one possible bounding box. This idea is introduced in order to deal with different positions of the possible bounding boxes. The k-means clustering algorithm was used to find these combinations rather than manually deciding on these ratios.

Last but not least, YOLOv2 model has the time to adjust its filters in order to work better in terms of higher resolution inputs. The original YOLO, used 224x224 image size during the training phase, then the resolution was increased to 448. With YOLOv2 the classification network was tuned for 10 epochs in 448x448 resolution on ImageNet. The network has 30 convolutional layers for Darknet. The fine-grained features of the image were lost because the input image was downsampled. Thus, this version could not deal with the small objects.



### 2.2.1.3 YOLOv4

This version of YOLO was not published by the authors who developed the first three versions of the model. As well as the other YOLO versions, YOLOv4 [55] is an improved single-shot detector. There are three main parts of this version:

- **Backbone:** SPP [48] integrated with CSPDarknet53 [55] for feature extraction where the CSP [67] used as an improvement to the backbone. Moreover, DropBox Regularization and Class Labels Smoothing added to the backbone in YOLOv4.
- **Neck:** PANet[68] for semantic representation as well as Mish Activation, Spatial Pyramid Pooling added as novelty to YOLOv4.
- **Head:** The original YOLO layers in YOLOv3 [21] are used for prediction as well as Complete IoU (CIoU-loss) used in the head of this version of YOLO.

As also explained in the YOLOv2 part in this section, the k-means clustering algorithm is used to decide the width and height ratios of the anchor boxes. Also, mosaic augmentation was released as a part of this paper. Mosaic augmentation, simulates random crops and also maintain the size of the labeled objects and with this way it can help the model to perform better. Also, it combines classes which may not come together in the training set. Finally, it increases the number of objects in the images. In order to achieve this final model authors of the paper made a lot of experiments on different approaches from the computer vision community as can be read in [55].

Single-Shot Detector (SSD) [56] has many feature maps of various scales which were generated after each layer. By using these feature maps, SSD predicts the class probability scores for a certain set of bounding boxes for different scales. As a result, there are various numbers of bounding boxes with different scales for that particular feature map extracted from one layer of the model. Deconvolutional Single Shot Detector (DSSD) [57] is the altered version of SSD and developed to predict different sets of objects having different sizes as well. It uses a ResNet as backbone and in the prediction module one residual block was added to the

single prediction layer. The deconvolution module augments the feature map resolution which makes it possible for smaller objects to be detected. Another single-shot object detection model RetinaNet, [58] considers a novel focal loss as a classification loss in order to solve the class imbalance problem. This loss focuses on the positive training samples and avoids the major number of negative samples. Thus, the model has the ability to be trained with unbalanced datasets.

Another popular single-shot object detector is M2Det [59] contains a Multi-level Feature Pyramid Network (MLFPN) which is able to construct more efficient feature pyramids. Multi-level and multi-scale features which are obtained by MLFPN, are generated and these features are used for object detection. RefineDet [60] consists of two major modules. The first one is the refinement module and the second one is the object detection module which is connected with a transfer connection block. RefineDet is used to transfer features from the previous module to the upcoming module for improved prediction of objects with an end-to-end manner. RefineDet uses a two-step method to predict the hard-to-detect objects such as small objects more accurately. Another single-shot object detection method which focuses on one of the major issues with the regular CNNs: CNNs can only focus on the features with fixed square size because of the kernel. As a result, the receptive field is not able to cover the entire object pixels as it's supposed to do. To solve this issue Deformable Convolutional Networks (DCNs) [63] are introduced to handle this by producing the deformable kernel.

CornerNet [62], eliminates the need for anchor boxes, which were used in many single-stage methods including YOLO models. CornerNet uses hourglass [69] architecture as backbone model which is then followed by two prediction layers one is being used for bottom-right corner and other one is being used for top-left corner. Each one has its own corner pooling module which is used to pool features extracted from the hourglass model before predicting. Also, the paper proposes a new pooling layer type named "Corner Pooling", that is used for networks to localize corners in a better way. Another anchorless idea is CenterNet [61] where at the pre-processing step, a map of ground-truth bounding box centers are drawn. Then, a "Gaussian Filter" is used to generate a smooth distribution that reaches to a peak on the center of the object. After the pre-processing step

two prediction heads were trained; one is to predict the confidence heat map, where other one is to predict the regression values for box dimensions and offsets.

## 2.3 Object Detection Datasets

To train a deep learning model one of the most important things is to have a dataset. Deep learning models are a solution for computer vision problems however it is a time and data-intensive task. Many deep learning models are data-hungry and they require a lot of data to create the best model. The quality of the data is as important as the quantity, even for the deep neural network models designed by experts. Deep learning models can be described as Garbage In Garbage Out (GIGO) by George Fuechsel which basically means unless the model is fed with high-quality data, it will not deliver a high-performance result. As also mentioned in [70], most researchers spend almost 70% of their time analyzing and understanding the datasets. Although there are some alternative solutions such as transfer learning, unsupervised learning, having a high-quality dataset remains one of the most important ingredients to build a working deep learning architecture in the real-world.

Yet, the computer vision datasets typically comes with various limitations because of the way and the environment that they are acquired. The performance of any deep learning model depends on datasets' quality and quantity. Although there are many open-source datasets with high amount of data samples, they still suffer from the following limitations [71]:

1. **Insufficient Data:** Unavailability of broad data samples of data points compulsory for deep learning algorithms especially CNNs.
2. **Bias and Human Error:** Many methods used for collecting data may cause bias or human error.
3. **Data Quality:** The datasets which are obtained from real-world images or videos are not organized and they are complex.
4. **Privacy and Compliance:** Datasets are not shared by many resources

because of some privacy regulations such as PDPC [72] or mainly known as KKVK [73] in Turkey.

5. **Annotations Process:** Most of the datasets are labelled manually for quality. However, this is such a time consuming and expensive process.

Therefore, as it can be understood from the explained limitations of the datasets, it is vital to work with the right kind of data. To find the right kind of dataset, it's important to review the dataset before starting to train the deep learning model. Also, it is important to review the source, as well as, how the dataset was labelled. Using the correct dataset also means using enough data. It can be said that, the more data you use for training, the better your model gets. Not only large and open source datasets can be used but also two small datasets can be combined. Not only quantity but also quality matters in deep learning datasets. The quality of a dataset can be understood from the real-life performance of a model which was trained with that dataset. More diversity from a dataset to the model, will result in better recognizing subtle differences which will also help to avoid false positive predictions. It is also very critical to consider the bias that may exist in datasets.

There are different ways in which training the model with biased data can negatively impact the accuracy of a deep learning model. The bias can be explained as anything that's missing from the dataset which shows that the dataset does not cover the real-world cases. Therefore, it is critical to consider the differentiating factors: seasonal and weather trends, geographic differences, different time of the day, camera angle or background. In order to deal with the bias in datasets, Data Augmentation methods are used widely. Data augmentation is a method for increasing the number of images in the existing dataset by adding small changes to those images. The data augmentation techniques used in this work will be explained in Chapter 4 which is the part of the experimental process.

The scope of this thesis is to study object detection tasks in images which are obtained from Unmanned Aerial Vehicles (UAVs) commonly known as drones. Most of the existing datasets on this task are obtained from cameras which are placed on cars as it can be seen from Figure 2.6. Therefore, it is possible to say that the datasets obtained from drones and cars have different challenges and so require different approaches. These challenges in drone datasets are because of the scale of the objects and variety of the capturing angles. Because of these differences the solutions for detecting multiple objects should be unique for the images obtained from drones. The following part of this section provides brief information about the used object detection dataset. The datasets are explained to make it easier to compare and state the differences and challenges between them in Appendix A. The datasets which are used in this work, considered according to all the explained factors above.

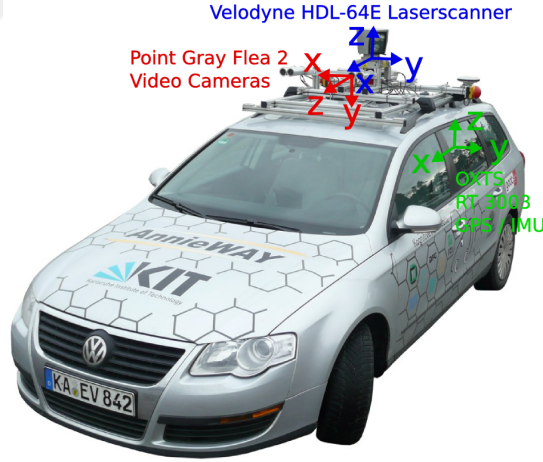


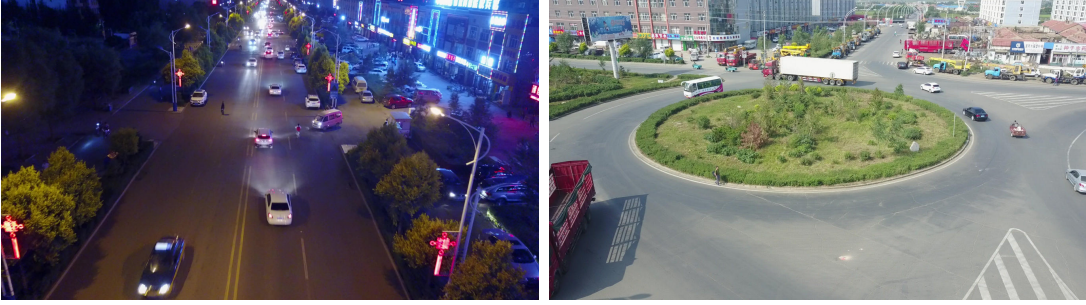
Figure 2.6: Shows the data collection setup which was used to collect data for the KITTI dataset [3].

VisDrone dataset used as the main training and testing dataset in this thesis. VisDrone is published with two versions. The first one published in 2018 [74] as VisDrone2018 which consists of 263 video sequences and 10,209 static images. While recording these images, multiple drones were used over the 14 different cities of China. Later, the second one was published in 2019 as visDrone2019 [4] which introduced 25 more new videos and made the total number of representative frames 261,908. Between these 25 new sequences, 12 of them are recorded in day time and rest is recorded during the night time.

This difference in recording provides a variety, thus giving more learning capability to the deep learning models. These datasets are publicly available for multi-object tracking, single-object tracking and object detection problems. This dataset is also used as a primary work to this thesis in order to solve the object detection problem [75, 76].

The datasets which are created by using images and video footage that are taken by using UAVs require extra consideration and learning capacity. The detailed samples from ground-taken and aerial datasets are given in Appendix A. The drone datasets are challenging for similar reasons because of the difference in as also given in Figure 2.7:

- camera angle
- altitude of the drone
- density of the frames in terms of labelled objects
- time of the day
- weather conditions



(a)

(b)

Figure 2.7: Sample images from VisDrone dataset. The recording time as well as angle and altitude of the drone is varied in this dataset as can be seen from the samples.

## Chapter 3

# Proposed Models: YOLODrone & YOLODrone+

YOLODrone and YOLODrone+ models were proposed with this thesis and in this chapter their architecture and the approach to increase their learning capacity are explained. The improvement in learning capacity is required for the already existing YOLO models to be able to detect small objects in UAV datasets more accurately. YOLODrone is built on top of YOLOv3 [21] and YOLODrone+ is built on top of YOLOv5 [22].

As mentioned in Chapter 2, techniques using CNNs provide monumental growth in accuracy to solve computer vision problems including object detection. A CNN takes an input image and then with the help of its trainable weights and biases, it can learn to differentiate one object from another.

The improvements and studies proposed with this thesis concerning YOLO architectures can be summarized on three main subjects:

1. Number of detection layers;
2. Effect of using a vision transformer;
3. Effect of data augmentation.

Object detection research is considered as a growing field. As Figure 3.1 suggests, in the last 5 years the number of published papers related to object detection increased dramatically, when compared to the research done before 2015. Development in YOLO architecture also affected this dramatic increase. The starting point of this research was YOLOv3, then because of the increase in accuracy introduced with YOLOv5 the scope of this thesis was extended to YOLOv5 as well. Therefore, in the next two subsections, first the architecture of YOLO-Drone which is based on YOLOv3 will be described, and then the architecture of YOLODrone+ which is based on YOLOv5 will be discussed. Then in Chapter 4 the experimental results of the proposed models will be given.

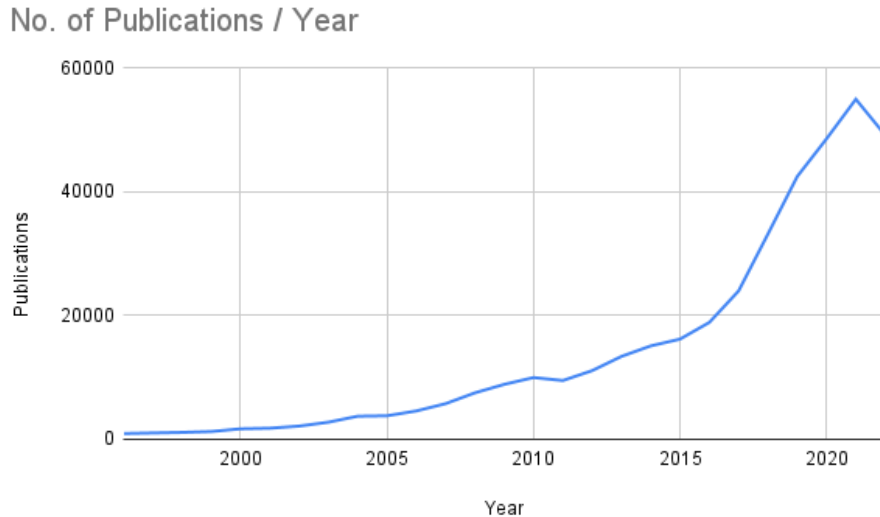


Figure 3.1: Shows the number of papers published related to “object detection” over each year, data used in the graph was obtained from Google Scholar.



## 3.1 YOLODrone

The state-of-the-art object detectors are mostly designed by considering the problems that come together with datasets that are created for object detection such as MS COCO [2]. However, most of the time, these datasets consist of ground-taken images. Therefore, object detectors which are trained with datasets containing ground-taken images generally perform poorly on datasets which are obtained with UAVs. This situation is also valid for the YOLO algorithm, thus, as a part of this thesis, the YOLOv3 architecture was extended.

### 3.1.1 YOLOv3

To deal with the smaller objects and increase the accuracy performance of the model in YOLOv3 [21], the YOLOv3 authors used ResNet [77] which includes skip connection layers to deal with the vanishing gradient problem of the deep neural networks. With the help of ResNet, they proposed the Darknet53 architecture as can be seen from Figure 3.3. Also, with YOLOv3, the model is able to detect the objects which vary in size in the dataset. In this version of YOLO, the authors used different sizes of the YOLO layers. Therefore, the model is able to predict objects in different scales where the features are coming from the last 3 residual blocks of 3 different sizes.

YOLOv3 is also known for using anchor boxes. The main purpose of anchor boxes is to detect the objects with different sizes. They also help when the center of the objects to be detected are in the same grid cell. The final width and height values of the predicted bounding boxes are determined by multiplying the anchor boxes with the width and height values obtained from the output of the backbone part. K-means algorithm can be used to determine the values of the initial values of these boxes before the training.

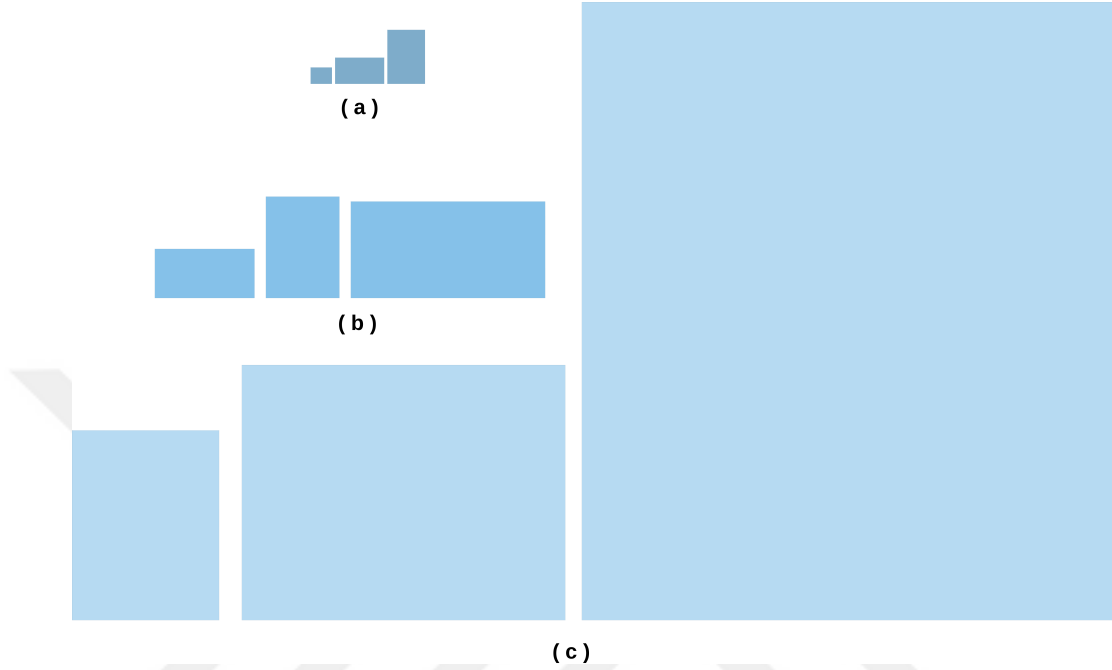


Figure 3.2: Shows various scales of the anchor boxes in YOLOv3. The values of these anchor boxes are calculated by the K-Means algorithm. There are three different sizes because the detection layers of YOLO split the input image into grid cells in three different sizes. In this figure, the sizes of the anchor boxes are defined by using MS COCO [2]. The following shows the (height, width) pairs in: (a) (10,13), (16,30), (33,23); (b) (30,61), (62,45), (59,119); (c) (116,90), (156,198), (373,326)

K-means clustering [78] is a popular unsupervised learning algorithm in machine learning. Unsupervised learning algorithms find the hidden patterns in the given data by running the algorithm without without any supervision [79]. “K” in k-means represents the total number of clusters or groups to be seen as a result. Also, it is known as the number of centroids which are the locations representing the center of the cluster. Then the k-means algorithm allocates every data point in the dataset to the nearest centroid.

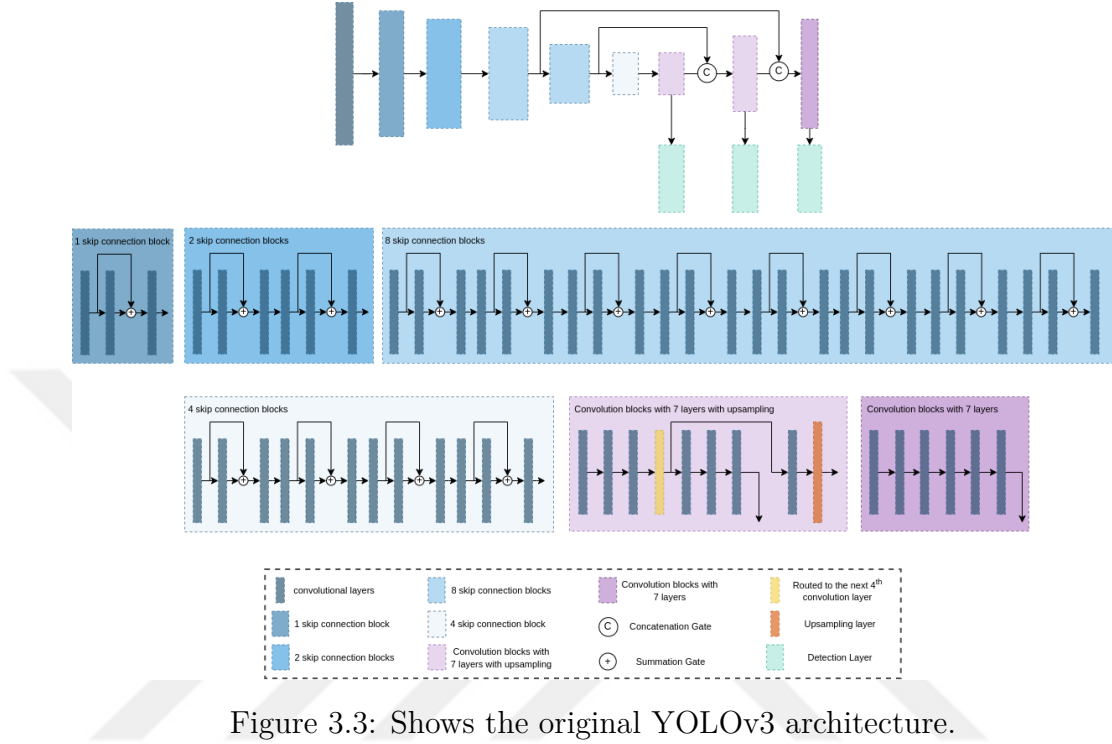


Figure 3.3: Shows the original YOLOv3 architecture.

The dimensions of the anchor boxes were calculated according to the used dataset in YOLODrone. The number of pairs of anchor boxes were experimented with for the original YOLOv3 model. For example, if the experiment is to test 4 anchor boxes for each detection layer, then the "k" value for the k-means algorithm would be set to 12, which is derived from the 3 detection layers and 4 different types of anchor boxes. The results of experimenting with the total number of anchor boxes are presented in Chapter 4.

### 3.1.2 YOLODrone Architecture

It is hard to detect small objects accurately with YOLOv3 model in images taken from unmanned aerial vehicles (UAVs). In aerial datasets, the size of the objects can range from very small to large, which can be challenging for the model, when compared to the ground-taken images, such as those in the PASCAL VOC or MS COCO datasets, as illustrated in Figure 3.4. One reason why YOLOv3 has better accuracy than the original YOLO and YOLOv2 version is that YOLOv3 is making proposals for objects at three different layers (see [21]). As the total number of prediction layers increases, the model is able to adjust itself and its parameters better to focus on objects at different scales. With that idea, YOLODrone architecture was developed by having more detection layers, when compared to original YOLOv3 architecture.



Figure 3.4: Two sample images demonstrate how the viewing angle and the object scales change an object’s appearance in an UAV-taken image. Both of the sample images are taken from the VisDrone2019 [4] dataset.

While the original YOLOv3 architecture uses 3 detection (prediction) layers, our YOLODrone architecture uses 5 detection layers. The overall architecture of YOLODrone is given in Figure 3.5 and further details of its architecture are provided in Table 3.1. In Figure 3.5, the contents of each block and how they are connected to other blocks are illustrated. The detailed scheme of each block is also given in the same figure in a color-coded way. Similar to YOLOv3 architecture, the proposed architecture is similar to an encoder-decoder architecture where in the first part of the network, the height and width of the feature maps shrink; and after that, they expand in the second half.

The shrinking process, which is, essentially, to reduce the height and width of the feature map, is done by the stride operation. The first convolutional layer has stride of 2, and after that first convolution operation, the stride was set to 1 for each additional convolution operation in the same block. To increase the height and width of the feature maps, additionally, an up-sampling layer is used at the beginning of each block in the second half of the network. In YOLODrone architecture, there were 6 different blocks which were color-coded as shown in Figure 3.5. Each color-coded block contains varying numbers of convolutional layers and skip connections.

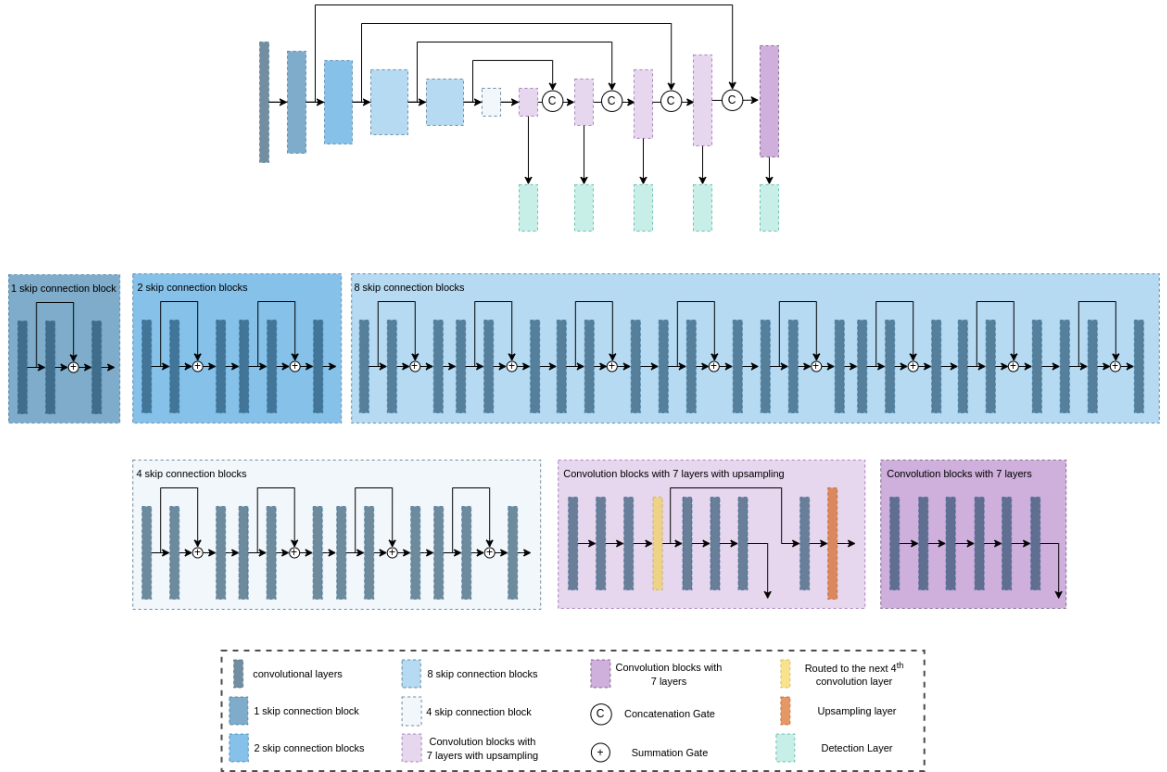


Figure 3.5: Shows the architecture of YOLODrone. Different colored blocks in the architecture represent different blocks. The meaning of each color is given below the architectures. YOLODrone architecture has total of five detection layer

As mentioned earlier, anchor boxes are used in YOLODrone architecture. As the anchor box values are given prior to training, they act as hyperparameters affecting the performance of the detection process. In our proposed approach, three different anchor boxes for each grid cell were experimented on for each detection layer. Similar to YOLOv3, the optimal anchor box values were found

| Block Name          | Skip Connections | Convolutional Layers | Size    | Channel (Input) | Channel (Output) | Number of Filters  | Activation Function |
|---------------------|------------------|----------------------|---------|-----------------|------------------|--------------------|---------------------|
| Input               | -                | -                    | 416x416 | -               | -                | -                  | -                   |
| Convolutional Layer | -                | 1                    | 3x3     | 1               | 8                | 8 (x1)             | Leaky Relu          |
| Residual Block      | 1                | 3                    | 3x3     | 32              | 64               | 32 (x1), 64 (x2)   | Leaky Relu          |
| Residual Block      | 2                | 5                    | 3x3     | 64              | 128              | 64 (x2), 128 (x3)  | Leaky Relu          |
| Residual Block      | 8                | 17                   | 3x3     | 128             | 256              | 128 (x8), 256 (x9) | Leaky Relu          |
| Residual Block      | 8                | 17                   | 3x3     | 256             | 512              | 256 (x8), 512 (x9) | Leaky Relu          |
| Residual Block      | 4                | 9                    | 3x3     | 512             | 1024             | 512(x4), 1024(x5)  | Leaky Relu          |
| Detection Layer     | -                | 7                    | 3x3     | -               | -                | -                  | Leaky Relu          |
| Detection Layer     | -                | 7                    | 3x3     | -               | -                | -                  | Leaky Relu          |
| Detection Layer     | -                | 7                    | 3x3     | -               | -                | -                  | Leaky Relu          |
| Detection Layer     | -                | 7                    | 3x3     | -               | -                | -                  | Leaky Relu          |
| Detection Layer     | -                | 7                    | 3x3     | -               | -                | -                  | Leaky Relu          |
| Output              | -                | -                    | 416x416 | -               | -                | -                  | -                   |

Table 3.1: Shows the details of YOLODrone architecture.

by applying k-means algorithm on the used (drone) dataset. Table 3.2 compares proposed YOLODrone to YOLOv3 in various aspects.

To sum up, the following contributions were made in this part of this thesis:

- experimenting on the effect of using different data augmentation methods on the detection performance,
- introduction of the YOLODrone architecture for higher object detection performance in UAV-taken images, when compared to YOLOv3,
- study the effect of changing the values of the anchor boxes,
- presenting improved results in  $mAP_{50}$  metric for the VisDrone2019[26], when compared to YOLOv3.

|                            | Original YOLOv3 | YOLODrone |
|----------------------------|-----------------|-----------|
| Number of Convolutions     | 75              | 95        |
| Number of Skip Connections | 23              | 23        |
| Number of Detection Layers | 3               | 5         |
| Number of down sampling    | 6               | 8         |
| Number of up sampling      | 2               | 4         |

Table 3.2: Comparison of YOLOv3 to YOLODrone.

## 3.2 YOLODrone+

As of the time when this thesis was initially written, the YOLOv5 was the latest, the widely accepted YOLO version of the YOLO family. Therefore, as a part of this thesis, all the improvements originally introduced for YOLOv3 were also implemented on top of YOLOv5 architecture. Therefore, here, we first explain YOLOv5 which is our base network, transformers, and then explain the proposed YOLODrone+ model.

### 3.2.1 Related Background: YOLOv5

An important note to mention about YOLOv5 is that there is no any official peer-reviewed paper for YOLOv5 version. After YOLOv4 published, YOLOv5 was only published as a GitHub repository by the company called Ultralytics LLC [22]. The Darknet frameworks which were explained in Chapter 2 earlier were written in C. With this new version, the framework was transferred into PyTorch which is an open source framework for machine learning tasks based on the Torch library [80].

Although, the YOLOv5 architecture is quite similar to the YOLOv4 architecture, the main difference is the implementation of the selection process of the anchor boxes. In the YOLOv5, the model is capable of learning the anchor boxes before the training process. As also mentioned in Section 3.1, the prediction of the bounding boxes was determined according to the scales of the anchor boxes. Moreover, the authors were used LeakyReLU [81] and sigmoid activation function as the activation function in this model; The Binary Cross-Entropy with Logits Loss function was also used in the loss function in YOLOv5 algorithm.

CSPDarknet53 is derived from CSPNet[82], which is a method that aims to optimize the weights of a network during the back-propagation process by reducing the amount of duplicate gradient information. This is an alternative approach to traditional methods. The CSPNet is derived from DenseNet [83] which uses the

previous layer's input as a part of its input, and concatenates that with the output of the convolutional layer of the previous layer and passes that to the next layer as input. This process is illustrated in Figure 3.6. Each stage of DenseNet has a dense block and a transition layer. The network moves forward through these stages until the final transition layer. Because of this idea, similar to the ResNet [77], the model suffers from vanishing gradient problems when the network is too deep.

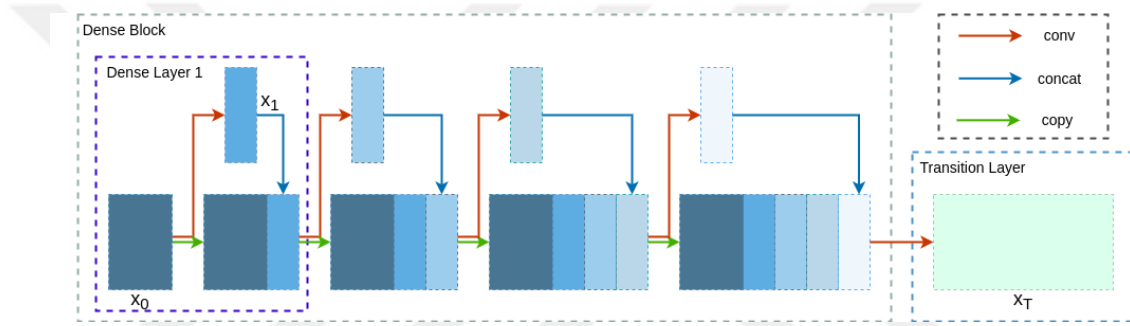


Figure 3.6: DenseNet is a type of convolutional neural network (CNN) designed to address the problem of vanishing gradients in deep learning. It does this by introducing connections between all of the layers in the network, allowing information to flow more easily between them and improving the flow of gradients during training as shown.

The "Cross Stage Partial" (CSP) [82] part was added to DenseNet[83] in order to solve the vanishing gradient problem and with this way, it is possible to make the model even more deep. The main idea of the CSP is to split the input into two parts as shown in Figure 3.7. One part to follow the DenseNet branch and the other one to save for later and to be concatenated with the transition layer. The reason behind this idea is to learn from copied gradient information which is coming from different dense layers. The CSP preserves the features during the back-propagation operation while giving the network the ability to use the learned features. Also, the number of parameters decreased as the learned features were preserved while moving forward into deeper layers of the network.



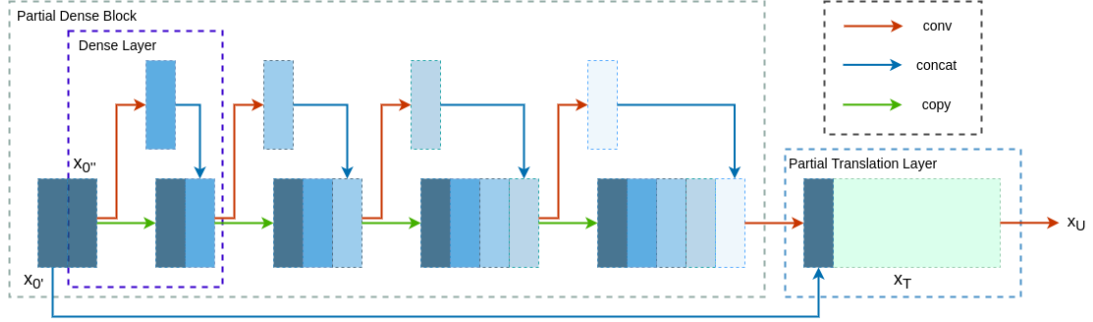


Figure 3.7: DenseNet with the Cross Stage Partial part added which makes CSP-DenseNet

In YOLOv5, the CSP idea was integrated with the Darknet53. The ResNet architecture was used as a part of the backbone and in this version, the residual blocks were replaced by the dense blocks. In order to use the detection time efficiently, only the last convolutional block of the Darknet53 architecture was replaced with the Dense Block.

Another addition to YOLOv5 is the "Spatial Pyramid Pooling" (SPP) part in the architecture. The feature map, which is coming from the CSPDarknet53 backbone, passes through a SPP block before it moves forward to the neck part. One reason behind this additional block is to increase the receptive field of the feature map. The original SPP block was modified in order to save the information coming from the spatial dimension. The  $1 \times 1$  convolutional block was used between the backbone network and SPP in order to reduce the input feature map. The Figure 3.8 also explains the modified SPP [84].

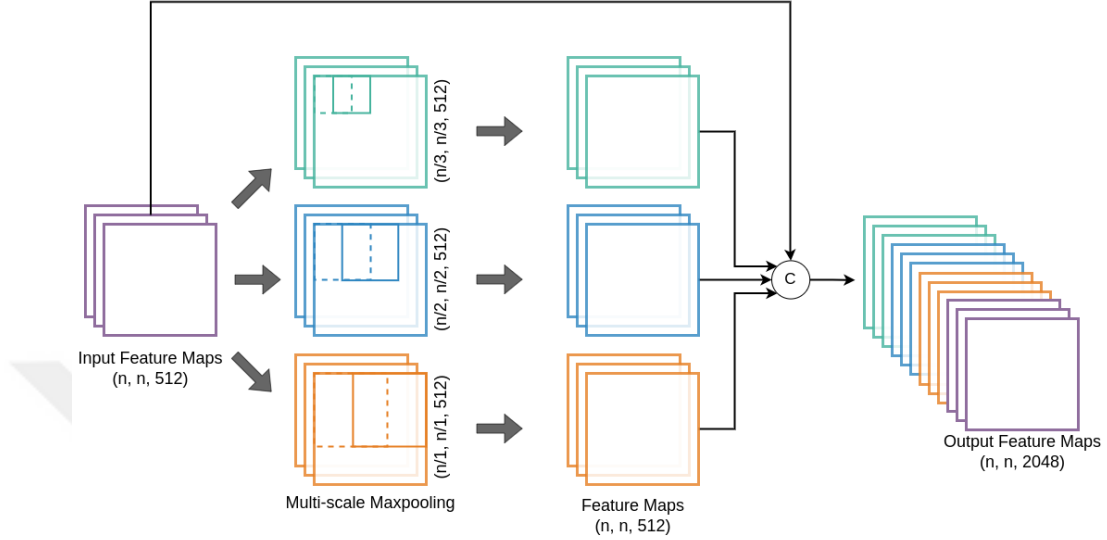


Figure 3.8: Modified SPP divides the features along the depth dimension and applies SPP on each divided part. Then combines them again to create an output feature map.

As mentioned earlier, the idea of FPN is used in order to deal with the loss of spatial information. With YOLOv5, the PANet[68] architecture, which is an advanced version of the FPN, was used. Because the introduced backbone is deep, it takes a long way for the feature map to travel from low-level to high-level. In PANet, not only top-down path is used as in FPN, but also bottom-up augmentation path is added. That way, there is a direct connection between the learned features from low-level layers and the higher ones.

The head part of the YOLOv5 is kept the same as in the YOLOv3, where the model uses the anchor based detection and three different sizes of possible predictions. The head part of the model's purpose is to make the dense predictions which are used to find the bounding box coordinates of  $x, y, h, w, Pr_{confidencescore}, Pr_{class}$ . To summarize, YOLOv5 includes major improvements to the YOLOv3.

The YOLOv5 architecture includes three essential parts: (i) the "backbone" responsible for feature mapping, which form the basis of object recognition models, (ii) the "neck" responsible for extracting more detailed features so that objects of different sizes can be detected, and (iii) the "head", which is responsible for

object recognition. The original YOLOv5 architecture is illustrated in Figure 3.9. Each of those sections use best-to-date structures and approaches:

- **Backbone:** Due to its learning capacity in feature extraction, *Spatial Pyramid Pooling (SPP)* [48] is added to the *Darknet53* structure and *Cross Stage Partial Network*, as shown in Figure 3.7, is integrated into the architecture. This architecture is called *CSPDarknet53* [55].
- **Neck:** *Path Aggregation Network (PANet)* [68] has been added to the neck for semantic definition.
- **Head:** YOLOv3 is kept the same as the structure used in the [21] statement.

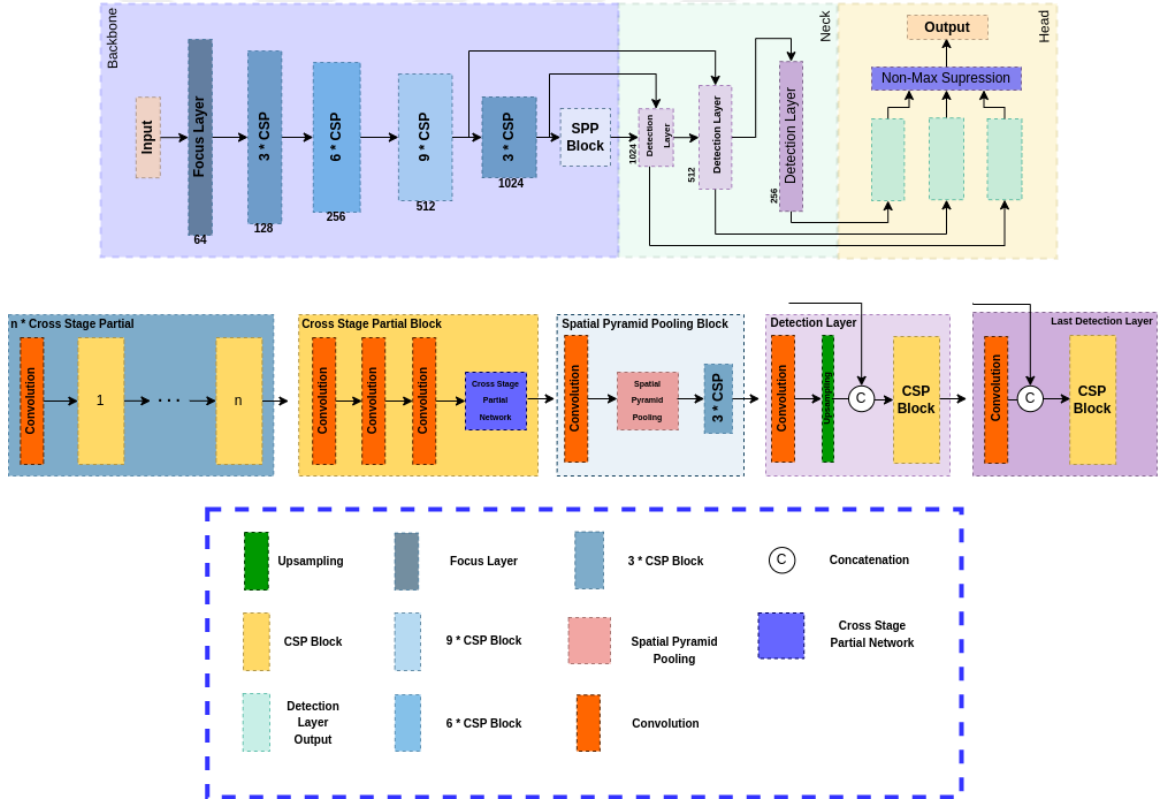


Figure 3.9: Shows YOLOv5 architecture with all blocks are given in color-coded.

### 3.2.2 Related Background: Vision Transformers

The transformers were initially used in the Natural Language Processing (NLP) field. Transformers yielded state of the art results on many recent NLP problems and has become almost a standard solution since they were introduced in [85] and also used in [86, 87, 88]. In NLP problems, data consists of a sequence of words which are coming from a set of vocabulary. The transformer architecture includes an encoder that produces a general encoded output that is not specific to any particular task, such as classification. This makes the transformer architecture versatile and suitable for a wide range of tasks. The Vision Transformer (ViT) idea proposed in [89] is an extension of the original transformer concept with a small alteration in the implementation to deal with the image and video based data. Specifically, a Vision Transformer uses tokens but the generic architecture of the originally proposed transformer [85] and Vision Transformers [89] remains same.

To begin the process, the input is divided to a set of patches (known as visual tokens). These tokens are added to a set of vectors that are encoded. The position of one patch is included in the encoded vector, which is then used as an input into the transformer encoder. An example model of a vision transformer can be seen in the Figure 3.10. Vision Transformer encoder have multiple blocks and each of these blocks consists of three main processing elements:

1. **Layer Normalization:** training process was kept on track and the model adapted to the variations in the training images.
2. **Multi-head Attention Network (MSP):** network responsible for generating attention maps from visual tokens. The attention maps help the network to focus on important regions in the image such as objects.
3. **Multi-Layer Perceptrons (MLP):** a two-layer classification network with Gaussian Error Linear Unit (GELU) [90]. The last MLP block in the model, also termed as MLP head, is the output of the transformer layer. An application of softmax on this output can provide the predicted labels if the task is classification.

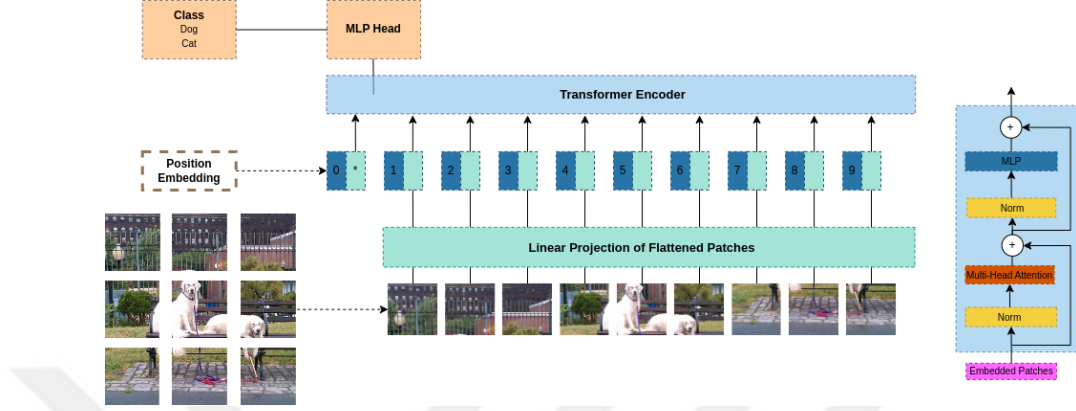


Figure 3.10: The visual representation of the vision transformers. Instead of computing relationships among all pixels in an image, Vision Transformers uses small sections of the image such as e.g., 3x3 pixels in this figure and computes the relationships among the pixels in these sections. The sections of the sample sections of the image are placed in a sequence and given position embeddings, that are learnable vectors. The sections are arranged into a linear sequence and multiplied by an embedded matrix. The resulting sequence, along with the position embedding, is fed into a transformer for further processing as the figure suggests.

### 3.2.3 YOLODrone+ Architecture

Since the dimensions of the objects in the images obtained with UAVs vary, deep learning models to be used for object detection with UAVs should have a learning capacity more than ever before. To increase this learning capacity in YOLOv3, the number of layers responsible for detecting the head section was increased as published in [75]. The use of vision transformers in a deep YOLO architecture along with the increase in the total number of the used detection layers of the head is among the introduced changes in this thesis. While applying the attention mechanism of the transformers in the field of computer vision, the method of using patches by dividing the images into 16x16 image patches was proposed previously (see [89]), and an increase in the total number of parameters of the network was observed.

Considering all these methods, in YOLODrone+, the total number of detection layers was increased and the Vision Transformers (ViT) were used in the detection layer (the one which was divided by the least number of grids). The main goal here

is to provide a better consideration of the size difference between the objects that occur in the UAV datasets. As it is known from the literature, [75] the model can better detect small objects by increasing the number of detection layers. On the other hand, to detect small objects and large objects next to each other without proposing a false positive detection by the model, vision transformers were added to the detection layer which is responsible for detecting larger objects, and as a result, a hybrid architecture including both convolutional neural networks and vision transformers was obtained.

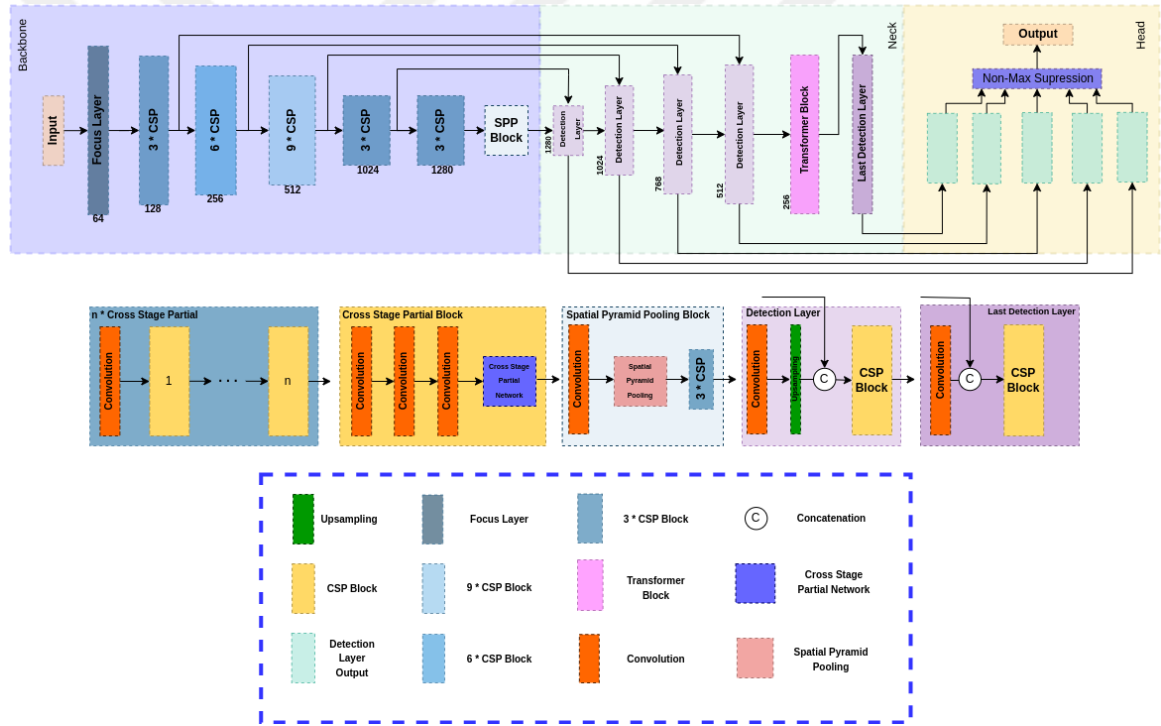


Figure 3.11: Shows YOLODrone+ architecture in a color-coded form. Not only are the number of detection layers different from YOLOv5 but also, the Vision Transformers were added to the neck part of the model.

# Chapter 4

## Experiments

In this chapter, the performance of our proposed YOLO-based algorithms are evaluated on different datasets and compared to other YOLO architectures. To compare the performance of the models in a fair way, an experimental methodology followed where we used the same metrics and the same training and test data sets. To evaluate and compare the performance of the algorithms, appropriate performance metrics are needed. For that reason, in this chapter, the used datasets and metrics are described before presenting the experimental processes and results.

## 4.1 Object Detection Evaluation Metrics

Understanding and using the appropriate evaluation metrics are a vital part for deep learning research. Therefore, this subsection is dedicated to explaining the common evaluation metrics used for object detection. The elements of the output of an object detection model essentially consists of an array of bounding boxes, confidence scores and predicted class number (class id). Since there are many benchmarks and a large variety of different models, there are not any particular standard output style. However, the logic to calculate a test metric is the same and, thus, can be implemented for different output styles. Although, there are different metrics to calculate the performance of the object detectors, commonly used detection metrics rely on Average Precision (AP).

According to [91], the metric AP and its variants are calculated by using the following primitive terms:

- True Positive (TP) : correctly detected ground-truth bounding box.
- True Negative (FP) : incorrectly detected bounding box which is a non-existent object or a misplaced detection of an existing object according to the ground truth.
- False Negative (FN) : undetected ground truth bounding box.
- False Positive (TN): this metric is usually used as a part of the confusion matrix. However, in object detection task, there can be an infinite number of bounding boxes that are not supposed to be detected in any image. Therefore, this metric is omitted while calculating the test values. Therefore, measurement curves that involve the True Negative value such as Receiver Operating Characteristic (ROC) which is obtained by plotting True Positive Rate (TPR) against the False Positive Rate (FPR) according to various thresholds to measure the if a model is optimal and discard less optimal ones [92].



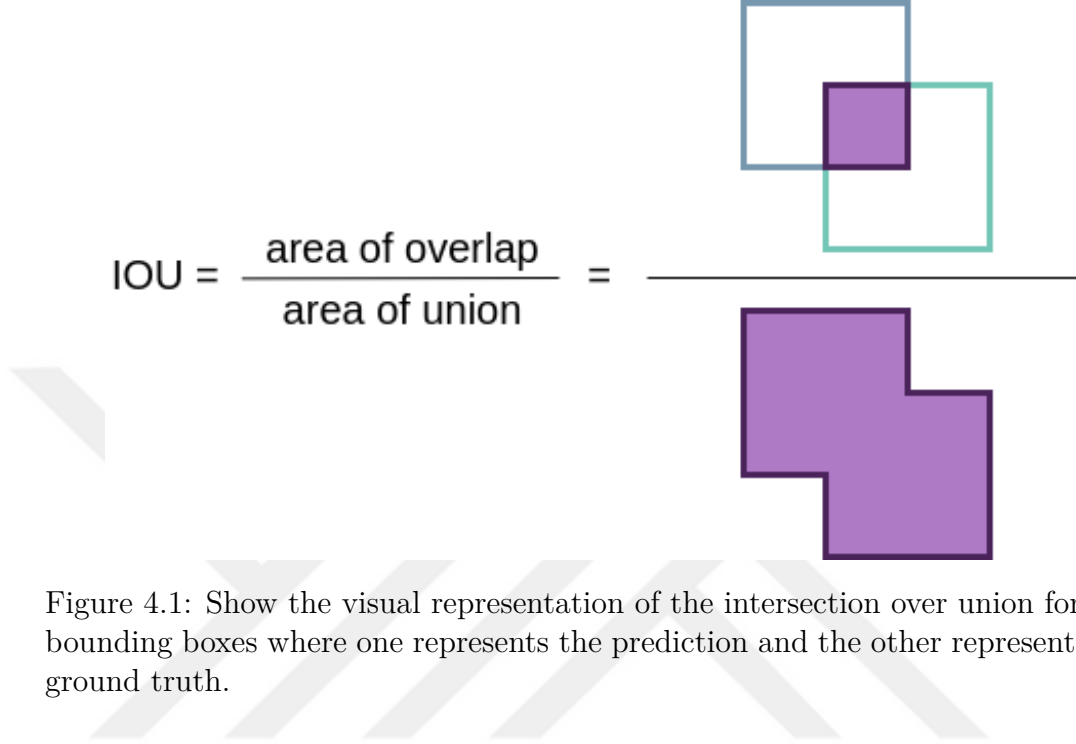


Figure 4.1: Show the visual representation of the intersection over union for two bounding boxes where one represents the prediction and the other represents the ground truth.

In order to decide whether a detection is correct or incorrect, Intersection over Union (IoU) is another measure. It is a measurement based on Jaccard Index [93] which is a statistical formula for finding the similarity and diversity of sample sets as can be seen from the Figure 4.1 with formula given below:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (4.1)$$

Thus, the formula can be applied to the object detection task such as, where intersection is the overlapping area of the ground-truth  $B_{gt}$  with the predicted bounding box  $B_{pr}$  and union is the total area covering both of the boxes, as shown below:

$$J(B_{pr}, B_{gt}) = IOU = \frac{\text{area}(B_{pr} \cap B_{gt})}{\text{area}(B_{pr} \cup B_{gt})} \quad (4.2)$$

After obtaining the IOU value, it is compared with a predefined threshold value  $t$ . It is possible to assign the TP, FP and FN in the explained approach above to calculate the precision and recall results of the model by using the given equations below:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{AllDetections} \quad (4.3)$$

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{AllGroundTruths} \quad (4.4)$$

Once we compute precision and recall values, it is possible to plot the precision/recall curve and use the Area Under Curve (AUC) as a performance indicator. The precision/recall curve plotted according to different confidence values of the predicted bounding boxes.

It is also important to mention the  $mAP$ . It can also be considered as the AUC. In order to measure the accuracy of the model over all the classes of the dataset the mean AP used. The following formula can be used while obtaining the mAP value, where  $AP_i$  is the AP of the  $i$ th class and  $N$  is the total number of classes:

$$mAP = \frac{1}{N} \sum_{n=1}^N AP_i \quad (4.5)$$

These were selected as the initial measures due to the datasets' class imbalance such as the number of cars is larger than the number of awning-tricycles in VisDrone dataset. Therefore, it became important to measure the performance of the model also considering this challenge.

## 4.2 Datasets and Data Augmentation

Data augmentation can be considered as one of the first steps to increase the amount of the data where aim is to have more variance in the training dataset. In PyTorch [94] framework, there are multiple data augmentation techniques available. Some of the common data augmentation techniques from that framework are listed below:

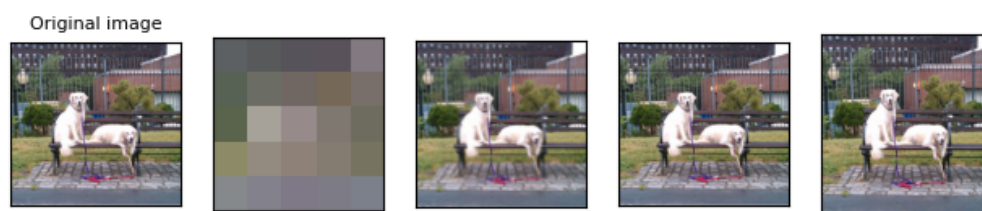
1. **Padding:** pads the input image all around 4.2a.
2. **Random Rotating:** rotates the input image by a random angle 4.2b.
3. **Resizing:** resizes the input image 4.2c.
4. **Vertical/Horizontal Flipping:** vertically and (or) horizontally flips the input image randomly with a given probability. 4.2d, 4.2e
5. **Five Crop:** crops the input image into five pieces including four corners and the central crop 4.2f.
6. **Gaussian Blur:** adds blur to the input image with randomly chosen Gaussian value 4.2g.
7. **Random Perspective:** performs a random perspective transformation on the given input image with a given probability 4.2h.
8. **Random Coloring:** randomly changes the brightness, contrast, saturation and hue of the given input image 4.2i.



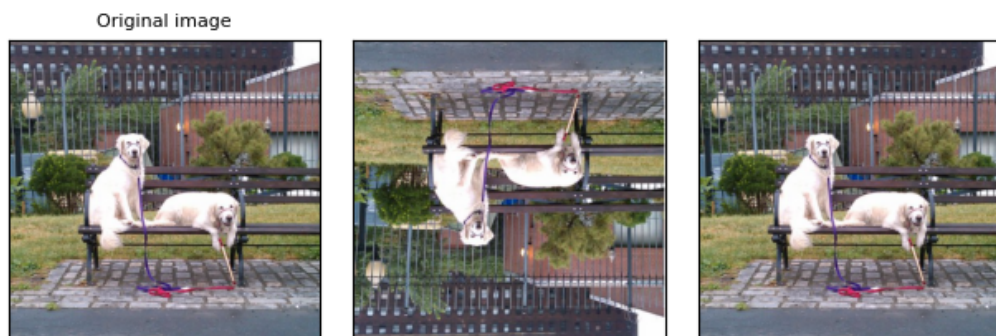
(a) Padding



(b) Random Rotating

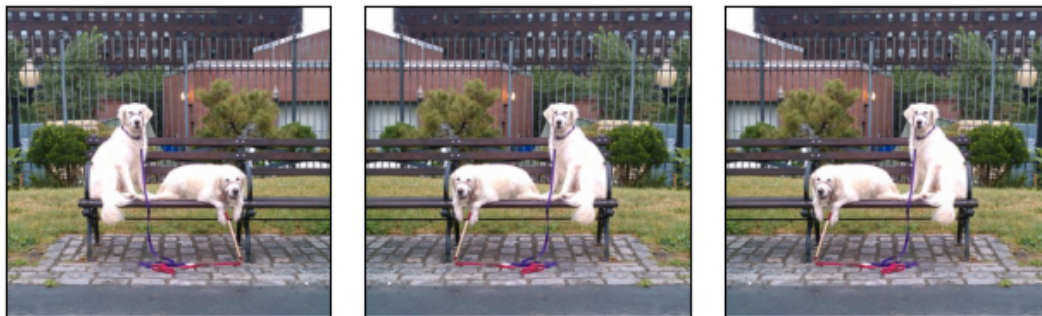


(c) Resizing



(d) Vertical Flipping

Original image



(e) Horizontal Flipping

Original image



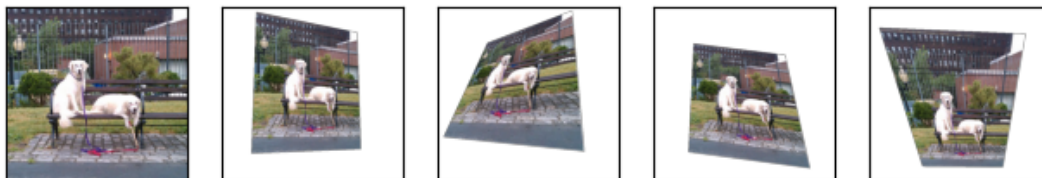
(f) Five Crop

Original image



(g) Gaussian Blur

Original image



(h) Random Perspective



(i) Random Coloring

Figure 4.2: Shows the sample outputs from various data augmentation techniques.

## 4.3 Experimental Methodology

The rest of this section summarizes the experiments which were held on the performances of applying data augmentation, studying the performance of the network with respect to using different anchor boxes, increasing the detection layers and adding vision transformers to the model.

### 4.3.1 The Effect of Using Data Augmentation

In addition to classical data augmentation techniques such as flip, crop and rotate, another data augmentation technique is also utilized in this work and we describe that technique here: Certain datasets (such as the VisDrone dataset) provide a class called ignored regions. Since the proposed model was expected to not learn from those parts, those ignored patches in a dataset are blackened as a part of data augmentation process (see Figure 4.3 for an example). This idea was also proposed in [74].A.24. The following table demonstrates the effect of the augmentation technique that was used. As it can be seen in Table 4.1, although in some classes the precision decreases, the overall mAP value increases when this augmentation technique is used.





Figure 4.3: Shows a sample image from VisDrone dataset. The black regions represent the ignored regions.

| Augmentation | Pedestrian (%) | Person (%)  | Bicycle (%) | Car (%)     | Van (%)     | Truck (%)   | Tricycle (%) | Awn (%)     | Bus (%)     | Motor (%)   | mAP (%)     |
|--------------|----------------|-------------|-------------|-------------|-------------|-------------|--------------|-------------|-------------|-------------|-------------|
| No           | 15.1           | 19.9        | 10.4        | 21.5        | 20.9        | 13.1        | 8.17         | 11.0        | 40.6        | 11.6        | 18.1        |
| Yes          | <b>19.8</b>    | <b>21.6</b> | <b>21.2</b> | <b>35.9</b> | <b>26.3</b> | <b>17.0</b> | <b>10.2</b>  | <b>11.6</b> | <b>46.8</b> | <b>19.9</b> | <b>29.9</b> |

Table 4.1: Shows the results of YOLODrone on VisDrone2019 dataset with and without using the data augmentation.

### 4.3.2 The Effect of Number of Anchor Boxes

Anchor boxes are important while scaling the predicted width and height to get a candidate bounding box in YOLOv3, as explained in Chapter 3. k-means clustering is used to find the anchor box values for VisDrone dataset. These anchor box pairs are scaled for each layer. The number of anchor box pairs are changed to be able to observe their effect on the performance of the YOLOv3 model.

An optimal number of anchor boxes for our dataset, the following width/height pairs, were obtained by k-means algorithm are used:

- 2 anchor box / layer (6 pairs): (7,16), (18,35), (37,44), (41,81), (77,100), (125,177)
- 3 anchor box / layer (9 pairs): (10,13), (16,30), (33,23), (30,61), (62,45), (59,119), (116,90), (156,198), (373,326)
- 5 anchor box / layer (15 pairs): (2,2), (3,3), (4,4), (6,12), (9,20), (17,28), (18,34), (30,48), (33,53), (49,79), (53,79), (80,117), (84,125), (125,209), (209,212)

Table 4.2 shows the results of the experiment on the number of the anchor box pairs.

| Anchor Boxes | Pedestrian (%) | Person (%)  | Bicycle (%) | Car (%)     | Van (%)     | Truck (%)   | Tricycle (%) | Awn (%)     | Bus (%)     | Motor (%)   | mAP (%)     |
|--------------|----------------|-------------|-------------|-------------|-------------|-------------|--------------|-------------|-------------|-------------|-------------|
| 6            | 21.6           | 19.5        | 4.60        | <b>44.4</b> | 18.5        | 9.50        | 4.00         | 7.90        | 32.1        | 19.6        | 21.6        |
| 9            | 19.8           | 21.6        | <b>10.2</b> | 35.9        | <b>26.3</b> | <b>17.0</b> | <b>10.2</b>  | <b>11.6</b> | 46.8        | <b>19.9</b> | 29.9        |
| 12           | <b>21.9</b>    | <b>46.8</b> | 7.71        | 43.7        | 21.9        | 15.7        | 8.10         | 8.86        | <b>47.9</b> | 18.6        | <b>36.3</b> |

Table 4.2: Shows results of using different numbers of anchor boxes in YOLOv3 Algorithm.

### 4.3.3 Varying the Total Number of Detection Layers

As mentioned in Chapter 3, YOLOv3 suffers when the size of the objects differ in a large spectrum, which is the natural case in many images acquired by the modern unmanned aerial vehicles. The scales of the objects in those datasets are typically smaller in size than the ones appear in ground taken datasets. In order to overcome this issue, YOLODrone architecture was developed. In YOLODrone architecture, the main focus was on the detection of the small objects in aerial datasets taken by drones or UAVs. In the preliminary results of training YOLOv3 model with VisDrone dataset, it has been observed that the performance of the model was lower when compared to the one which was trained on COCO dataset. Therefore, YOLODrone architecture was developed by keeping the strong parts of the YOLOv3 model, consequently, it is built on top of the YOLOv3 architecture.



YOLOv3 works in a way where the total number of used grid cells matter. As a part of the post-process, which uses the feature maps obtained from Darknet53 backbone, YOLOv3 splits the input image into grid cells. The total number of grid cells differs in each detection layer. Every grid cell is responsible for proposing candidate bounding box(es). An idea in YOLODrone is to increase the number of grid cells to split the input image into smaller pieces to be able to detect small objects. For that purpose, YOLODrone uses 5 detection layers as shown and explained in Figure 3.5. To be able to check the performance of this idea, we performed the following steps: (i) the original YOLOv3 model was trained with VisDrone dataset; (ii) the number of detection layers were incrementally increased, starting from 3 (the original number of detection layers in YOLOv3), first to 4 and then, to 5 (which is the total number of detection layers in YOLODrone). Table 4.3 summarizes the results of this experiment for each class of VisDrone2019 dataset for networks with 3, 4 and 5 detection layers, respectively.

| Detection Layers | Pedestrian (%) | Person(%)   | Bicycle(%)  | Car(%)      | Van(%)      | Truck(%)    | Tricycle(%) | Awn(%)      | Bus(%)      | Motor(%)    | mAP(%)      |
|------------------|----------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 3                | 19.8           | <b>21.6</b> | 10.2        | 35.9        | <b>26.3</b> | <b>17.0</b> | <b>10.2</b> | 11.6        | <b>46.8</b> | 19.9        | 29.9        |
| 4                | 23.7           | 20.2        | 7.10        | <b>43.3</b> | 22.9        | 12.6        | 9.30        | <b>11.7</b> | 44.7        | <b>22.9</b> | 37.0        |
| 5                | <b>25.1</b>    | 19.9        | <b>11.4</b> | 41.5        | 22.9        | 13.1        | 8.20        | 11.0        | 40.6        | 21.6        | <b>39.1</b> |

Table 4.3: Compare the test results of using 3 detection layers, 4 detection layers and 5 detection layers in the YOLO architecture which was trained on VisDrone2019 dataset.

As a result, the YOLODrone model was built from using the ideas of the best results from the experiments which were explained until here. In addition to using data augmentation and specifically obtained anchor boxes via k-means, we also used 5 detection layers in the architecture. The model was trained 300 more epochs with the selected hyperparameters. The results in Table 4.4 summarizes the class based accuracy and the total mAP values.

Table 4.4 summarizes the final results of using all those design ideas used in YOLODrone to YOLOv3 for each class. Also, the experiment was held to find the optimum threshold level for both IoU and confidence scores. The accuracy was better when the confidence threshold was reduced from 0.5 to 0.3. However, increasing the IoU threshold did not provide any benefits. Table 4.4 compares the best results of YOLODrone to the original YOLOv3. As shown in the table,

| Architecture     | Pedestrian(%) | Person(%)   | Bicycle(%)  | Car(%)      | Van(%)      | Truck(%)    | Tricycle(%) | Awn(%)      | Bus(%)      | Motor(%)    | $AP$ (%)    | $mAP_{50}$ (%) |
|------------------|---------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|----------------|
| YOLOv3           | 19.8          | <b>21.6</b> | 10.2        | 35.9        | 26.3        | <b>17.0</b> | 10.2        | 11.6        | 40.6        | <b>19.9</b> | 28.3        | 29.9           |
| YOLODrone (ours) | <b>22.4</b>   | 16.3        | <b>10.9</b> | <b>39.7</b> | <b>35.6</b> | 16.7        | <b>10.3</b> | <b>16.0</b> | <b>50.6</b> | 19.1        | <b>29.0</b> | <b>48.4</b>    |

Table 4.4: Comparison of YOLOv3 to YOLODrone on the visDrone2019 dataset (mAP values are shown for each class).

out of the listed 10 classes, YOLODrone yielded the best results for 7 classes. It also yielded the best  $mAP_{50}$  and  $mP$  value among YOLOv3.

Figure 4.4 and Figure 4.5 show results of both YOLOv3 and YOLODrone models on two image samples from the VisDrone dataset. As it can be seen from the Figure 4.4c, the performance of YOLODrone is better than YOLOv3 in terms of detecting small objects in the center of the image. This is also a very specific sample from the dataset, because the objects that are located in the bottom of the sample image are larger than and the ones that are located on the top of the image. This difference exists in most of the scenes in UAV datasets as explained before. In Figure 4.4b YOLOv3 is far away from detecting small objects that are located closer to the top. On the other hand, in Figure 4.4c YOLODrone successfully detected the object that YOLOv3 missed. Similarly in Figure 4.5, YOLODrone can successfully detect the smaller objects as Figure 4.4c shows, when compared to 4.4b. As a conclusion, not only quantitative but also qualitative results suggest YOLODrone can perform better in terms of detecting small objects in UAV images. The definition of the labels in the sample images are explained as captions below the images.



(a) Shows sample image with ground truth labels.



(b) Outputs of original YOLOv3 model.



(c) Outputs of the YOLODrone model.

Figure 4.4: Shows outputs from the proposed model. In Figure 4.4b red circles represent the objects that are missed by YOLOv3 and in Figure 4.4c green circles represent the objects that were successfully detected by YOLODrone model.



(a) Shows sample image with ground truth labels.



(b) Outputs of original YOLOv3 model.



(c) Outputs of the YOLODrone model.

Figure 4.5: Shows outputs from the proposed model. In Figure 4.5b red circles represent the objects that are missed by YOLOv3 and in Figure 4.5c green circles represent the objects that were successfully detected by YOLODrone model.

### 4.3.4 The Effect of Adding Vision Transformers

Because of the fast-paced improvements in the computer vision and deep learning areas, YOLOv5 was introduced to the computer vision community shortly after **YOLOv3**. Since the performance of YOLOv5 was better than YOLOv3, the scope of this thesis which is to detect the small objects in drone datasets was expanded to the YOLOv5 architecture as well.

YOLODrone+ architecture contains a vision transformer and is built on top of the YOLOv5 architecture. The details of the YOLODrone+ architecture was given in Chapter 3. To study the performance of YOLODrone+, the training was conducted on two different drone datasets. The hyper-parameters that were given in Table 4.5 are kept as controlled variables during all training and testing steps. Since all of the proposed methods offer changes in the architectural structure, these variables were kept constant so that the results could be compared experimentally.

| Hyper-parameter: | Epoch | Learning Rate | Batch Size | Confidence Threshold | IoU Threshold | Image Size |
|------------------|-------|---------------|------------|----------------------|---------------|------------|
| Value:           | 300   | 0.01          | 32         | 0.001                | 0.6           | 640        |

Table 4.5: Shows the names and the values of the hyperparameters used for training the models unless otherwise stated.

VisDrone [4] and SkyData<sup>1</sup> are two datasets which were used to train the proposed models in this thesis and the testing part of these datasets were used to compare the results of these proposed models. There are 25 sequences in the VisDrone dataset and 12 of these sequences were taken during the day, while the rest were taken at night. This difference provides deep learning models with more learning opportunities. There are 261,908 frames in total, and in this many frames all objects are marked and made open-source for object recognition problems. SkyDataV1 is a new dataset and offers computer vision challenges such as segmentation masks that are not available in VisDrone. SkyDataV1 can present a much larger number of small objects of inner city traffic objects such as pedestrians, cars, motors and buses in one frame than the VisDrone dataset does. This

---

<sup>1</sup><https://www.skydatachallenge.com>

results in creating more difficult problems. In the SkyDataV1 dataset, there are 10,360 frames made up of 7 different scenes and 3,695,245 marked bounding boxes in which 9 different classes are specified. The SkyData dataset is available as an open source public dataset.

As also mentioned in Chapter 3 YOLODrone+ is a hybrid model that includes both CNNs and Vision Transformers. In this way, the learning capacity of the model that proposes objects in this layer has been increased. The resulting YOLODrone+ architectural structure is shown in Figure 3.11. Compared to Figure 3.9 and Figure 3.11, YOLODrone+ has more parameters as given in Table 4.6 because there are more blocks in the spine of the architecture. While there are 3 detection blocks in the Neck of YOLOv5, this number is 5 in YOLODrone+ and a vision transformer has been added to the last layer.

To be able to test the performance of YOLODrone, the models which were given in Table 4.6 were also trained. These models are:

- YOLOv3: Original YOLOv3 [21],
- YOLOv5s: One of the smallest models introduced within YOLOv5,
- YOLOv5s-tr: the one where one vision transformer added to the neck part of the YOLOv5s model,
- YOLOv5s-5: YOLOv5s model with 5 detection layers,
- YOLODrone: our original YOLODrone architecture,
- YOLODrone+: Built on top of YOLOv5, contains 5 detection layers and one ViT in its last detection layer.

Table 4.6 summarizes the results of performing various experiments on those above-listed models. YOLODrone+ has more detection layers, consequently it makes more predictions (uses 5 detection layers instead of 3), and also includes a vision transformer layer. As the table shows, the YOLODrone+ model was able to obtain more successful bounding box predictions.



| Dataset     | Model             | N.Of Parameter | FPS   | AP%         | AR%         | AP50%       | Car(AP%)    | Person(AP%) | Bus (AP%)   | Motor (AP%) |
|-------------|-------------------|----------------|-------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| VisDrone-19 | YOLOv3            | 61,949,149     | 12,69 | 28.3        | 19.8        | 28.9        | 35.9        | 21.6        | 40.6        | 19.9        |
|             | YOLOv5s           | 7,235,389      | 76,34 | 31.1        | <b>25.5</b> | 22.6        | 62.5        | 23.2        | 13.0        | 41.6        |
|             | YOLOv5s-tr        | 7,235,645      | 72,99 | 33.6        | 25.3        | 22.9        | 62.6        | 22.1        | 20.0        | 40.9        |
|             | YOLOv5s-5         | 144,281,723    | 4,48  | 37.9        | 21.1        | 20.4        | <b>66.9</b> | <b>33.3</b> | 21.6        | <b>45.6</b> |
|             | YOLODrone         | 61,959,200     | 11,57 | 29.9        | 19.3        | <b>48.4</b> | 39.7        | 16.3        | <b>50.6</b> | 19.4        |
|             | <b>YOLODrone+</b> | 154,646,139    | 3,19  | <b>41.9</b> | 20.9        | 29.0        | 61.8        | 27.1        | 20.3        | 42.9        |
| SkyData-v1  | YOLOv3            | 61,949,149     | 12,69 | 49.6        | 30.1        | 32.8        | 32.1        | 25.6        | 46.8        | 53.0        |
|             | YOLOv5s           | 7,235,389      | 76,34 | 52.7        | <b>35.4</b> | 38.1        | 33.7        | 25.1        | 48.2        | <b>57.2</b> |
|             | YOLOv5s-tr        | 7,235,645      | 72,99 | 46.0        | 33.4        | 33.9        | 28.5        | 22.6        | <b>48.4</b> | 51.2        |
|             | YOLOv5s-5         | 144,281,723    | 43.1  | 4.48        | 32.9        | 35.3        | 28.8        | 21.4        | 19.7        | 36.5        |
|             | YOLODrone         | 61,959,200     | 11,57 | 50.9        | 30.5        | 36.0        | 32.7        | 23.4        | 47.0        | 52.0        |
|             | <b>YOLODrone+</b> | 154,646,139    | 3,19  | <b>56.0</b> | 32.7        | <b>38.2</b> | <b>38.8</b> | <b>28.9</b> | 43.5        | 45.4        |

Table 4.6: In order to understand how the architectural structures perform on the dataset, both the accuracy score and the sensitivity score are shown for both SkyDataV1 and VisDrone datasets on all models. The last four columns provide class-based results for the sample classes.

In the experiments, average precision scores were used as the main performance metric. In addition, the average precision scores of the common classes in both datasets are given in Table 4.6. This experiment has different results as the two datasets differed in both object diversity and number of objects. However, as it can be observed from the table, an increase of up to 10.8 for VisDrone dataset and 3.3 for SkyData provided with the proposed improvements, especially when compared to YOLOv5s and YOLODrone+ average accuracy score metric ( $AP\%$ ). As it can also be seen from the following Figure 4.6 YOLODrone+ model can successfully detect the small objects; the YOLOv5s model missed the objects which are standing next to large boats. With the help from self-attention mechanism from vision transformers: The small objects standing next to larger objects can be detected as well.



(a) Shows sample image with ground truth labels.



(b) Outputs of original YOLOv5s model.



(c) Outputs of YOLODrone+ model.

Figure 4.6: Shows outputs from the proposed YOLODrone+ model. In Figure 4.6b red circles represent the objects that are missed by YOLOv5s and in Figure 4.6c green circles represent the objects that were successfully detected by YOLODrone+ model.



## Chapter 5

# Discussion and Concluding Marks

Object detection is an important task and many real-world applications of the computer vision rely on object detection. Examples to such real-world applications are: object tracking, activity detection, human counting, sign detection for autonomous vehicles. Existing deep learning based solutions can provide sufficient performance for the object detection problem on many datasets; however, when it comes to UAV-taken datasets, the performance of the existing algorithms drops and the algorithms need a set of adjustments to obtain higher performance for UAV-taken images. YOLO family is one such popular solution used in many ground taken image sets. In terms of improving the performance of YOLOv3 on UAV datasets, as a part of this thesis, YOLODrone architecture was proposed. We observed that increasing the number of detection layers helps the model to consider the small objects more effectively. In our experiments, we observed that the difference between original YOLOv3 and YOLODrone was 18.5 in ( $mAP\%$ ) and was 0.7 in ( $AP\%$ ) on the VisDrone dataset.

For a long time, YOLOv5 was used as the state of the art YOLO iteration until those versions were proposed. Consequently, as a part of this thesis, we were able to study the most predominant YOLO iteration: YOLOv5. Combining the improvements introduced in YOLOv5 and the findings that were obtained from YOLODrone, YOLODrone+ was proposed as a part of this thesis. In addition to increasing the total number of detection layers as an addition, Vision Transformers were also added to the last detection layer in the YOLODrone+ architecture. The experiments to understand the performance of YOLODrone+ yielded an increase of up to  $10.8AP\%$  AP for VisDrone dataset and  $3.3AP\%$  for SkyData, when compared to YOLOv5s.

| Dataset  | Model      | AP%         | AR%         | AP50%       |
|----------|------------|-------------|-------------|-------------|
| VisDrone | YOLOv3     | 28.3        | 19.8        | 28.9        |
|          | YOLODrone  | 29.9        | 19.3        | <b>48.4</b> |
|          | Yolov5s    | 31.1        | <b>25.5</b> | 22.6        |
|          | YOLODrone+ | <b>41.9</b> | 20.9        | 29.0        |
| SkyData  | YOLOv3     | 49.6        | 30.1        | 32.8        |
|          | YOLODrone  | 50.9        | 30.5        | 36.0        |
|          | Yolov5s    | 52.7        | <b>35.4</b> | 38.1        |
|          | YOLODrone+ | <b>56.0</b> | 32.7        | <b>38.2</b> |

Table 5.1: Shows the performance comparison of the proposed models for conclusion.

The performance of the proposed algorithms were tested on two aerial datasets which are acquired by drones: VisDrone and SkyData datasets. As the new YOLO versions were introduced to the literature, we followed them (as stated previously, we studied both YOLOv3 and YOLOv5 versions in this thesis) and improved them for better object detection in aerial image datasets as Table 5.1 suggests. The ( $AP\%$ ) is increased from 13.6% in VisDrone and 6.4% in SkyData after all the improvements proposed in this thesis. While different versions of YOLO (such as YOLOv8 [95] and YOLOv7 [96]) have also been proposed recently, at the time when the experiments of this thesis were concluded, YOLOv5 was the most recent version and consequently, YOLOv8 and YOLOv7 were not included in this thesis. As a part of future work, a special loss function to learn more from the small objects in drone datasets can be tested on YOLOv7. Moreover, object detection is an important part of many multi-object tracking algorithms using the tracking-by-detection approach. The relation between the performance of multi-object tracking and the accuracy of object detectors were given in Appendix C.

# Appendix A

## Object Detection Datasets

To train a deep learning model for object detection with high accuracy, a high-quality dataset is crucial. Deep learning models are often used to solve computer vision problems, but they can be time and data-intensive. Luckily, there are several datasets proposed for object detection in the literature. Those datasets can be broadly classified as: (i) datasets taken by Drones; and (ii) datasets taken from the ground.

## A.1 Datasets That Are Not Obtained by Drones

Although the scope of this work does not include object detection from the ground taken images, the popular and ground-taken datasets can provide a good starting point for building an architecture to detect objects. Therefore, understanding and explaining widely used datasets, even if they include images taken from ground vehicles, is important. Below, we summarize the most commonly used such ground-taken datasets used in many object detection applications.

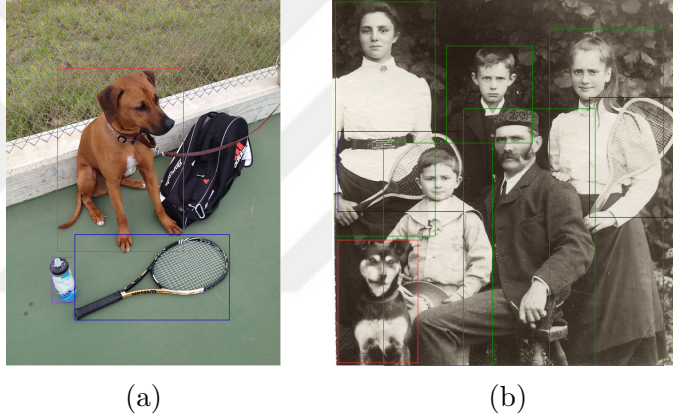


Figure A.1: Sample images from the COCO dataset [2]. As can be seen in both Figure A.1a and Figure A.1b the classes include most of the common objects including a tennis racket and the way the images are obtained vary. Therefore, the lighting and the angle is different as well as the size of the objects.

**MS-COCO** [2] is one of the most challenging, publicly available datasets. It contains annotations for various vision tasks including object detection. Therefore, MS-COCO dataset evolves over the years. The MS-COCO-17 version (from 2017) contains approximately 164k images and approximately 897k annotated objects from 80 different classes. Also, when compared to other publicly available datasets such as PASCAL VOC [5] and ILSVRC [6], MS-COCO has smaller objects whose area is smaller than 1% of the image. Just like ImageNet was an important dataset for classification, MS-COCO has become the de facto standard for the object detection community. Figure A.1 shows two samples from the MS-COCO dataset.

**PASCAL Visual Object Classes (VOC)** [5] includes multiple tasks including image classification, object detection, semantic segmentation and action detection. VOC12, which is the most commonly used version which consists of 11k images with 27k annotated objects of 20 different classes of common real-life objects such as person, animals, vehicles, indoor objects. Figure A.2 shows two samples from the PASCAL VOC dataset.



(a)

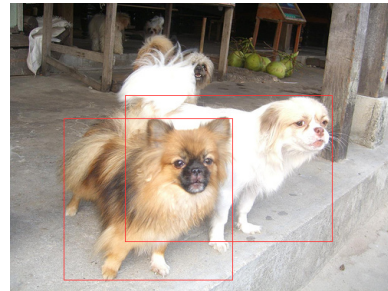


(b)

Figure A.2: Sample images from the PASCAL VOC [5]. Both Figure A.2a and Figure A.2b show different classes.



(a)



(b)

Figure A.3: Sample images from the ILSVRC dataset [6]. The images are obtained from ImageNet however, the annotations are re-labeled for object detection task.

**ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** [6] was annual challenge between the years: 2010 and 2017. The ILSVRC dataset has 200 unique classes and the average number of total object number per image was two times larger than PASCAL VOC. ILSVRC-14 contains over 517,000 images with approximately 534,000 annotated objects.

**KITTI** [3] is one of the most popular public dataset on multi-object tracking tasks. While this dataset is not directly used for object detection, since it also includes bounding box information for each object to be tracked, it can also be used for object detection algorithms. This dataset was recorded from a platform which was moving on the ground while driving across Karlsruhe, Germany. An image of the hardware used to collect KITTI dataset is shown in Figure 2.6. The multi-object tracking benchmark of KITTI has 21 training and 29 test sequences. As it was also mentioned in the [97], the dataset was labeled according to 8 different classes however only “car” and “pedestrian” classes are evaluated. While collecting the data, the recorded images are stored with loss-less compression and the raw image edited by cropping the sky and engine hood of the car.



(a)



(b)

Figure A.4: Shows two sample images from KITTI [3] dataset. As also explained in the beginning of this section in order to obtain the KITTI dataset a car is used as shown in Figure 2.6. Therefore, the angle of the way the footage is obtained, is the same which provides an advantage in training.

**MOT Challenge** provides annotated datasets and metrics to measure the performance of the multi-object tracking methods and has been one of the top datasets. Until now there has been 3 different release of the dataset and for every version both the number of sequences and annotations were increased. Since multi-object tracking is a very important and crucial task of the scene understanding, the methods which are developed in order to solve this problem would be crucial to take further steps in computer vision. Moreover, the number of dataset to train the state-of-the-art deep learning methods are lack in number. With the first release of this dataset MOT15 [98], there are 22 sequences of 996 seconds of data were available. Since the dataset is public to this day it is one of the most popular datasets existing in the literature. MOT20 [7], being the most recent version, introduces 8 novel sequences of video which includes 3 very crowded scenes.



(a)



(b)

Figure A.5: Shows two samples from MOT [7] dataset. Figure A.5a shows a recording during the night time and Figure A.5b shows the footage obtained during day time.



## A.2 Datasets That Are Obtained by Drones

Since the focus of this thesis is object detection in drone-taken aerial datasets, in this part of the thesis, we describe such datasets in detail.

**UA-DETRAC** dataset [8] is one of the most popular multi-object tracking and detection benchmarks that consists of almost 10 hours of video recordings which are obtained from a camera. There are a total of 100 video recordings with almost 140,000 frames of the challenging real-world traffic recordings. The videos are labeled manually considering useful attributes such as illumination of scenes, occlusion and the type of the vehicle. The main contribution this dataset offers to the artificial intelligence community is that the annotated objects mostly consist of vehicles in crowded traffic scenes. Also, it is possible to find the videos which are recorded in different weather conditions such as Sunny, Cloudy and Rainy in different times of the day including night time.

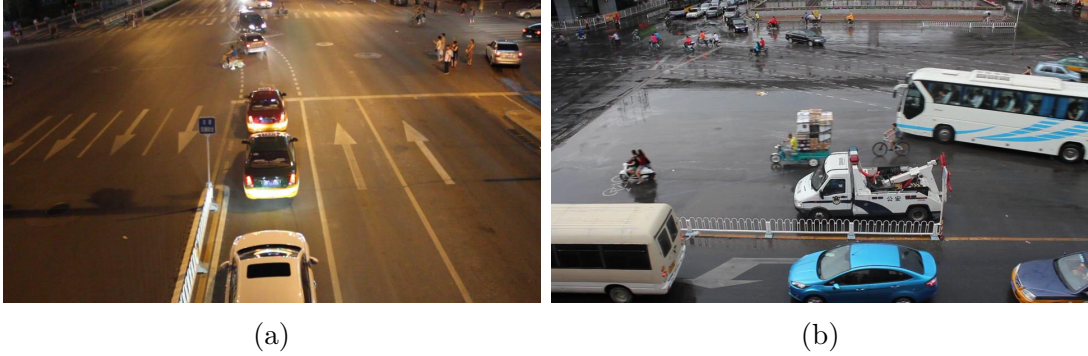


Figure A.6: Figure A.6a and Figure A.6b UAV Detrac [8] shows two samples from UA-DETRAC dataset. These two figures represent the variant nature of the UAV datasets. Although most of the objects are similar in these figures, the difference in their angle makes the UAV datasets challenging.



**Stanford Drone** Dataset [9] consists of 8 different scenes with 6 different classes such as Bicyclist, Pedestrian, Skateboarder, Cart, Car and Bus. The total number of videos in this dataset is 60. The scenes are from the university campus, therefore, the total number of pedestrians and the total number of bicyclists are far more than the total number of vehicles with motors. The dataset was recorded with a 4K camera on an unmanned aerial vehicle with 80 meters flight height. Therefore, the objects in the dataset can easily be considered as small objects.

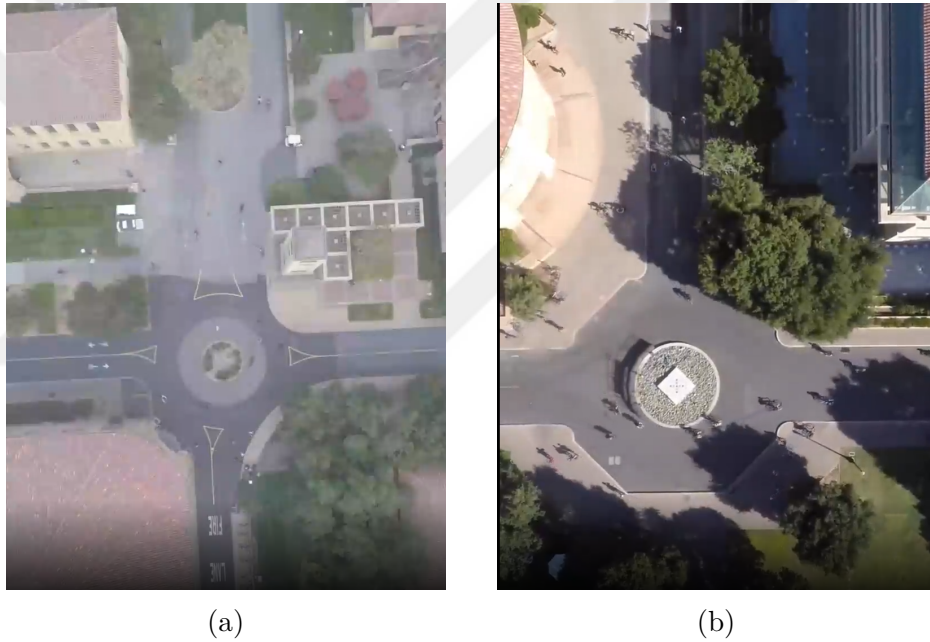


Figure A.7: Two sample images from the Stanford dataset [9] are shown. As it can be seen from these two samples, the recorded objects are quite small.

**UAV123** [10], there are 123 sequences of videos which are recorded from an UAV. In order to create this dataset, the authors used 3 different drones. The objects in the dataset vary in size. Moreover, the recorded frames also vary in different scenes and the subset of this dataset UAV20 is specifically created for long term tracking which is an important problem in computer vision problems.



(a)



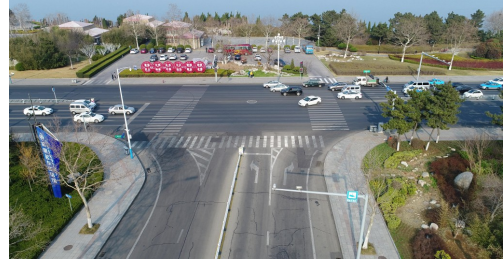
(b)

Figure A.8: Two samples from UAV123 [10] drone dataset. Figure A.8a is a car and Figure A.8b is a wake-board. This dataset is designed for single-image tracking. However, because of the variety in class, it can also be used for object detection.

**Unmanned Aerial Vehicle Benchmark Object Detection and Tracking (UAVDT)** [11] dataset is a large scale and created in order to solve 3 important problems such as object detection, single-object tracking and multi-object tracking. The total number of videos are 10 and the total duration of them is 10 hours with 80,000 labeled frames. The dataset was introduced in 2018 and as also explained in the benchmark paper, 14 different attributes are also labeled while creating the dataset. The variety of the scenes also covers all the challenges of the UAV datasets which can be summed as the high density, small object, camera motion and real time tracking as also mentioned in the benchmark paper.



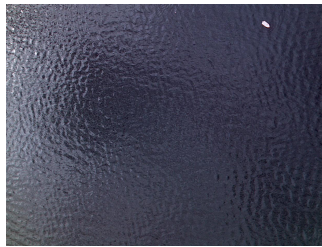
(a)



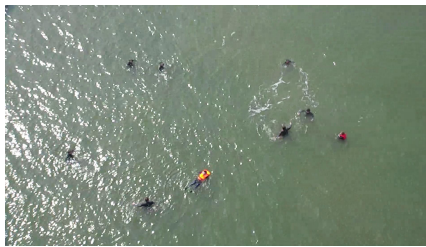
(b)

Figure A.9: Sample figures from UAVDT dataset [11]. As can be seen from Figure 2.7 it has similar characteristics with VisDrone dataset.

**SeaDronesSee** [12] is unique because most of the object detection datasets obtained from drones have traffic footage of land vehicles. Unmanned Aerial Vehicles (UAVs) have a crucial role in rescue missions especially in maritime environments because of their agile and fast nature. Therefore, in order to use drones in maritime rescue missions this dataset was developed. This object detection and tracking benchmark dataset named as SeaDronesSee introduced. In this dataset there are over 54,000 annotated frames with 400,000 instances which were captured from various altitudes (from 5 to 260 meters) and angles (0 to 90 degrees). Also meta data such as altitude and angle are provided along with the object annotations.



(a)



(b)

Figure A.10: Shows two samples from an unique UAV dataset SeaDronesSee [12]. As can be seen from the given figures the objects are so small that they look similar therefore, requires very careful architectural design of deep learning models.

In addition to all these datasets, there is also a recent drone-based dataset proposed and it is called SkyData <sup>1</sup>. It will be explained it in next chapter.

<sup>1</sup>([www.skydatachallenge.com](http://www.skydatachallenge.com))

# Appendix B

## Data Labeling

As also mentioned in Chapter 2, the datasets to solve multi-object tracking problems can also be used for object detection tasks. The number of datasets obtained by UAVs are less than the ground taken ones. In order to contribute to this area, SkyData, an aerial dataset, was introduced to train novel deep learning models and test their performances.

SkyData comes with object segmentation and object detection annotations. Once we have segmentation annotations, we can also have the detection annotations in a trivial manner. Generating such segmentation annotations can be challenge for a new dataset. Therefore, in this part of the thesis, we provide the details of using a GUI based annotation tool: CVAT [99].

## B.1 SkyData Dataset

SkyData is a state-of-the-art benchmark dataset which contains multiple videos. The image frames from those videos can be used to train algorithms for many computer vision tasks such as object segmentation, object detection and multi-object tracking including object detection. The dataset contains videos from different scenes. During the labeling process, these scenes are divided into smaller videos so that the annotation process can be handled easily by multiple people simultaneously.

A summary of the current version of the SkyData (SkyDataV1) is given in the table below:

1. **Scene Name:** names of videos from Scene 1-7.
2. **Total Number of Frames:** total number of frames.
3. **Training Range:** range of the video belongs to the training set.
4. **Testing Range:** range of the video belongs to the testing set.
5. **Part of the Scene Name:** names of the divided parts.
6. **Frame Range:** range of the part of the scene

| SkyData Version 1 |          |                           |              |                 |              |
|-------------------|----------|---------------------------|--------------|-----------------|--------------|
| Recorded Scenes   |          |                           |              | Separated Parts |              |
| Scene Name        | # Frames | Train Range               | Test Range   | Part Name       | Frame Range  |
| Scene 1           | 14652    | [2000, 2299]              | N/A          | scene1_part1    | [2000, 2299] |
| Scene 2           | 8933     | [1, 3000]                 | [3001, 4000] | scene2_part1    | [1, 2000]    |
|                   |          |                           |              | scene2_part2    | [2001, 4000] |
| Scene 3           | 10413    | [1, 2470]                 | [4001, 4490] | scene3_part1    | [1, 2500]    |
|                   |          |                           |              | scene3_part2    | [4001, 4500] |
| Scene 4           | 9051     | [1, 560],<br>[1000, 1689] | [2000, 2199] | scene4_part1    | [1, 1000]    |
|                   |          |                           |              | scene4_part2    | [1001, 2000] |
|                   |          |                           |              | scene4_part3    | [2001, 3000] |
| Scene 5           | 1650     | [1, 1065]                 | [1066, 1650] | scene5_part1    | [1, 1650]    |
| Scene 6           | 9289     | N/A                       | N/A          | N/A             | N/A          |
| Scene 7           | 3693     | N/A                       | N/A          | N/A             | N/A          |

Table B.1: Shows various details of the first version of the SkyData dataset. Source: [www.skydatachallenge.com](http://www.skydatachallenge.com)

Annotations of the SkyData Version 1 are saved in JSON format and although the test set is hidden because SkyData is a benchmark dataset, the training set is publicly available. Additionally, SkyData also contains unlabeled images, therefore SkyData can be used for unsupervised or semisupervised learning as well. Currently, SkyData consists of 5 different scenes with 9 sequences of frames as it can be seen from Table B.1. In the dataset, there are 9 classes which are: people, bicycle, motor, pickup, car, van, truck, bus and boat. The data is collected with an on-board UAV camera in the region of Üsküdar/İstanbul during day time. As it can be seen from the Figure B.1 and from Table B.2, the videos are taken at various altitudes. Also, it is important to mention that the recording was completed during the COVID-19 pandemic in 2020 and because of that, almost all of the pedestrians are wearing face masks. For further details, please refer to <https://www.skydatachallenge.com>.

| SkyData Challenge |          |          |              |
|-------------------|----------|----------|--------------|
| Sequence Name     | Duration | Altitude | Total Frames |
| Scene1            | 08:08.89 | N/A      | 14652        |
| Scene2            | 04:58.23 | 119m     | 8933         |
| Scene3            | 05:47.61 | 69m      | 10413        |
| Scene4            | 05:02.00 | 69m      | 9051         |
| Scene5            | 00:54.58 | 48m      | 1650         |

Table B.2: This table shows additional information for each scene of the SkyData. each scene's name, its duration, altitude and the total number of frames of five scenes from the SkyData dataset are provided. Source: [www.skydatachallenge.com](http://www.skydatachallenge.com)





(a)



(b)



(c)



(d)



(e)



(f)



(g)



(h)

Figure B.1: Raw and annotated sample images from the SkyData Challenge are visualized. The left-hand side shows the original images and the right-hand side shows the objects with their bounding boxes on the original image. Image source: [www.skydatachallenge.com](http://www.skydatachallenge.com)

## B.2 CVAT

Annotated data is the vital part for any type of supervised learning solution. CNNs of Deep Neural Networks provides solutions for most of the computer vision tasks. CNNs are trained by using annotated images and videos. Annotating a dataset is a very challenging and time-consuming process. More than three thousand hours of Intel's annotation team spend to annotate more than seven hundred thousand objects for just one of the computer vision solutions, according to the [100]. Therefore, it can be said that creating datasets is a research area of its own to find more efficient ways and to develop more user friendly tools for annotating the images or frames.

As part of this thesis, an open source online annotation tool Computer Vision Annotation Tool (CVAT) [99] was also studied. CVAT helps to increase the performance of the data annotation process of images and video frames. Also, the privacy policy of various organizations may not give access to their data or to their annotations. Consequently, annotating your own dataset for a particular task could come handy in many situations as well.

Generally, there are two ways for computer vision researchers to obtain a new dataset:

1. With a third-party company which specialized in annotation. This option may cause a problem in data validation and thus increase the total time to spend on annotation.
2. With an internal data annotation team which may make it easier to come to a balance between quality of the work and time spent on annotation.



In the annotation process of SkyData dataset the second option was selected and an internal team was created. Computer Vision Annotation Tool (CVAT) was used to annotate SkyData. It is possible to use the tool both online and offline. As also mentioned in the official github page<sup>1</sup>, the developers of CVAT developed the user interface part by considering the feedback received from many dataset annotation teams. For further details related to this open source tool, please refer to <https://github.com/opencv/cvat>.

One of the best parts of this tool is that it makes it possible to dump the annotations in COCO, Pascal VOC and ImageNet format. Therefore, without any further processing it is possible to use the dumped annotations in object detection models which were specifically developed with these datasets. The following images show the steps that were taken during the labeling process.

After logging in with credentials on its main page, a user should select the task and then the job. As a result, the screen should be similar to Figure B.2.

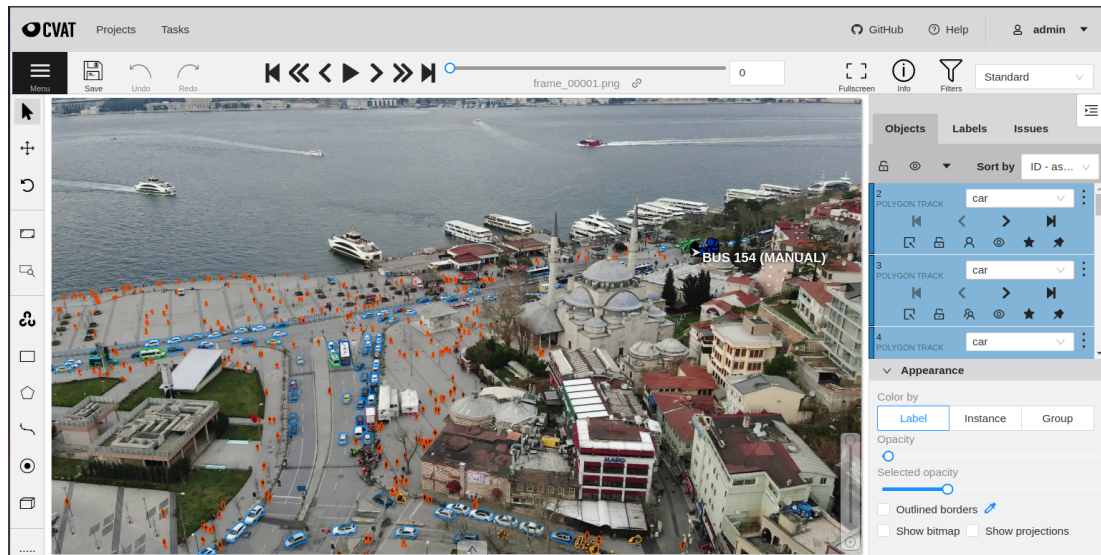


Figure B.2: The main screen after selecting the task and the job that needs to be annotated.

<sup>1</sup><https://github.com/opencv/cvat>

As also mentioned in one section above, the objects in the video are so small. Therefore, it is important to zoom in to the region which includes the objects that are supposed to be labeled. The zoomed-in version of the scene is given with Figure B.3.

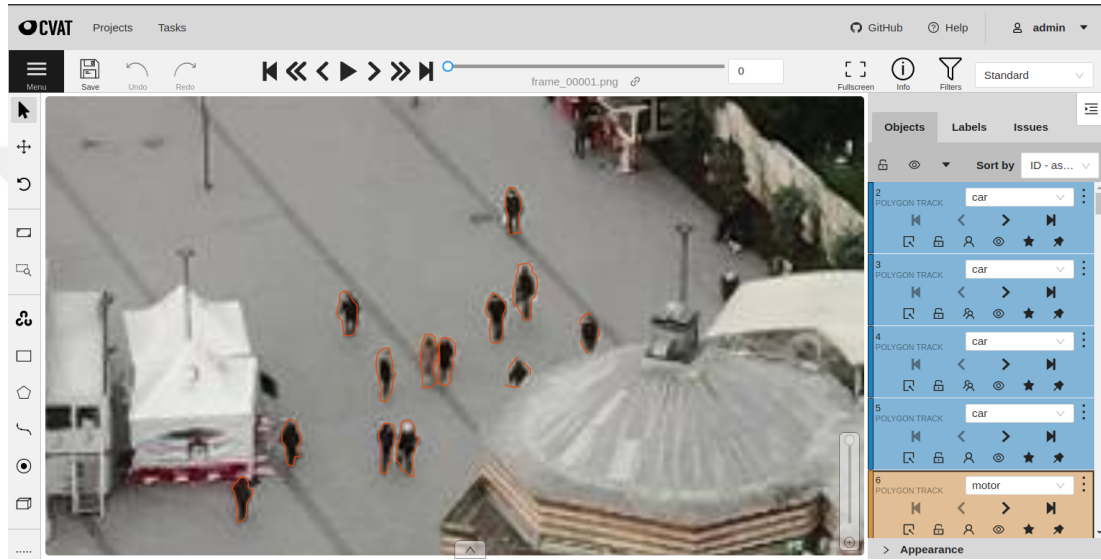


Figure B.3: The zoomed in version of the main screen.

The tools of CVAT which were used while labeling SkyData are given in Figure B.4. The description of the tools is as follows:

- (A) Save icon, to save the labeled frames or images.
- (B) A set of arrows to move between the frames. Single arrows are moving forward or backward for one frame. Double arrows are for moving forward or backward for 10 frames. The number of frames to be skipped with a double arrow can be changed from the settings.
- (C) Shows the current frame number.
- (D) Helps to see all the objects and operate on them specifically. The color of the object is based on class. The class motor is orange while the class car is blue in Figure B.3.
- (E) Helps us to scroll through the image. Since the resolution of the recorded footage is very high in SkyData, this was very useful while double checking the labeled objects.

(F) To label polygon objects. SkyData labeled only considering instance segmentation therefore, rectangles were not used to label the frames.



Figure B.4: The red box shows the number of the frame which is opened for annotation.

The following figures shows how the labeling with CVAT works in video frames. Figure B.5 shows one of the regions of the image and as it can be seen from the figure the objects in the frame are labeled. Then we proceed to the next 10 frames with the tool set (B) shown in Figure B.4, to obtain Figure B.6.

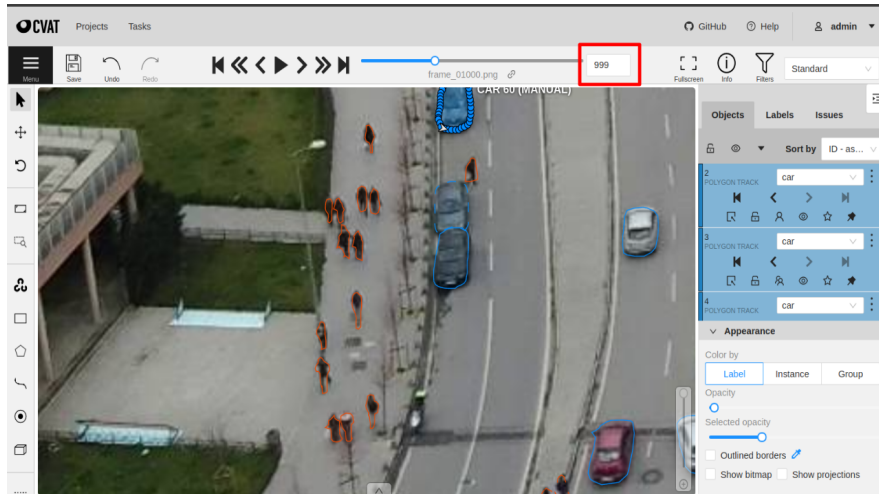


Figure B.5: The red box shows the number of the frame which is opened for annotation.

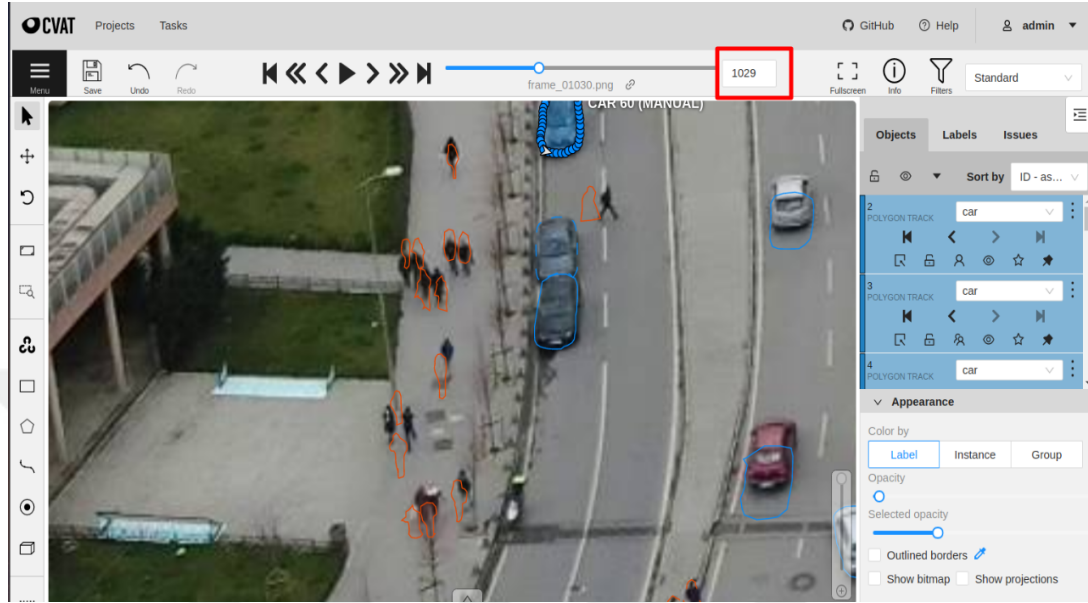


Figure B.6: The red box shows the number of the frame which is opened for annotation.

After Figure B.6 is obtained, then by using (E) as shown in Figure B.4, the mismatched masks around the objects were relocated according to their new location. The best part of CVAT is that, CVAT was able to move the relocated label between the initial position and the new position while considering the skipped frames. The labeling of the SkyData dataset was completed by following the given steps.

# Appendix C

## Multi-Object Tracking

This chapter provides a review of the recent research and studies on multi-object tracking topics. The aim of the multi-object tracking task is to solve the correspondence (association) problem. Plenty of methods have been proposed for multi-object tracking challenges during the last decade. Although the recent studies provide promising results on the task, multi-object tracking has been a challenging task to solve in computer vision for a very long time. There are many deep-learning based and non-deep learning based solutions available in the literature to solve this challenge.

Many solutions approach the MOT problems by using a tracking-by-detection method. The idea behind the tracking-by-detection approach, also known as separate-trackers, is that they perform detection per every frame first and treat the MOT task as an association problem. This method can take the advantage of high-accuracy of the object detection models such as YOLO Family. Thus, tracking-by-detection based solutions have gained favor due to the excellent performance of object detectors. Therefore, in this chapter of this thesis, the relation between the performance of YOLO based object detection models and multi-object tracking performance were investigated.

## C.1 Deep Learning Embedded MOT

Many researchers used a tracking-by-detection approach to provide solutions for the MOT task. Some of the most popular and recent trackers are ByteTrack [101], DeepSORT [102] and StrongSORT [103]. Since the baseline solution that was picked is DeepSort in this part, a whole section is devoted to explain that algorithm in detail. Some of the important deep learning based solutions are Deep Learning Embedded MOT methods which have the training step to learn about which detection belongs to which object. DeepSort is one of the most popular trackers that belongs to this category. Here, a deep metric learning model is trained where motion features are fused with appearance features to achieve this goal.

Most of the time DeepSORT uses a single-object tracking problem. In order to track multi-class objects, a Multi-class DeepSORT (MCD-SORT) tracker [104] is used. In this improved version appearance and motion features are used in order to associate the detected objects to a tracklet. Also, this association of object is restricted only within the same class. Thus, the performance in multi-class tracking improved. Classical tracker DeepSORT is revisited in StrongSORT [103] and upgraded in terms of detection, embedding and association. Also, with this paper two plug-and-play algorithms proposed to improve tracking results AFLink and Gaussian-smoothed interpolation (GSI).

Another popular method which trains DNNs for tracking is Siamese Network [105] where quadruplets of input image patches are fed to Siamese Network. Then, distances are measured between these patches. The output provides a ranking of these distances. Another approach in training is to learn the distance metric between two previously tracked objects as suggested in [106]. Here, the model can extract the features as set from the previously tracked objects. After, the model can use these feature sets to feed a Gated Recurrent Unit (GRU) Network [107]. Based on the distance between GRU Network’s outputs, several candidate tracklets are generated to connect to the previously tracked objects by using similarity between the local and global features.

With trained DNNs, the object trackers are also capable of learning the discriminative features in frames. Since DNNs can train to learn the discriminative features, this information can be used to solve tracking tasks. The algorithms which use this information are also developed and can be grouped into Deep Learning Embedded MOT. These algorithms are looking for the best locations in the following frames to associate the detections with the previously tracked objects. A particle filtering architecture is introduced for MOT tasks where two CNN classifiers are used [108]. Features which are coming from different layers of CNN are fed to classifiers. Training a DNN as a part of MOT framework using object-trackers is also used where the tracker is looking for the best image patch [109]. In order to handle occlusion, spatial features are eventually learned based on the visible map using the convolution and fully connected layer. The top feature of this approach is it considers occlusion features for tracking.

Detection based deep features plays a critical role in data association which is the second part of tracking-by-detection MOT solutions. The high-performance detection and deep learning based appearance feature are explored and introduced by a novel tracker named Person of Interest (POI) in [110]. For detection part a Faster R-CNN module fine tuned from VGG-16 while using additional training data. The Kalman Filter [111] used for predicting the motion and for associating the objects Kuhn-Munkers Algorithm [112] is used. Outputs from detection and tracking are used to handle the unreliable detection to complement tracking in different cases[113]. Detections with high confidence scores prevent tracking from having drifts and predictions of tracks are able to handle the noisy detection results which can be caused by occlusion. Also, a scoring function on CNN model was introduced to share most of the computations on the entire image. Similarly, a MOT architecture with high learning capacity to make use of long and short cues for handling real-life cases in [114]. Short-term cues are used to find false negatives, while long-term cues are used to avoid critical mistakes when occlusion happens. For a better association a switcher-aware classification (SAC) is used as well.

## C.2 Tracker Algorithm: DeepSort

DeepSORT [102] is a multi-object tracking algorithm. It works by assigning an ID to each object in a frame. DeepSORT is developed as an extension of the SORT (Simple Online Realtime Tracking) algorithm [115] by introducing deep learning approach into the SORT by adding an appearance descriptor layer to reduce identity switches, thus, the tracking becomes more efficient. To understand DeepSORT, it is important to understand SORT algorithms where solutions such as Kalman Filter [111] and Hungarian Algorithm [112] are used as a part of the solution. The SORT algorithm has 4 main parts as follows:

1. **Detection:** is the initial step in DeepSort where an object detector (such as any version of YOLO) can be used to detect the objects that the tracker is expected to track. Then the detected objects are passed to the next layer.
2. **Estimation:** the detections are propagated from the current frame to the next one where the position estimation of the target object is completed by using “Constant Velocity Model”. After a target is associated with a detection, the target state is updated with the detected bounding box, where components of velocity are solved with the “Kalman Filter”.
3. **Data association:** a cost matrix is computed with the intersection-over-union (IoU) difference between detected and predicted bounding boxes and it is solved by using “Hungarian Algorithm”. If IoU of detected object and target object is less than a threshold then, that particular assignment is rejected. This method solves the occlusion problem of the video footage and helps to maintain IDs of the objects.
4. **Creation and Deletion of Track Identities:** if the intersection over union (IoU) value for two objects is less than a specified threshold, then the objects are considered to be separate and are given different IDs. If one of these objects is not detected for a certain number of frames, it is considered to be “lost” and its tracking is terminated. If the object reappears in a later frame, it will be given a new ID and its tracking will resume. This process allows the algorithm to accurately track objects even when they are temporarily occluded or leave the field of view.



SORT has high precision performance however, it returns tracking predictions with ID switches and the algorithm is likely to fail in occluded scenes because of association matrix. DeepSORT has a better performing association metric which is a combination of both appearance and motion descriptors. DeepSORT is a multi-object tracking algorithm that tracks objects based on their velocity, motion and the appearance which is provided by the CNN layer.

The CNN part is responsible for discriminating feature embeddings which are an output of the trained CNN model. That CNN model was trained on a person re-identification datasets and to train the deep association metric, cosine learning approach is used in DeepSORT. “Cosine Distance” considers the appearance information which is useful to recover identities after long-term occlusions when motion is less discriminative because cosine distance is a similarity function that helps the model to recover identities in occlusion scenes where motion estimation fails. The following image C.1 shows the whole process of multi-object tracking with DeepSORT.

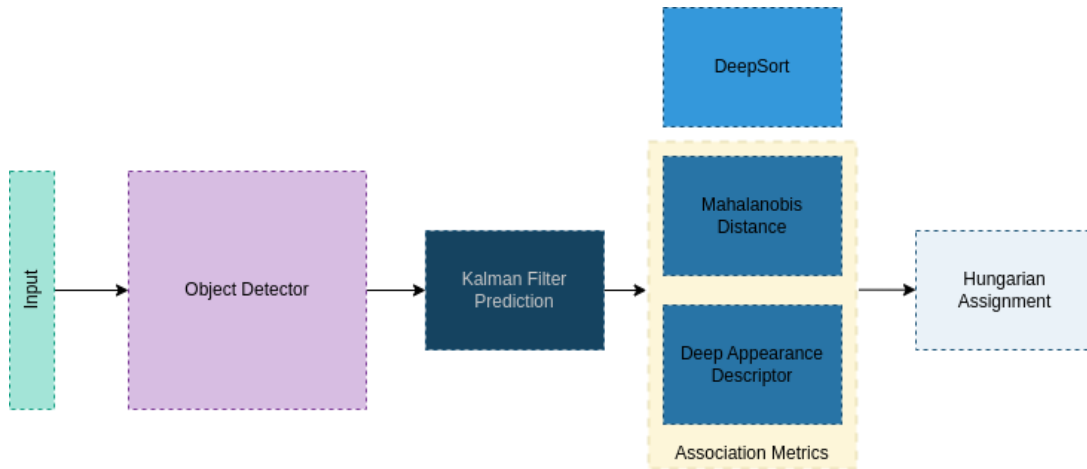


Figure C.1: DeepSORT uses a deep learning-based object detection model to identify and locate objects in each frame of a video sequence. The tracking algorithm then uses the information provided by the object detection model, along with appearance and motion information, to associate the same object across different frames and maintain its identity. This allows DeepSORT to continue tracking objects even when they are temporarily occluded or leave the field of view.

### C.3 HOTA Metrics

It is possible to divide the Multi-Object Tracking task into two main subtasks: detection and association. The previous metrics do not consider both of these tasks equally. HOTA (Higher Order Tracking Accuracy) metrics [116] are introduced as a development for this part in order to evaluate the test metrics for Multi-Object Tracking tasks. The proposed metrics can be explained as a single metric for comparing the performance of trackers while balancing the effect of accurate detection, association and localization. HOTA, build upon MOTA metrics [98] which are previously used as a performance indicator for the MOT tasks, is useful for measuring the performance of the trackers, not only with a single unified metric but also by decomposed family of sub-metrics for more detail on these metrics refer to the paper. These family of sub-metrics are quite useful while evaluating various kinds of errors of the trackers and models by tuning different requirements. This ranking system is designed for providing a single tracker evaluation metric that combines different aspects of multi-object tracking evaluation. According to the authors, HOTA can be considered as the geometric mean score of the association and detection score which according to the formula are equally balanced.

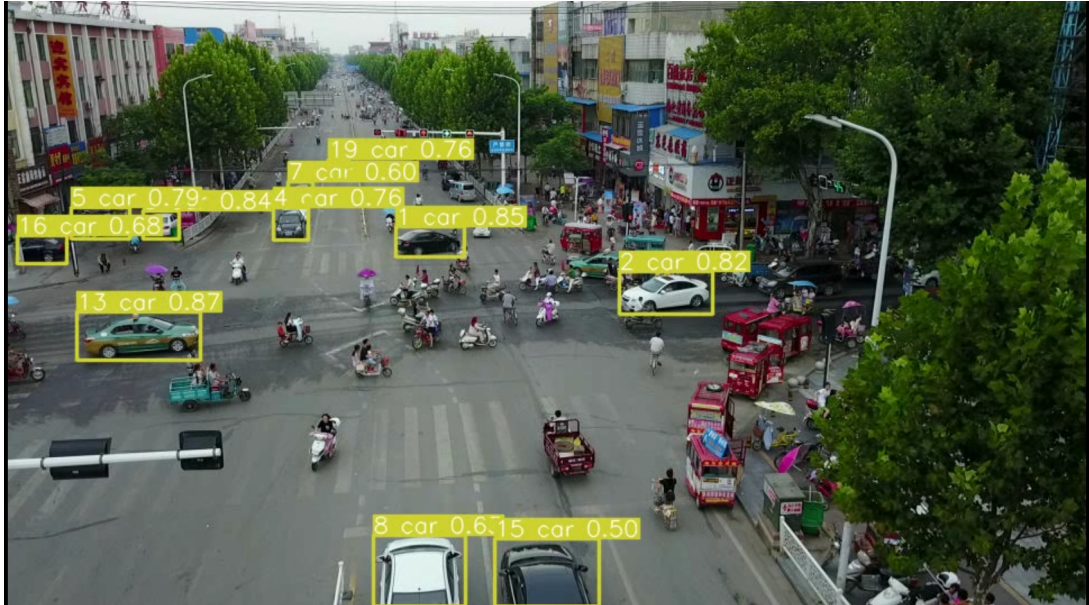
As well as MOTA, HOTA considers the matching at a detection level. In order to calculate True Positive (TP), the predicted detection (prDet) results and the ground truth detection (gtDet) are use where the similarity (S) of the location of them is greater than or equal to a threshold level  $\alpha$ . False Negative (FN) are the cases where none of gtDet matches with any prDet. Whereas, False Positive (FP) are the cases where none of prDet matches with any gtDet. These metrics are used while calculating matching of the predictions and the ground truth. As also mentioned above HOTA score considers not only detection but also the association performance of the model. In order to measure the association, came up with a novel concept named as True Positive Association (TPA), False Negative Association (FNA) and False Positive Association (FPA) which are the values defined for each hit of TP.

## C.4 Tracking Results

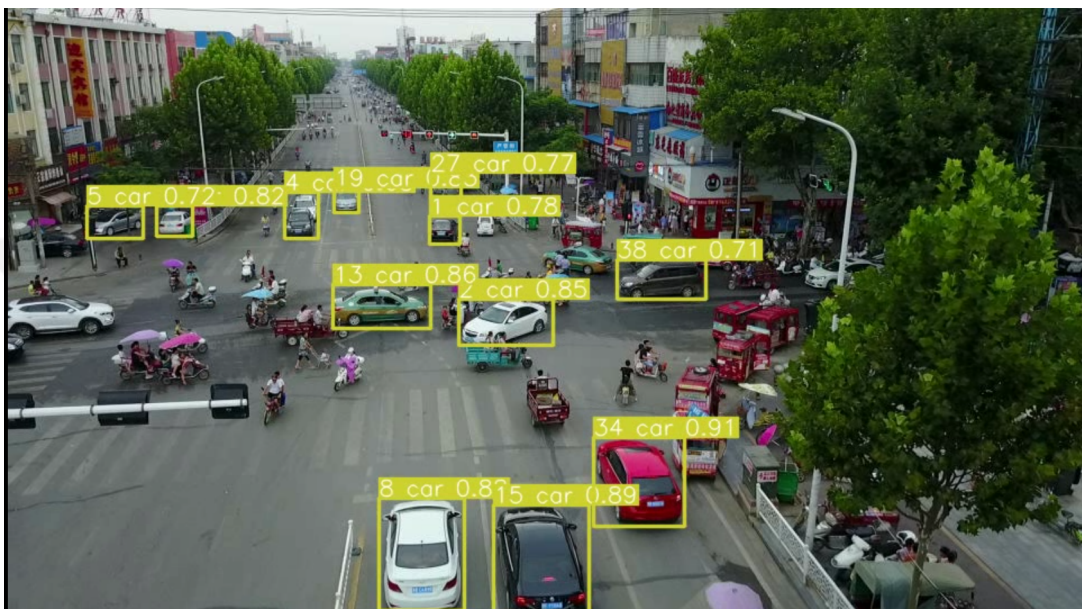
To be able to see the performance of object detection in object tracking, the following experiments were held. Different object detectors that were discussed throughout this thesis are used on VisDrone Multi-Object Tracking Dataset and as it can be seen from the Table C.1 the HOTA metric reaches the highest score with YOLODrone+ as the object detector. Also, the visual results of the tracker are provided below.

| Tracker  | YOLO Model     | HOTA          | DetA          | AssA          | DetRe         | DetPr         | AssRe         | AssPr         | LocA          | RHOTA         |
|----------|----------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| DeepSort | yolov3         | 27,59%        | 20,10%        | 38,77%        | 21,85%        | 60,64%        | 52,98%        | 50,82%        | 76,90%        | <b>28,87%</b> |
|          | yolodrone      | 27,44%        | 19,02%        | 38,87%        | <b>21,95%</b> | 61,07%        | 53,00%        | 51,92%        | 76,39%        | 28,11%        |
|          | yolov5s        | 27,09%        | 18,96%        | 39,38%        | 20,33%        | 61,17%        | 53,90%        | 51,69%        | 77,68%        | 28,13%        |
|          | yolov5s-tr     | 27,47%        | 18,73%        | <b>40,88%</b> | 20,03%        | <b>63,46%</b> | <b>54,92%</b> | 50,90%        | 77,18%        | 28,47%        |
|          | yolov5s-5-head | 26,53%        | 18,42%        | 38,90%        | 19,83%        | 61,37%        | 53,92%        | 50,88%        | 77,15%        | 27,60%        |
|          | yolodrone+     | <b>28,08%</b> | <b>21,03%</b> | 38,42%        | 19,40%        | 63,06%        | 53,27%        | <b>54,05%</b> | <b>77,63%</b> | 27,13%        |

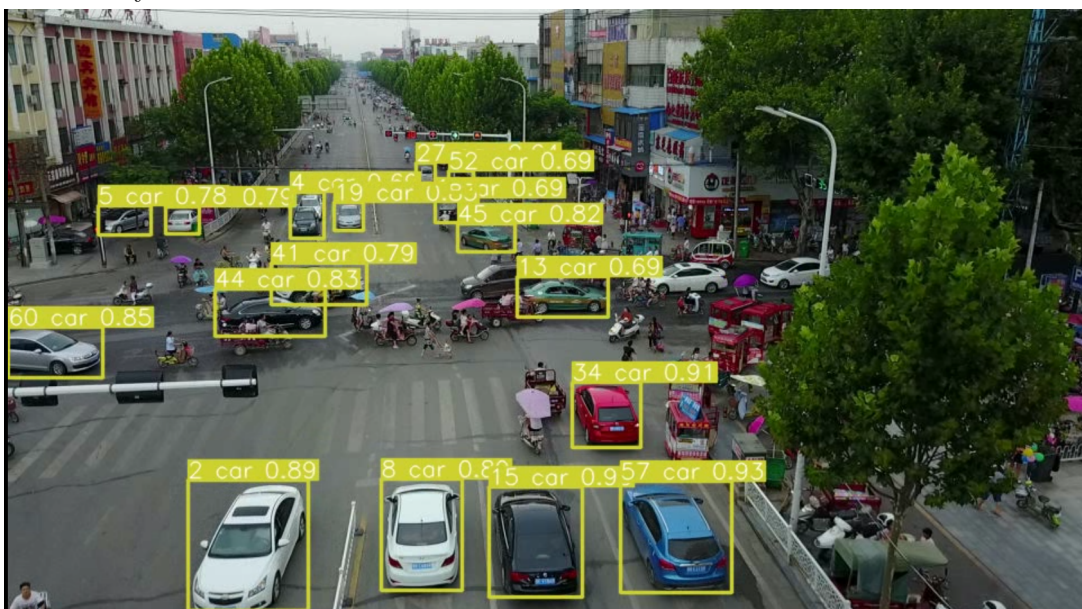
Table C.1: Shows the tracking results of different detectors of DeepSort on HOTA metrics.



(a) First frame when a car with id 2 was detected.

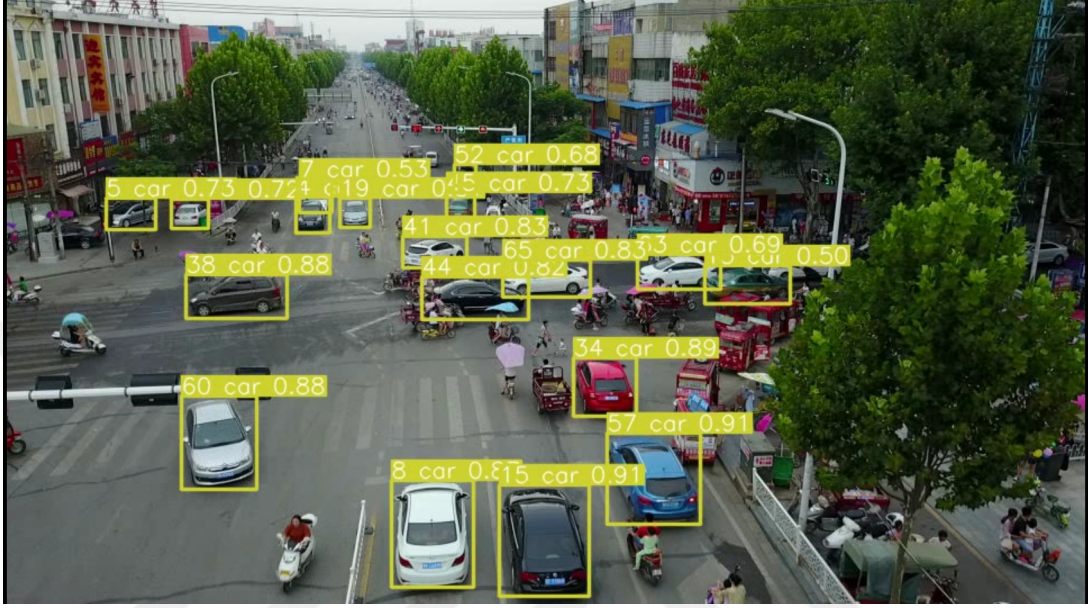


(b) First frame when a car with id 38 was detected and another car with id 2 still was successfully tracked.

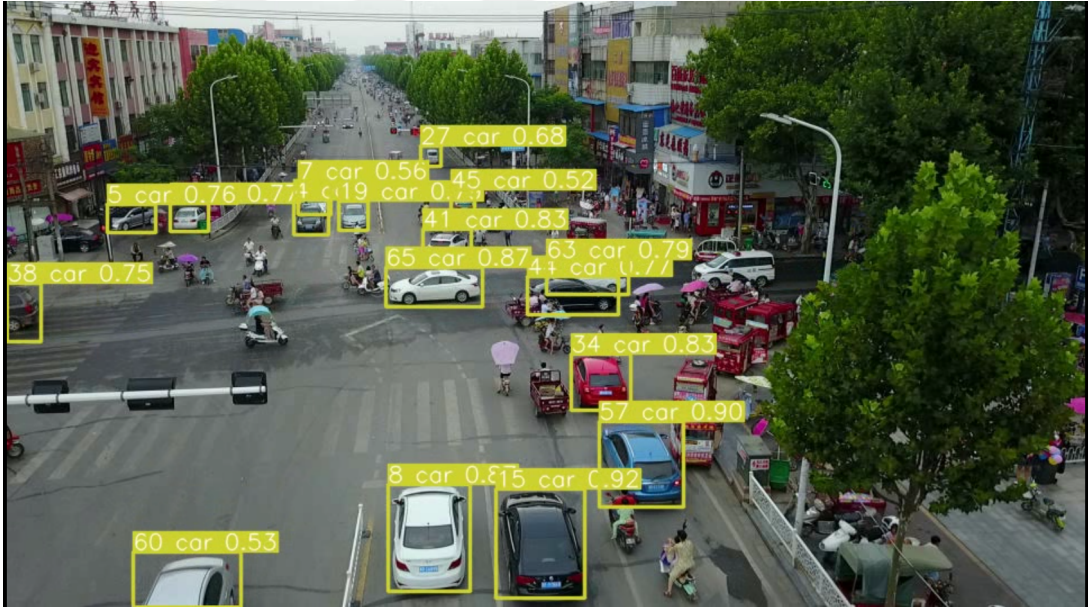


(c) Lost track of car with id 38 because of other cars covering it and car with id 2 still was successfully tracked.





(d) Successful track of car with id 38.



(e) Successful track of car with id 38 before it leaves the frame.

Figure C.2: Shows the sample output of tracking with DeepSort as a tracking algorithm and YOLODrone+ as the object detector. Figure C.2c shows that DeepSort lost the track id for black car because other cars covered it. Then as Figure C.2d suggests the tracker finds the car again and gives it, its previous id which was 38 once again.



(a) First frame when a person with id 20 was detected while the drone is in motion as it can be seen from the following images that the recording angle differs.



(b) Successful track of person with id 20 after 30 frames.





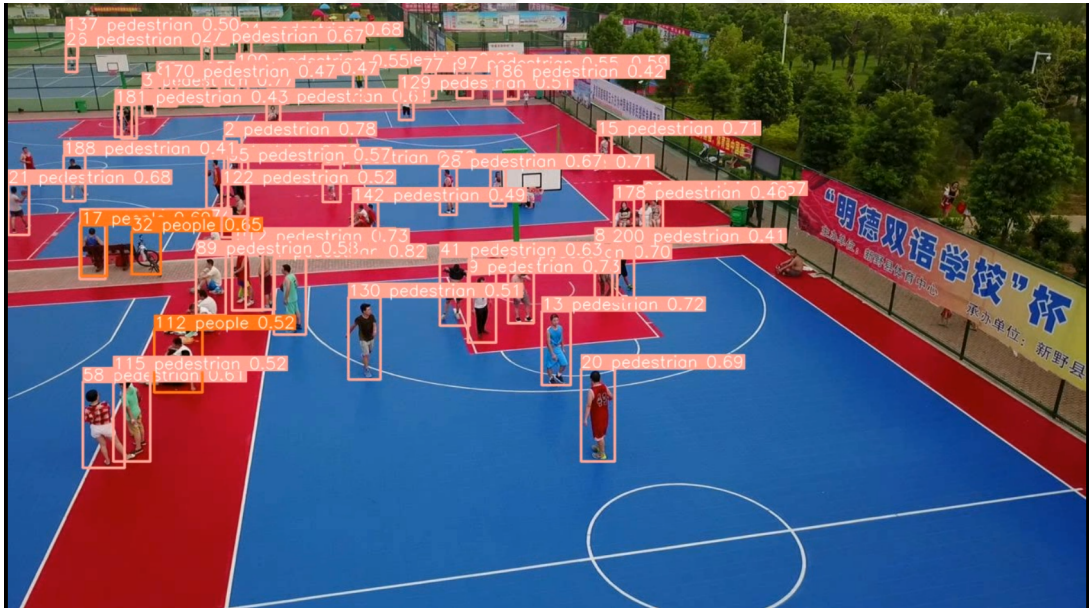
(c) Successful track of person with id 20 after 40 frames.



(d) Successful track of person with id 20 after 40 frames.



(e) Successful track of person with id 20 after 20 frames.



(f) Successful track of person with id 20 after 20 frames.

Figure C.3: Shows the sample output of tracking with DeepSort and YOLO-Drone+. As it can be seen from the given figures the tracker successfully tracks a person with id 20 even though both the drone and the target person are on the move and the scene is crowded with small objects.



# Bibliography

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *European conference on computer vision*, pp. 740–755, Springer, 2014.
- [3] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *2012 IEEE conference on computer vision and pattern recognition*, pp. 3354–3361, IEEE, 2012.
- [4] L. Wen, P. Zhu, D. Du, X. Bian, H. Ling, Q. Hu, J. Zheng, T. Peng, X. Wang, Y. Zhang, *et al.*, “Visdrone-mot2019: The vision meets drone multiple object tracking challenge results,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.
- [5] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.
- [6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

- [7] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, “Mot20: A benchmark for multi object tracking in crowded scenes,” *arXiv preprint arXiv:2003.09003*, 2020.
- [8] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, and S. Lyu, “Ua-detrac: A new benchmark and protocol for multi-object detection and tracking,” *Computer Vision and Image Understanding*, vol. 193, p. 102907, 2020.
- [9] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, “Learning social etiquette: Human trajectory understanding in crowded scenes,” in *European conference on computer vision*, pp. 549–565, Springer, 2016.
- [10] M. Mueller, N. Smith, and B. Ghanem, “A benchmark and simulator for uav tracking,” in *European conference on computer vision*, pp. 445–461, Springer, 2016.
- [11] D. Du, Y. Qi, H. Yu, Y. Yang, K. Duan, G. Li, W. Zhang, Q. Huang, and Q. Tian, “The unmanned aerial vehicle benchmark: Object detection and tracking,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 370–386, 2018.
- [12] L. A. Varga, B. Kiefer, M. Messmer, and A. Zell, “Seadronessee: A maritime benchmark for detecting humans in open water,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2260–2270, January 2022.
- [13] O. Sahin, F. E. Doğanay, S. Ozer, and C. H. Chen, “Segmentation of covid-19 infected lung area in ct scans with deep algorithms,” in *Computational Intelligence and Image Processing in Medical Applications*, pp. 33–48, World Scientific, 2022.
- [14] H. E. Ilhan, S. Ozer, G. K. Kurt, and H. A. Cirpan, “Offloading deep learning empowered image segmentation from uav to edge server,” in *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pp. 296–300, IEEE, 2021.

- [15] B. M. Albaba and S. Ozer, “Synet: An ensemble network for object detection in uav images,” 2020.
- [16] C. Zheng, X. Yan, H. Zhang, B. Wang, S. Cheng, S. Cui, and Z. Li, “Beyond 3d siamese tracking: A motion-centric paradigm for 3d single object tracking in point clouds,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8111–8120, 2022.
- [17] D. Gozen and S. Ozer, “Visual object tracking in drone images with deep reinforcement learning,” in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 10082–10089, 2021.
- [18] M. A. Özkanoglu and S. Ozer, “Infragan: A gan architecture to transfer visible images to infrared domain,” *Pattern Recognition Letters*, vol. 155, pp. 69–76, 2022.
- [19] S. Özer, M. Ege, and M. A. Özkanoglu, “Siamesefuse: A computationally efficient and a not-so-deep network to fuse visible and infrared images,” *Pattern Recognition*, vol. 129, p. 108712, 2022.
- [20] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018.
- [21] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.
- [22] G. J. et. al., “ultralytics/yolov5: v6.0 - YOLOv5n ‘Nano’ models, Roboflow integration, TensorFlow export, OpenCV DNN support,” Oct. 2021.
- [23] U. N. E. P. (UNEP), “Yes, climate change is driving wildfires,” 2022.
- [24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.
- [25] R. Huang, J. Pedoeem, and C. Chen, “Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers,” in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2503–2510, IEEE, 2018.

- [26] D. Du, P. Zhu, L. Wen, X. Bian, H. Lin, Q. Hu, T. Peng, J. Zheng, X. Wang, Y. Zhang, L. Bo, H. Shi, R. Zhu, A. Kumar, A. Li, A. Zinollayev, A. Asker-galiyev, A. Schumann, B. Mao, B. Lee, C. Liu, C. Chen, C. Pan, C. Huo, D. Yu, D. Cong, D. Zeng, D. R. Pailla, D. Li, D. Wang, D. Cho, D. Zhang, F. Bai, G. Jose, G. Gao, G. Liu, H. Xiong, H. Qi, H. Wang, H. Qiu, H. Li, H. Lu, I. Kim, J. Kim, J. Shen, J. Lee, J. Ge, J. Xu, J. Zhou, J. Meier, J. W. Choi, J. Hu, J. Zhang, J. Huang, K. Huang, K. Wang, L. Sommer, L. Jin, L. Zhang, L. Huang, L. Sun, L. Steinmann, M. Jia, N. Xu, P. Zhang, Q. Chen, Q. Lv, Q. Liu, Q. Cheng, S. S. Chennamsetty, S. Chen, S. Wei, S. S. S. Kruthiventi, S. Hong, S. Kang, T. Wu, T. Feng, V. A. Kollerathu, W. Li, W. Dai, W. Qin, W. Wang, X. Wang, X. Chen, X. Chen, X. Sun, X. Zhang, X. Zhao, X. Zhang, X. Zhang, X. Chen, X. Wei, X. Zhang, Y. Li, Y. Chen, Y. H. Toh, Y. Zhang, Y. Zhu, Y. Zhong, Z. Wang, Z. Wang, Z. Song, and Z. Liu, “Visdrone-det2019: The vision meets drone object detection in image challenge results,” in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 213–226, 2019.
- [27] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1, pp. I–I, Ieee, 2001.
- [28] P. Viola and M. J. Jones, “Robust real-time face detection,” *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.
- [29] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*, vol. 1, pp. 886–893, Ieee, 2005.
- [30] P. Felzenszwalb, D. McAllester, and D. Ramanan, “A discriminatively trained, multiscale, deformable part model,” in *2008 IEEE conference on computer vision and pattern recognition*, pp. 1–8, Ieee, 2008.
- [31] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester, “Cascade object detection with deformable part models,” in *2010 IEEE Computer society*

- conference on computer vision and pattern recognition*, pp. 2241–2248, Ieee, 2010.
- [32] T. Malisiewicz, A. Gupta, and A. A. Efros, “Ensemble of exemplar-svms for object detection and beyond,” in *2011 International conference on computer vision*, pp. 89–96, IEEE, 2011.
  - [33] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.
  - [34] R. Girshick, P. Felzenszwalb, and D. McAllester, “Object detection with grammar models,” *Advances in neural information processing systems*, vol. 24, 2011.
  - [35] R. B. Girshick, *From rigid templates to grammars: Object detection with structured models*. The University of Chicago, 2012.
  - [36] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, “Deepflow: Large displacement optical flow with deep matching,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1385–1392, 2013.
  - [37] H. Y. Cheng and J. N. Hwang, “Multiple-target tracking for crossroad traffic utilizing modified probabilistic data association,” in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP’07*, vol. 1, pp. I–921, IEEE, 2007.
  - [38] Y.-C. Lim, M. Lee, C.-H. Lee, S. Kwon, and J.-h. Lee, “Improvement of stereo vision-based position and velocity estimation and tracking using a stripe-based disparity estimation and inverse perspective map-based extended kalman filter,” *Optics and Lasers in Engineering*, vol. 48, no. 9, pp. 859–868, 2010.
  - [39] X. Cao, J. Lan, P. Yan, and X. Li, “Vehicle detection and tracking in airborne videos by multi-motion layer analysis,” *Machine Vision and Applications*, vol. 23, no. 5, pp. 921–935, 2012.

- [40] R. Valiente, M. Zaman, Y. P. Fallah, and S. Ozer, “Connected and autonomous vehicles in the deep learning era: A case study on computer-guided steering,” in *Handbook Of Pattern Recognition And Computer Vision*, pp. 365–384, World Scientific, 2020.
- [41] F. E. Doğanay, O. Sahin, S. Ozer, and C. H. Chen, “Deep learning based 3d brain tumor segmentation with multispectral mri,” in *Computational Intelligence and Image Processing in Medical Applications*, pp. 119–133, World Scientific, 2022.
- [42] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab, and H. Radha, “Deep learning algorithm for autonomous driving using googlenet,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 89–96, IEEE, 2017.
- [43] Z. Zou, Z. Shi, Y. Guo, and J. Ye, “Object detection in 20 years: A survey,” *arXiv preprint arXiv:1905.05055*, 2019.
- [44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [45] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.
- [46] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.
- [47] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *Advances in neural information processing systems*, vol. 28, 2015.
- [48] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

- [49] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.
- [50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.
- [51] M. Hearst, S. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.
- [52] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A survey of modern deep learning based object detection models,” *Digital Signal Processing*, p. 103514, 2022.
- [53] J. Redmon and A. Farhadi, “Yolo9000: better, faster, stronger,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.
- [54] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, “Yolox: Exceeding yolo series in 2021,” *arXiv preprint arXiv:2107.08430*, 2021.
- [55] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” *arXiv preprint arXiv:2004.10934*, 2020.
- [56] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *European conference on computer vision*, pp. 21–37, Springer, 2016.
- [57] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, “Dssd: Deconvolutional single shot detector,” *arXiv preprint arXiv:1701.06659*, 2017.
- [58] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

- [59] Q. Zhao, T. Sheng, Y. Wang, Z. Tang, Y. Chen, L. Cai, and H. Ling, “M2det: A single-shot object detector based on multi-level feature pyramid network,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 9259–9266, 2019.
- [60] R. N. Rajaram, E. Ohn-Bar, and M. M. Trivedi, “Refinenet: Refining object detectors for autonomous driving,” *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 4, pp. 358–368, 2016.
- [61] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, “Centernet: Key-point triplets for object detection,” in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6569–6578, 2019.
- [62] H. Law and J. Deng, “Cornernet: Detecting objects as paired keypoints,” in *Proceedings of the European conference on computer vision (ECCV)*, pp. 734–750, 2018.
- [63] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, “Deformable convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, pp. 764–773, 2017.
- [64] D. Thuan, “Evolution of yolo algorithm and yolov5: The state-of-the-art object detection algorithm,” 2021.
- [65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.
- [66] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pp. 448–456, PMLR, 2015.
- [67] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, “Cspnet: A new backbone that can enhance learning capability of cnn,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 390–391, 2020.



- [68] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path aggregation network for instance segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.
- [69] A. Newell, K. Yang, and J. Deng, “Stacked hourglass networks for human pose estimation,” in *European conference on computer vision*, pp. 483–499, Springer, 2016.
- [70] Anaconda, “The state of data science 2020 moving from hype toward maturity,” 2020.
- [71] R. Khan, “Importance of datasets in machine learning and ai research,” 2022.
- [72] P. D. P. Commission, “Managing personal data.”
- [73] P. D. P. Authority, “Personal data protection law in türkiye,” 2016.
- [74] P. Zhu, L. Wen, D. Du, X. Bian, H. Ling, Q. Hu, H. Wu, Q. Nie, H. Cheng, C. Liu, *et al.*, “Visdrone-vdt2018: The vision meets drone video detection and tracking challenge results,” in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 0–0, 2018.
- [75] O. Sahin and S. Ozer, “Yolodrone: Improved yolo architecture for object detection in drone images,” in *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pp. 361–365, 2021.
- [76] O. Sahin and S. Ozer, “Yolodrone+: Improved yolo architecture for object detection in uav images,” in *2022 30th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, 2022.
- [77] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in *European conference on computer vision*, pp. 630–645, Springer, 2016.
- [78] S. Lloyd, “Least squares quantization in pcm,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

- [79] T. M. Mitchell and T. M. Mitchell, *Machine learning*, vol. 1. McGraw-hill New York, 1997.
- [80] E. Stevens, L. Antiga, and T. Viehmann, *Deep learning with PyTorch*. Manning Publications, 2020.
- [81] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical evaluation of rectified activations in convolutional network,” *arXiv preprint arXiv:1505.00853*, 2015.
- [82] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, “Mots: Multi-object tracking and segmentation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7942–7951, 2019.
- [83] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.
- [84] Z. Huang, J. Wang, X. Fu, T. Yu, Y. Guo, and R. Wang, “Dc-spp-yolo: Dense connection and spatial pyramid pooling based yolo for object detection,” *Information Sciences*, vol. 522, pp. 241–258, 2020.
- [85] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [86] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [87] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, *et al.*, “Exploring the limits of transfer learning with a unified text-to-text transformer,” *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.
- [88] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, “Xlnet: Generalized autoregressive pretraining for language understanding,” *Advances in neural information processing systems*, vol. 32, 2019.

- [89] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*.
- [90] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus),” *arXiv preprint arXiv:1606.08415*, 2016.
- [91] R. Padilla, S. L. Netto, and E. A. B. da Silva, “A survey on performance metrics for object-detection algorithms,” in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242, 2020.
- [92] J. A. Hanley and B. J. McNeil, “The meaning and use of the area under a receiver operating characteristic (roc) curve.,” *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.
- [93] P. Jaccard, “Étude comparative de la distribution florale dans une portion des alpes et des jura,” *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.
- [94] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimeshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.
- [95] G. J. et. al., “ultralytics/ultralytics,” Jan. 2023.
- [96] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” *arXiv preprint arXiv:2207.02696*, 2022.
- [97] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

- [98] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, “Motchallenge 2015: Towards a benchmark for multi-target tracking,” *arXiv preprint arXiv:1504.01942*, 2015.
- [99] B. Sekachev, N. Manovich, M. Zhiltsov, A. Zhavoronkov, D. Kalinin, B. Hoff, T. Osmanov, D. Kruchinin, A. Zankevich, Dmitriy Sidnev, M. Markelov, Johannes222, M. Chenuet, a andre, telenachos, A. Melnikov, J. Kim, L. Ilouz, N. Glazov, Priya4607, R. Tehrani, S. Jeong, V. Skubriev, S. Yonekura, vugia truong, zliang7, lizhming, and T. Truong, “opencv/cvat: v1.1.0,” Aug. 2020.
- [100] M. T., “New computer vision tool accelerates annotation of digital images and video,” 2019.
- [101] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, “Bytetrack: Multi-object tracking by associating every detection box,” 2022.
- [102] N. Wojke, A. Bewley, and D. Paulus, “Simple online and realtime tracking with a deep association metric,” in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3645–3649, 2017.
- [103] Y. Du, Y. Song, B. Yang, and Y. Zhao, “Strongsort: Make deepsort great again,” *arXiv preprint arXiv:2202.13514*, 2022.
- [104] A. Pramanik, S. K. Pal, J. Maiti, and P. Mitra, “Granulated rcnn and multi-class deep sort for multi-object detection and tracking,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 1, pp. 171–181, 2022.
- [105] J. Son, M. Baek, M. Cho, and B. Han, “Multi-object tracking with quadruplet convolutional neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5620–5629, 2017.
- [106] C. Ma, C. Yang, F. Yang, Y. Zhuang, Z. Zhang, H. Jia, and X. Xie, “Trajectory factory: Tracklet cleaving and re-connection by deep siamese bi-gru for multiple object tracking,” in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, IEEE, 2018.

- [107] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.
- [108] L. Chen, H. Ai, C. Shang, Z. Zhuang, and B. Bai, “Online multi-object tracking with convolutional neural networks,” in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 645–649, 2017.
- [109] Q. Chu, W. Ouyang, H. Li, X. Wang, B. Liu, and N. Yu, “Online multi-object tracking using cnn-based single object tracker with spatial-temporal attention mechanism,” in *Proceedings of the IEEE international conference on computer vision*, pp. 4836–4845, 2017.
- [110] F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan, “Poi: Multiple object tracking with high performance detection and appearance feature,” in *European Conference on Computer Vision*, pp. 36–42, Springer, 2016.
- [111] R. E. Kalman, “A new approach to linear filtering and prediction problems,” 1960.
- [112] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
- [113] L. Chen, H. Ai, Z. Zhuang, and C. Shang, “Real-time multiple people tracking with deeply learned candidate selection and person re-identification,” in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, 2018.
- [114] W. Feng, Z. Hu, W. Wu, J. Yan, and W. Ouyang, “Multi-object tracking with multiple cues and switcher-aware classification,” *arXiv preprint arXiv:1901.06129*, 2019.
- [115] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” in *2016 IEEE international conference on image processing (ICIP)*, pp. 3464–3468, IEEE, 2016.

- [116] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé, and B. Leibe, “Hota: A higher order metric for evaluating multi-object tracking,” *International journal of computer vision*, vol. 129, no. 2, pp. 548–578, 2021.

