



Università Politecnica Delle Marche

Dipartimento di Ingegneria dell'Informazione

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E
DELL'AUTOMAZIONE

TITOLO:

**Progettazione e sviluppo di algoritmi basati su deep learning per il
rilevamento ed inseguimento di oggetti da immagini ottenute da
drone**

**Design and development of deep learning based algorithms for object
detection and tracking from drone images**

Relatore:

Prof. Adriano Mancini

Correlatore:

Dott. Luigi Ridolfi

Candidato: 1102498

Davide Olivieri

ANNO ACCADEMICO 2021 / 2022

Vorrei esprimere la mia gratitudine a tutte le persone che hanno contribuito alla realizzazione della mia tesi, in particolare al Prof. Adriano Mancini, al mio tutor aziendale il Dott. Luigi Ridolfi e all'intera azienda MBDA Italia s.p.a che mi ha dato la possibilità e i mezzi necessari per svolgere questo lavoro. Un ringraziamento speciale va ai miei genitori che con il loro costante sostegno mi hanno permesso di raggiungere questo importante traguardo, inoltre vorrei ringraziare i miei compagni di studio, Lorenzo e Antonio, per il loro supporto.

Abstract

The related work was carried out during a curricular internship at the company **MBDA Italia s.p.a**, Europe's leading designer and manufacturer of missiles and missile systems. The aim of this work is to study solutions for the detection and tracking of objects acquired by drones using Deep Learning approaches. Particular attention was paid to studying the relationship between the accuracy and inference speed of the models, in order to run them on embedded boards and test their benefits in Real Time. The dataset chosen for this work is **VisDrone 2019**, it represents a challenge, still completely open due to its enormous difficulties derived from the high unbalanced of classes and the size of the objects.

In the first part of the project we will look at the results obtained for Multi Object Detection through **YOLOv5** (You Only Look Once) and compare them with the state-of-the-art results for the VisDrone 2019 Challenge.

The second part of the project focuses on solving the problem of Multi Object Tracking of images taken by drones. For this purpose, a cascade approach was adopted between two algorithms, the first of which uses YOLOv5 for object detection, while the second (**strongSORT**) takes the output of YOLOv5 as input and creates traces for each detected object by assigning it an ID. This approach was tested first on a single person tracking task taken from a drone, and then on video sequences from the VisDrone Tracking dataset, achieving good results.

The last part of the project is focused on the deployment of the Object Detection algorithms on the **NVIDIA Jetson TX2** board, in order to compare the execution times and study the behaviour of the networks in Real Time.

Contents

1	Introduction	8
1.1	Problem Definition	9
1.2	Project Steps	11
1.3	Glossary	12
2	Materials and Methods	13
2.1	Convolutional Neural Network	13
2.2	Multi Object Detection	16
2.3	Multi Object Tracking	18
2.4	Materials	22
2.4.1	Docker	22
2.4.2	GitHub	25
2.4.3	PyTorch	26
3	Object Detection using YOLOv5	28
3.1	YOLOv5	29
3.1.1	Architecture	31
3.1.2	Non Maximum Suppresion Algorithm (NMS)	34
3.1.3	Loss Function	35
3.1.4	Anchors Box	36
3.2	Evaluation Metrics	37
3.3	VisDrone Dataset	40
3.3.1	DPNet	43
3.4	Implementation	43

CONTENTS

3.5	Training	44
3.6	Testing	47
3.7	Optimization	48
4	Object Tracking adding strongSORT	52
4.1	Architecture	52
4.1.1	Re-Identification	54
4.1.2	Hungarian Algorithm	55
4.2	Evaluation Metrics	57
4.3	Testing	59
4.3.1	Optimization	61
5	Deploy	63
5.1	NVIDIA Jetson TX2	64
5.2	Testing	64
5.3	Real-Time Testing	65
6	Conclusion	68
6.1	Future works	68
7	Sommario	70
	Bibliography	78

Chapter 1

Introduction

Computer Vision is one of the fields of artificial intelligence that trains and enables computers to understand the visual world. Computers can use digital images and deep learning models to accurately identify and classify objects and react to them.

Computer vision in Artificial Intelligence is dedicated to the development of automated systems that can interpret visual data in the same manner as people do. The idea behind computer vision is to instruct computers to interpret and comprehend images on a pixel-by-pixel basis. This is the foundation of the computer vision field. Regarding the technical side of things, computers will seek to extract visual data, manage it, and analyze the outcomes using sophisticated software programs.

In **history** scientists and engineers have been trying to develop ways for machines to see and understand visual data for about 60 years. In the 1960s, AI emerged as an academic field of study, and it also marked the beginning of the AI quest to solve the human vision problem. In 1982, neuroscientist David Marr established that vision works hierarchically and introduced algorithms for machines to detect edges, corners, curves and similar basic shapes. Concurrently, computer scientist Kunihiko Fukushima developed a network of cells that could recognize patterns. The network, called the Neocognitron, included convolutional layers in a neural network.

By 2000, the focus of study was on object recognition, and by 2001, the first real-time face recognition applications appeared. Standardization of how visual data sets are tagged and annotated emerged through the 2000s. In 2010, the ImageNet dataset [1] became available. It contained millions of tagged images across a thousand object classes and provides a foundation for Convolutional Neural Networks and deep learning models used today.

1.1 Problem Definition

One of the main problems in computer vision is object detection, it is the basis of many computer vision applications, such as instance segmentation, image captioning, object tracking, etc. This study will address the challenge of object detection and tracking using deep learning approaches. Before studying this type of problem, it is necessary to explain other fundamental aspects of computer vision.

Image classification [2] focuses on classifying objects within an image into one or more predefined categories. This problem is different from object detection as it does not focus on the precise location of objects. Currently, the challenge of this problem can be addressed with contemporary techniques such as convolutional neural networks, but there have been various classical approaches developed throughout the years. This report implements an image classifier using a combination of classic computer vision and deep learning techniques.

Semantic segmentation [3] is another problem that focuses on separating objects within an image into different categories. The main difference with image classification is that semantic segmentation provides information on the precise location of each object within the image. The objective of semantic segmentation is to assign a label to each pixel of an image. The task involves dividing an image into several segments, each of which corresponds to a different object or background, and assigning a label to each of these segments. This provides a much more detailed understanding of the scene than traditional image classification, in which only the presence of an object in an image is determined.

Instance segmentation [4] is a computer vision task in which objects in an image are segmented and differentiated from one another. The goal of instance segmentation is to differentiate objects belonging to the same category, for example, it can distinguish between two "car" objects within an image. This is an extension of semantic segmentation, which separates objects in an image into different categories, but does not differentiate between objects of the same category.

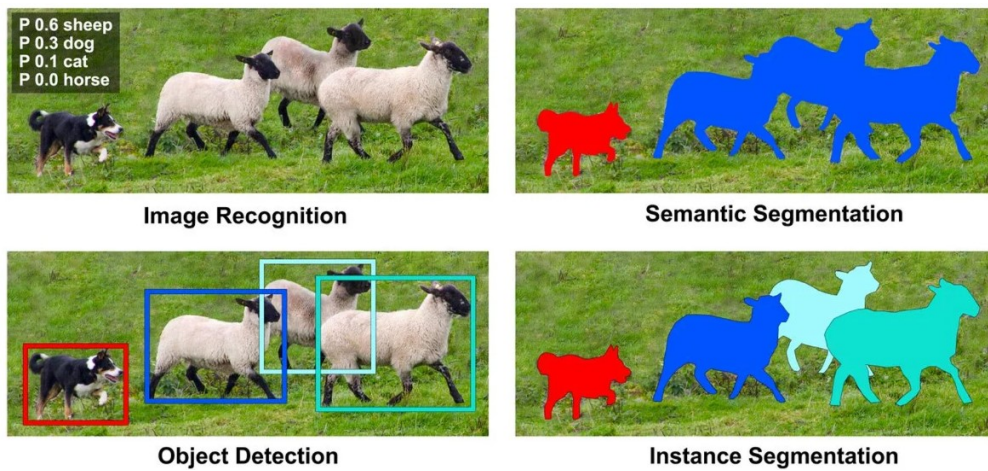


Figure 1.1: Computer Vision Challenges [5]

While object detection combines all challenges by focusing on the identification and classification of specific objects within an image or video, while also providing their coordinates.

1.2 Project Steps

The aim of the project is to test new Deep Learning approaches with CNN for applications on unmanned aerial systems, with a focus on execution speed for implementation on embedded platforms. This study will deal the following steps:

1. Fine-Tuning of five **YOLOv5** models for Multi Detection Object with VisDrone Dataset [sec. 3.3].
2. Testing YOLOv5 models on VisDrone Test Dataset [sec. 3.6].
3. Models optimization with quantization technical after training.
4. Study of the trade-off between inference speed and accuracy [3.7].
5. Implementation of an object tracker model obtained by adding the **strongSORT** algorithm to the detector output [4.1].
6. Testing the tracking algorithm with single object and multi object datasets to study the trade-off between speed and accuracy [4.3].
7. Deployment of YOLOv5 models for object detection on embedded platform (NVIDIA Jetson TX2) [sec. 5.1].
8. Testing and optimization of models on the NVIDIA Jetson TX2 [sec.5.2].
9. Real Time Object Detection with YOLOv5 models to evaluate inference speed [sec. 5.3].

1.3 Glossary

CNN : Convolutional Neural Network

AI : Artificial Intelligence

FC : Fully Connected

CV : Computer Vision

OS : Operating System

MOD : Multi Object Detection

MOT : Multi Object Tracking

R-CNN : Region Based Convolutional Neural Networks

YOLO : You Only Look Once

SSD : Single Shot Detector

CSRT : Comparing state of the art Region of Interest trackers

KCF : Tracking with Correlation Filters

SVM : Support Vector Machine

UAS : Unmanned Aerial Systems

CUDA : Compute Unified Device Architecture

VM : Virtual Machine

mAP : Mean Average Precision

AP : Average Precision

mAP : Mean Average Precision

CPU : Central Processing Unit

GPU : Graphics processing unit

CLI : Command-Line Interface

CSP : Cross Stage Spatial

SSP : Spatial Pyramid Pooling

NMS : Non Maximum Suppression

IoU : Intersection Over Union

FP32 : Floating Point 32 bit

FP16 : Floating Point 16 bit

INT8 : Integer 8 bit

Chapter 2

Materials and Methods

MBDA is the only European group capable of designing and producing missiles and missile systems to meet the whole range of current and future needs of the three armed forces, therefore the scenario considered is **airspace**, in particular terrestrial recognition from airspace.

Currently there is a lot of AI research in the defence sector, this work focuses on using Convolutional Neural Network (CNN) for the detection, recognition and tracking of objects seen by drones.

These Convolutional Neural Networks could be used on board missiles to engage target and track it. In this chapter, the challenges to be solved and the solutions proposed for this study will be explained.

2.1 Convolutional Neural Network

A **Convolutional Neural Network** (CNN) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.

There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice. This makes them highly suitable for computer vision tasks and for applications where object

recognition is vital, such as self-driving cars and facial recognition.

A CNN's **architecture** is analogous to the connectivity pattern of the human brain. Just like the brain consists of billions of neurons, CNNs also have neurons arranged in a specific way. In fact, a CNN's neurons are arranged like the brain's frontal lobe, the area responsible for processing visual stimuli. This arrangement ensures that the entire visual field is covered, thus avoiding the piecemeal image processing problem of traditional neural networks, which must be fed images in reduced-resolution pieces. Compared to the older networks, a CNN delivers better performance with image inputs, and also with speech or audio signal inputs.

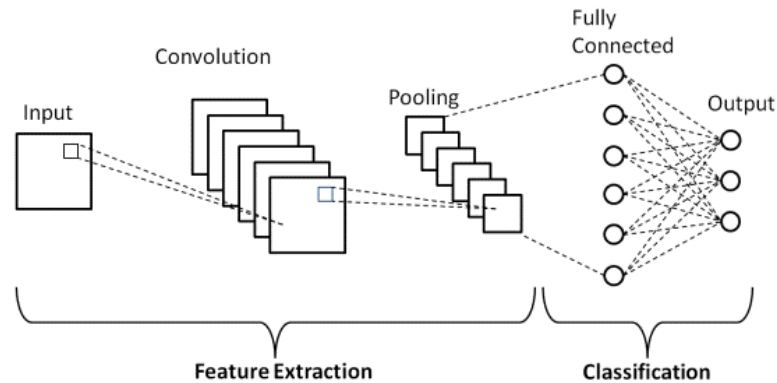


Figure 2.1: Basic CNN Architecture [6]

As shown in the image 2.1, a deep learning CNN consists of three layers:

- **Convolutional layer**, the majority of computations happen in the convolutional layer, which is the core building block of a CNN. A second convolutional layer can follow the initial convolutional layer. The process of convolution involves a kernel or filter inside this layer moving across the receptive fields of the image, checking if a feature is present in the image. Over multiple iterations, the kernel sweeps over the entire image. After each iteration a dot product is calculated between the input pixels and the filter. The final output from the series of dots is known as a feature map or convolved feature. Ultimately, the image is converted into numerical values in this layer, which allows the CNN to interpret the image and extract

relevant patterns from it

- **Pooling layer**, In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations. It basically summarises the features generated by a convolution layer.

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

- **Fully connected layer**, the FC layer is where image classification happens in the CNN based on the features extracted in the previous layers. The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place. The reason two layers are connected is that two fully connected layers will perform better than a single connected layer. These layers in CNN reduce the human supervision.

2.2 Multi Object Detection

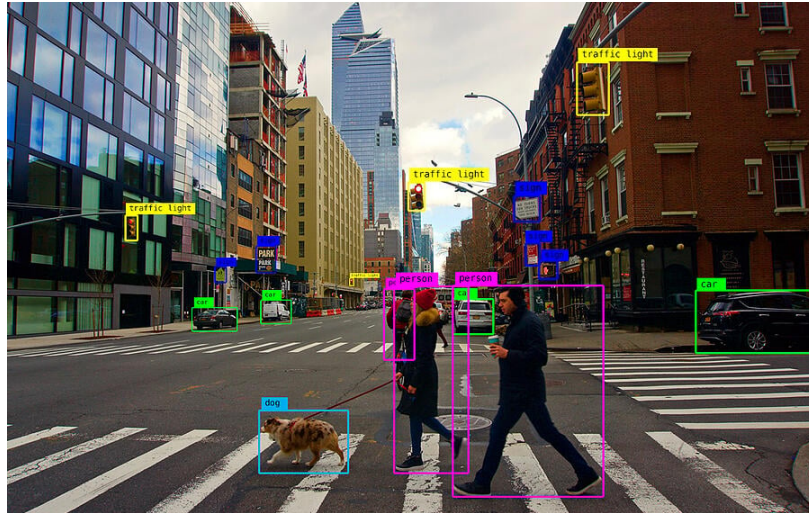


Figure 2.2: Object Detection [7]

One of main challenges on computer vision is the Object Detection. As explained by *Zou et al.* [8], the **Object Detection** is an important computer vision task that concerns the identification of class instance objects in an image (or video), and locating the actual position of said object in the picture. Commonly the spacial orientation of the detected object is framed by a rectangular bounding box that determines its height and width. Thus, object detection is far more powerful than mere image classification, not only because it "draws" a box where the object is located, but also because it can identify multiple object instances in a single image, while classification models have the limit of labeling only the one predominant object in the scene. As one of the fundamental problems of computer vision, object detection forms the basis of many other computer vision tasks, such as instance segmentation, image captioning, object tracking, etc.

From the application point of view, object detection can be grouped into two research topics "*general object detection*" and "*detection applications*", where the former one aims to explore the methods of detecting different types of objects under a unified framework to simulate the human vision and cognition, and the later one refers to the detection under specific application scenarios, such as

pedestrian detection, face detection, text detection, etc.

In recent years, as shown in figure 2.3, the rapid development of deep learning techniques has brought new blood into object detection, leading to remarkable breakthroughs and pushing it forward to a research hot-spot with unprecedented attention. Object detection has now been widely used in many real-world applications, such as autonomous driving, robot vision, video surveillance.

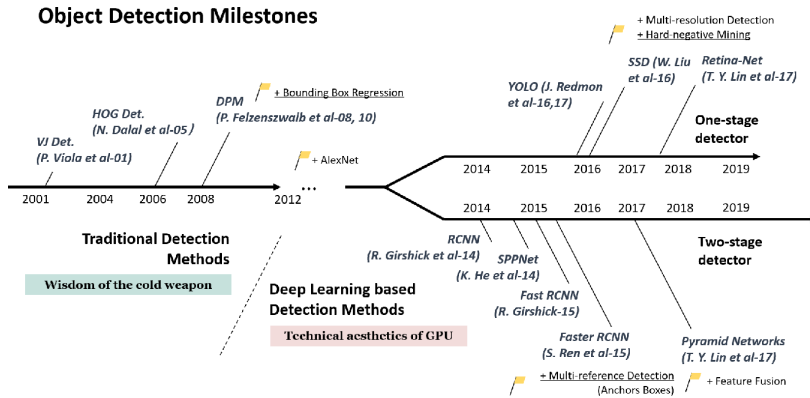


Figure 2.3: A road map of object detection [8]

Currently, the models for object detection can be divided in two categories:

Two-Stages and **One-Stage**.

In the first category (Two-Stages), we talk about models that split the object detection task in several steps, as:

- **R-CNN** (Region Based Convolutional Neural Networks) [9] first generates region proposals using an algorithm such as Edge Boxes. The proposal regions are cropped out of the image and resized. Then, the CNN classifies the cropped and resized regions. Finally, the region proposal bounding boxes are refined by a support vector machine (SVM) that is trained using CNN features.
- **The Faster R-CNN** [10] adds a region proposal network (RPN) to generate region proposals directly in the network instead of using an external algorithm like Edge Boxes. The RPN uses Anchor Boxes for Object Detection. Generating region proposals in the network is faster and better tuned

to your data.

To the second category (One-Stage) belong all models that compute the object detection in one step, as:

- **YOLO** (You Only Look Once), is the simplest object detection architecture. It predicts bounding boxes through a grid based approach after the object goes through the CNN. It divides each image into an $(N \times N)$ grid, with each grid predicting M boxes that contain any object. From those $(N \times N \times M)$ boxes, it classifies each box for every class and picks the highest class probability. To ensure the time of inference, for this work has been selected YOLOv5 model [?], described in detail in sec. 3.1.
- **SSD** (Single Shot Detector), is similar to YOLO, but uses the feature maps of each convolutional layer (output of each filter/layer) to predict the bounding boxes. After consolidating all the feature maps, it runs a 3×3 convolutional kernel on them to predict bounding boxes and classification probability. SSD is a family of algorithms, with the popular choice being RetinaNet.

2.3 Multi Object Tracking

Multi Object Tracking (MOT) is a crucial component of situational awareness in military defense applications. With the growing use of Unmanned Aerial Systems (UASs), MOT methods for aerial surveillance is in high demand [11].



Figure 2.4: Samples of tracking

Object Tracking in deep learning is the task of predicting the positions of objects throughout a video using their spatial as well as temporal features. More technically, tracking is getting the initial set of detections, assigning unique ids, and tracking them throughout frames of the video feed while maintaining the assigned ids.

When the Object Detection gives information about the presence of objects in a frame, Object Tracking goes beyond simple observation to more useful action of monitoring objects. Unmanned Aerial Vehicle (UAV), such as a missile might be, would need to know the location and all objects detected by the camera in order to target acquisition or to carry out reconnaissance, which is why the challenge of multi object tracking has become one of the most important in recent years with the advent of deep learning. By utilizing the information from a sequence of frames which measured with timestamp, autonomous object can come up with an estimate of the positions of object in future timestamp.

Target tracking has been studied for decades with numerous applications [12]. As explained by *Satya Mallick* on your blog [13], the several reasons why tracker is needed:

- **Tracking when object detection fails**, there are many cases where an object detector might fail. But if we have an object tracker in place, it will still be able to predict the objects in the frame. For example, Consider

a video where a motorbike running through the woods and we apply a detector to detect the motorbike. Here's what will happen in this case, Whenever a bike gets occluded or overlapped by a tree the detector will fail. But, if we have a tracker with it, we will still be able to predict and track the motorbike.

- **ID assignment:** while using a detector, it only showcases the location of the objects, if we just look at the array of outputs we will not know which coordinates belong to which box. On the other hand, A tracker assigns an ID to each object it tracks and maintains that ID till the lifetime of that object in that frame.
- **Real-time predictions:** trackers are very fast and generally faster than detectors. Because of this property, trackers can be used in real-time scenarios and has many applications in the real world.

There are many types of trackers can be classified based on number of objects to be tracked:

- **Single Object Tracker**, these types of trackers track only a single object even if there are many other objects present in the frame. They work by first initializing the location of the object in the first frame, and then tracking it throughout the sequence of frames. These types of tracking methods are very fast. Some of them are **CSRT**, **KCF**, and many more which are built using Traditional computer vision [14]. However, deep learning based trackers are now proved to be far more accurate than traditional trackers. For example, **GOTURN** [15] is a example of deep learning based single object trackers.
- **Multi Object Tracker**, these types of trackers can track multiple objects present in a frame. Multiple object trackers are trained on a large amount of data, unlike traditional trackers. Therefore, they are proved to be more accurate as they can track multiple objects and even of different classes at the same time while maintaining high speed.

In this related work, as explained in detail on chapter 4, will be use the **strongSORT** tracker.

Both type of tracker can be belong to two categories:

- **Tracking by Detection**, the type of tracking algorithm where the object detector detects the objects in the frames and then perform data association across frames to generate trajectories hence tracking the object. These types of algorithms help in tracking multiple objects and tracking new objects introduced in the frame. Most importantly, they help track objects even if the object detection fails.
- **Tracking without Detection**, the type of tracking algorithm where the coordinates of the object are manually initialized and then the object is tracked in further frames.

In this study will be used a **Tracker Multi Object with Detection 2.5**, where the detector is YOLOv5 model while the tracker is strongSORT algorithm.

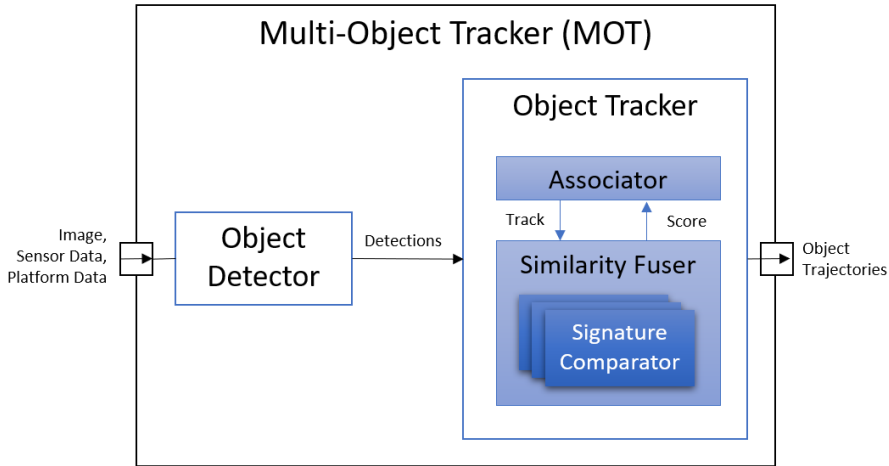


Figure 2.5: MOT Functional Architecture by Xie et al. [11]

In this study will be used a tracker multi object with detection, figure 2.5, where the Object Detector is **YOLOv5** while the Object Tracker is **strongSORT** with **OSNET**.

2.4 Materials

Throughout the development phase was used a **NVIDIA GPU P4000** [16] that combines a 1792 CUDA core Pascal GPU, large 8 GB GDDR5 memory and advanced display technologies to deliver the performance and features that are required by demanding professional applications, while **Ubuntu 20.04 LTS** [17] was used as Operating System.

To ensure a safe and clean development environment was used **Docker** [18] platform that creates a isolated container for application.

There are many frameworks for deep learning, the most widely used are PyTorch and TensorFlow, but we chose **PyTorch**, based on the Python programming language and the Torch library. [19], because Python's performance is faster for PyTorch, and also for convolutional networks PyTorch's training time is significantly higher than that of TensorFlow on GPUs [20].

2.4.1 Docker

Docker is an open source platform that enables developers to build, deploy, run, update and manage containers—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

Containers simplify development and delivery of distributed applications. They have become increasingly popular as organizations shift to cloud-native development and hybrid multicloud environments. It's possible for developers to create containers without Docker, by working directly with capabilities built into Linux and other operating systems. But Docker makes containerization faster, easier and safer.

Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as *Control Groups* (Cgroups) for allocating resources among processes, and *Namespaces* for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single

instance of the host operating system in much the same way that a hypervisor enables multiple virtual machines (VMs) to share the CPU, memory and other resources of a single hardware server. As a result, container technology offers all the functionality and benefits of VMs including application isolation, cost effective scalability, and disposability plus important additional advantages.

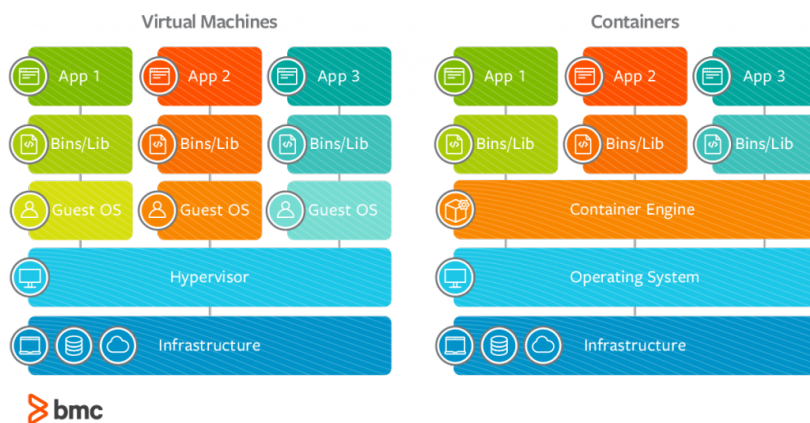


Figure 2.6: Docker Architecture [18]

In the figure 2.6 you can see the difference in architecture between VM and Docker.

Now will explain how was created the Docker Application. Every Docker container starts with a simple text file containing instructions for how to build the Docker container image. **DockerFile** automates the process of Docker Image creation. It's essentially a list of command-line interface (CLI) instructions that Docker Engine will run in order to assemble the image.

Docker Images contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container. It's possible to build a Docker image from scratch, but most developers pull them down from common repositories, for this work in the Docker Hub Store there is a **YOLOv5 Docker Image** [21] to pull down.

```
1 # YOLOv5 by Ultralytics, GPL-3.0 license
2 # Start FROM NVIDIA PyTorch image
3 FROM nvr.io/nvidia/pytorch:22.04-py3
4 RUN rm -rf /opt/pytorch
5
6 # Install linux packages
7 RUN apt update && apt install --no-install-recommends -y zip htop
   screen libgl1-mesa-glx
8
9 # Install pip packages
10 COPY requirements.txt .
11 RUN python -m pip install --upgrade pip
12 RUN pip uninstall -y torch torchvision torchtex Pillow
13 RUN pip install --no-cache -r requirements.txt albumentations wandb
   gsutil notebook Pillow>=9.1.0
14
15 # Create working directory
16 RUN mkdir -p /usr/src/app
17 WORKDIR /usr/src/app
18
19 # Copy contents
20 COPY . /usr/src/app
21 RUN git clone https://github.com/ultralytics/yolov5 /usr/src/yolov5
22
23 # Set environment variables
24 ENV OMP_NUM_THREADS=8
```


In the Dockerfile used to build the YOLOv5 Image there are 5 key words:

- **FROM**, a Dockerfile must begin with a FROM instruction, it specifies the Parent Image from which you are building. In this case was used *NVIDIA PyTorch Image*
- **RUN**, this instruction will execute any commands in a new layer on top of the current image and commit the results. It was used to install linux packages and to clone GitHub Repository [22].
- **COPY**, to copy files or folder from local to container, in this case to copy the requirements to install
- **WORKDIR**, used to create a working directory
- **ENV**, to set environment variables

Docker containers are the live, running instances of Docker images. While Docker images are read-only files, containers are live, ephemeral, executable content. Users can interact with them, and administrators can adjust their settings and conditions using Docker commands. Each time a container is created from a Docker image, yet another new layer called the container layer is created.

2.4.2 GitHub

GitHub is a for profit company that offers a cloud based Git repository hosting service. Essentially, it makes it a lot easier for individuals and teams to use Git for version control and collaboration.

GitHub's interface is user friendly enough so even novice coders can take advantage of Git. Without GitHub, using Git generally requires a bit more technical savvy and use of the command line.

GitHub is so user friendly, though, that some people even use GitHub to manage other types of projects, like writing books. Additionally, anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects. The main purpose of GitHub is to facilitate

the version control and issue tracking aspects of software development. Labels, milestones, responsibility assignment, and a search engine are available for issue tracking. For version control, Git allows pull requests to propose changes to the source code.

Inside the Docker container, the YOLOv5 repository was extracted from GitHub [22], then other GitHub repositories were used within the same container, creating a single repository. Have been integrated the strongSORT repository to MOT [23], the MOT Evaluation repository to calculate the metrics [24] and the Quatization repository [25].

2.4.3 PyTorch

PyTorch is an open source deep learning framework that's known for its flexibility and ease of use. This is enabled in part by its compatibility with the popular *Python* high level programming language favored by machine learning developers and data scientists.

PyTorch is a fully featured framework for building deep learning models, which is a type of machine learning that's commonly used in applications like image recognition and language processing. Written in Python, it's relatively easy for most machine learning developers to learn and use. PyTorch is distinctive for its excellent support for GPUs and its use of reverse mode auto differentiation, which enables computation graphs to be modified on the fly. This makes it a popular choice for fast experimentation and prototyping.

The framework combines the efficient and flexible GPU-accelerated backend libraries from Torch with an intuitive Python frontend that focuses on rapid prototyping, readable code, and support for the widest possible variety of deep learning models. PyTorch and TensorFlow are similar in that the core components of both are tensors and graphs.

Tensors are a core PyTorch data type, similar to a multidimensional array, used to store and manipulate the inputs and outputs of a model, as well as the model's parameters. Tensors are similar to NumPy's arrays, except that tensors can run on GPUs to accelerate computing.

Graphs are data structures consisting of connected nodes (called vertices) and edges. Every modern framework for deep learning is based on the concept of graphs, where Neural Networks are represented as a graph structure of computations. PyTorch is based on dynamic computation graphs, where the computation graph is built and rebuilt at runtime, with the same code that performs the computations for the forward pass also creating the data structure needed for back propagation.

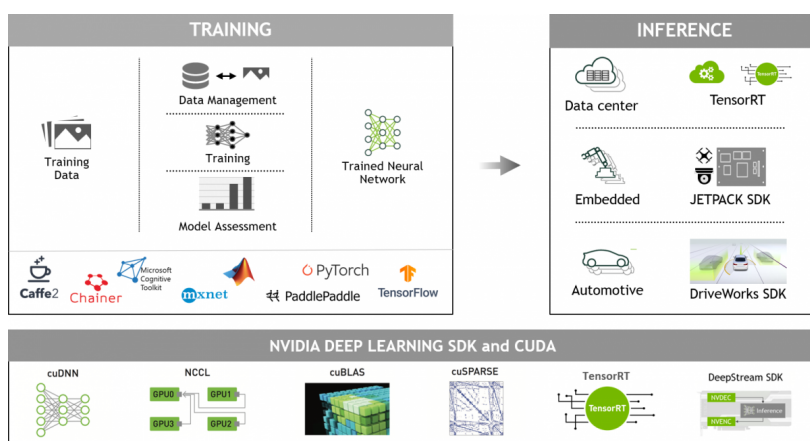


Figure 2.7: Deep Learning Software NVIDIA [19]

GPU accelerated deep learning frameworks offer flexibility to design and train custom deep neural networks and provide interfaces to commonly used programming languages such as Python. Widely used deep learning frameworks such as PyTorch on NVIDIA GPU accelerated libraries to deliver high performance, multi GPU accelerated training (figure 2.7).

Chapter 3

Object Detection using YOLOv5

The object detection algorithm chosen for the scope of this study is *YOLO* (You Only Look Once), version 5 (v5) in detail.

YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm “only looks once” at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions.

As shown in the figure 3.1 made by Wenying Chen et. al, after the analysis of the experimental results above, *RetinaNet* is more suitable if higher recognition accuracy is required, evaluated in mAP (Mean Average Precision) [sec. 4.2], while *YOLOv3* may be more suitable for use when the priority is real-time performance and a slightly lower accuracy can be accepted.

YOLOv5 is different from all other prior releases, as this is a PyTorch implementation rather than a fork from original Darknet. Same as *YOLOv4*, the YOLOv5 has a CSP backbone and PA-NET neck. The major improvements includes mosaic data augmentation and auto learning bounding box anchors.

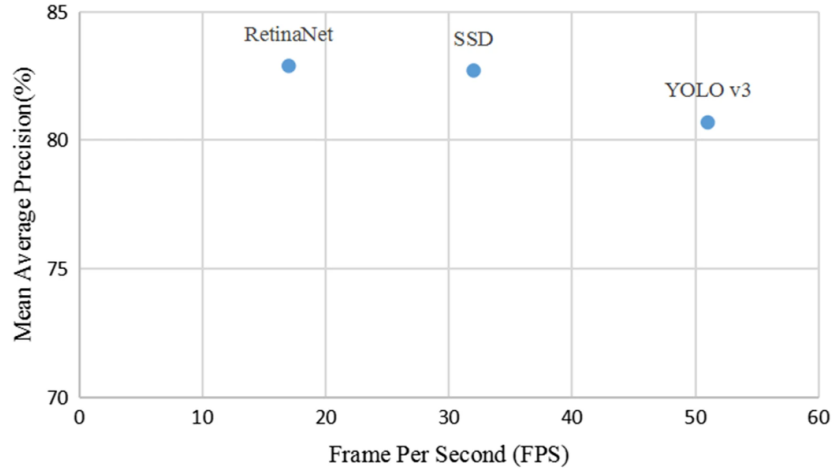


Figure 3.1: Comparison between the object detection models

3.1 YOLOv5

You Only Look Once is one of the most popular model architectures and object detection algorithms Redmon et al. in 2015 [26], presented YOLO, a new approach to object detection. Until then, classifiers were used for object detection, whereas here the creators frame object detection as a regression problem on spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in a single evaluation. **YOLOv5** was implemented by *Ultralytics* [22] in June 2020 and are available 5 pretrained models on COCO dataset with different number of layers and parameters so different results (figure 3.2). Version 5 of the YOLO was chosen because version 6 is 15% [27] slower than version 5, while the later versions 7 and 8 were developed after my work started. The graphic show the huge difference about speed inference on validation between YOLO models and EfficientDet model.

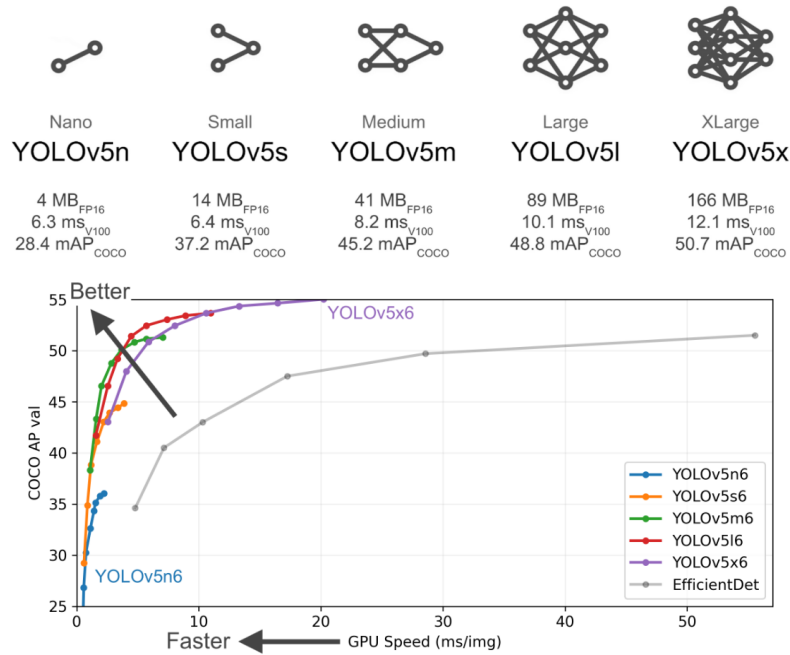


Figure 3.2: Types of YOLOv5 models [22]

All algorithms apply a single neural network to the full image, and then divide the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. It recognizes each bounding box using a tuple composed by four numbers:

- (x_c, y_c) = Center of the bounding box
- w = Width of the box
- h = Height of the box

In addition to that, it predicts the corresponding number c for the predicted class as well as the probability of the prediction.

3.1.1 Architecture

In figure 3.3 the architecture of YOLOv5 model is presented. The image is fed to *CSPDarknet53* (Backbone) for feature extraction and again fed to **PANet** (Neck) for feature fusion. Finally, the **YOLO** layer generates the results.

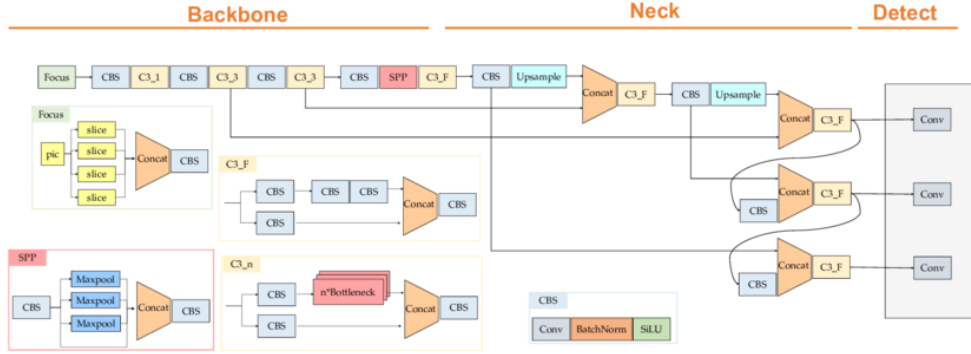


Figure 3.3: Architecture YOLOv5 (Kim et al., 2021 [28])

YOLOv5 is a single-stage object detector (i.e., object localization and classification are applied in the same step) and it has three important parts like any other single-stage object detector (figure 3.3):

1. As **Backbone** is used the **CSP-Darknet53**, it's just the Convolutional Network *Darknet53* used as the backbone for YOLOv3 [29] to which the authors applied the **Cross Stage Partial** (CSP) network strategy. The CSP is based on the same principle of instead of using the full-size input feature map at the base layer, the input will be separated into 2 portions. A portion will be forwarded through the dense block as usual and another one will be sent straight on to the next stage without processed [30].

The primary benefit of CSP is that it enables the network to effectively use the spatial context of an image, by blending features from various parts of the image and various stages of the CNN. This increases the network's robustness to object deformation and scale changes, resulting in more precise object detection. Darknet53 is composed by 53 convolutional layers for performing feature extraction. As shown in the Figure 3.3, the difference between YOLOv3 is the **Focus Layers**, it is the merge by three layers in

the previous version and that are designed to detect specific features in the input image. The focus layer is responsible for identifying potential areas in the image where an object may be located, these areas are called region proposals. These region proposals are then further analyzed by the rest of the network to make the final predictions of object detection. In the Backbone there is also a **CSB** Block, it employs a method known as "cross-scale boundary detection", which detects the edges of objects at various scales and locations, utilizing features from different scales. This enhances the network's robustness to changes in object scales and increases the precision of object detection. The last block in Backbone is **SSP** (Spatial pyramid pooling), it divides the feature maps into a pyramid of regions, each region at a different scale. Then it performs Max-Pooling operation on each region to extract the most important features and concatenates them.

2. **Nek**, The features maps generated by using a top-down architecture with lateral connections allow the **PANet** (Path Aggregation Network) to aggregate features from different scales and different positions of the image, by using a bottom-up architecture with lateral connections. This allows the network to make use of information from different scales of the image. PANet has been developed to enhance the precision of object detection by incorporating a feature pyramid and a path-aggregation mechanism, which enables the network to utilize information from various scales of the image. This makes the network more resistant to variations in object scale and improves the accuracy of object detection.
3. **Head**, the final portion of the **YOLOv5** network architecture, known as the head, is responsible for producing the final object detection predictions. It is made up of various components such:
 - **Detection layers** are responsible for making the final object detection predictions; it is composed from three convolution layers that predicts the location of the bounding boxes, the scores and the objects classes.

- **NMS** (Non-Maximum Suppression) is a technique used to suppress multiple bounding boxes that may have been generated for the same object in an image. It helps to increase the accuracy of the object detection predictions.
- **Anchor Boxes** are pre-established bounding boxes of varying aspect ratios and scales that are utilized to detect objects of different shapes and sizes within the input image. These anchors have an initial size, some of which will be adjusted to fit the size of the object, based on the outputs from the neural network. The network's role is not to predict the final size of the object, but rather to adapt the size of the closest anchor to match the object's size.
- **Loss** is a function that is used to measure the difference between the predicted object and the ground truth object. This is used to adjust the weights of the network during training.

3.1.2 Non Maximum Suppression Algorithm (NMS)

Before explaining in detail the procedure for calculating NMS, let us define a two fundamental metrics that will be used as as a thresholds.

The **IoU** (Intersection Over Union) The IoU measures the accuracy of detections. Given a ground-truth bounding box and a detected bounding box, we compute the IoU as the ratio of the overlap and union areas (Figure 3.4).

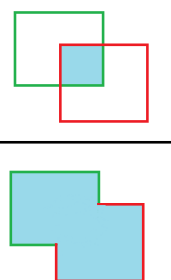
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure 3.4: Intersection Over Union

The other threshold to be set is **confidence**. The confidence threshold is the minimum probability that an anchor box contains an object to be detected.

Define a value for *Confidence Threshold* and *IoU Threshold*, NMS algorithm Loop over all boxes, starting first with the box that has highest confidence and remove the boxes that have a $Confidence < Confidence_Threshold$. In the second step calculate the IOU of the current box, with every remaining box that belongs to the same class, if the $IoU < IoU_Threshold$ remove the boxes.

Algorithm 1 NMS Algorithm

Input: $(n, x_c, y_c, w, h, Class, Conf)$

- 1: Box_List_Thresholded $\leftarrow []$
- 2: Box_ListNew $\leftarrow []$
- 3: Box_Sorted \leftarrow Sort_for_Conf(Input)
- 4: **for all** box \in Boxes_Sorted **do**
- 5: **if** box[Conf] $>$ Th_Confidence **then**
- 6: Box_List_Thresholded \leftarrow box
- 7: **end if**
- 8: **end for**
- 9: **while** len(Box_List_Thresholded) $>$ 0 **do**
- 10: Current_Box \leftarrow Box_List_Thresholded[0]
- 11: Box_List_New \leftarrow Current_Box
- 12: **for all** box \in Box_List_Thresholded **do**
- 13: **if** (Current_Box[Class] = box[Class]) and (IoU $>$ Th_IoU)
- 14: **then**
- 15: remove(Box)
- 16: **end if**
- 17: **end for**
- 18: **end while**

3.1.3 Loss Function

In agreement with Zu et al. [31], the input tensor is split into (S x S) grid cells, in our case (19,19). Each grid cell is responsible for identifying a target if the center point of the target falls within its boundaries. Each grid cell has B anchors box. Specifically, for each anchor, $(5 + C)$ values are predicted, with the first 5 values used to adjust the anchor’s center point position and size. If the center of the target is within the grid cell, it is then determined to be a target, and the position of the target’s bounding box is calculated using the following formula.

$$C_i^j = P_i^j \cdot IoU \tag{3.1}$$

C_i^j is the confidence score of the j bounding box of the i grid. $P_i^j = 1$ if in box j there is a target; otherwise $P_i^j = 0$. the loss function used by this network is the sum of three loss:

1. l_{box} , bounding box regression loss function [3.2]
2. l_{cls} , classification loss function [3.3]

3. l_{obj} , confidence loss function [3.4]

Let's see each loss function in detail.

$$l_{box} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{i,j}^{obj} (2 - w_i \times h_i) [(x_i - \hat{x}_i^j)^2 + (y_i - \hat{y}_i^j)^2 + (w_i - \hat{w}_i^j)^2 + (h_i - \hat{h}_i^j)^2] \quad (3.2)$$

where λ_{coord} is the position loss coefficient and \hat{x} , \hat{y} , \hat{w} , \hat{h} are the ground truth boxes. S is the grid total of the splitted image and for each grid there are B anchor boxes.

$$l_{cls} = \lambda_{class} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{i,j}^{obj} \sum_{C \in cls} p_i \log(\hat{p}_l(c)) \quad (3.3)$$

where λ_{class} is the category loss coefficient and $\hat{p}_l(c)$ is is the true value of the category.

$$l_{obj} = \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{i,j}^{noobj} (c_i - \hat{c}_i)^2 + \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{i,j}^{obj} (c_i - \hat{c}_i)^2 \quad (3.4)$$

If the anchor box at (i, j) contains targets then the value $I_{i,j}^{obj} = 1$, else the value is 0.

So the final function loss is:

$$loss = l_{box} + l_{cls} + l_{obj} \quad (3.5)$$

3.1.4 Anchors Box

Anchor boxes are a collection of bounding boxes with predefined dimensions. They are used to identify and track the size and shape of objects to be surveyed. Prior to training the network, it is necessary to decide which anchor boxes to use according to the data. This process is done by applying a **K-Means** clustering algorithm [32] using Intersection over Union (IoU) as the distance metric. In this work, a K of 9 was selected due to the 10 classes of VisDrone, with the first two (Pedestrian and People) having very similar characteristics.

K-Means is a clustering algorithm for unsupervised learning that organizes data into a specified number of groups, also known as clusters, based on their

similarity. The algorithm splits the data into K clusters, where K is the desired number of clusters, and continuously adjusts the cluster centroids and the data points assigned to them until the sum of distances between the data points and their corresponding cluster centroids reaches minimum.

3.2 Evaluation Metrics

In this section, we will define the algorithm evaluation metrics for multi object detection. The importance of the Confidence Threshold (**Th_Conf**) and the IoU Threshold (**Th_IoU**), which will influence the metrics, was explained earlier. Both score will determine whether a detection is a true positive or a false positive. The flowchart in Figure 3.6 shown as the choice of true positive detection depends on the both score.

The input is the tuple of bounding box predicted $(x, y, w, h, Conf, Class)$, the first step is to check whether the predicted confidence (Conf) is above the chosen confidence threshold (Th_Conf). If it does not exceed the threshold, it is necessary to check whether the box actually exists as ground truth, if not, the predicted box represents a False Negative (**FN**) otherwise it will be a True Negative (**TN**). If, on the other hand, the threshold is exceeded then there is a target in the image and we proceed to the second step. If the predicted class (Class) matches the class of a ground truth and the predicted bounding box has an IoU greater than a threshold (in this case 0.5) then a detection is considered True Positive (**TP**) else it will be False Positive (**FP**).

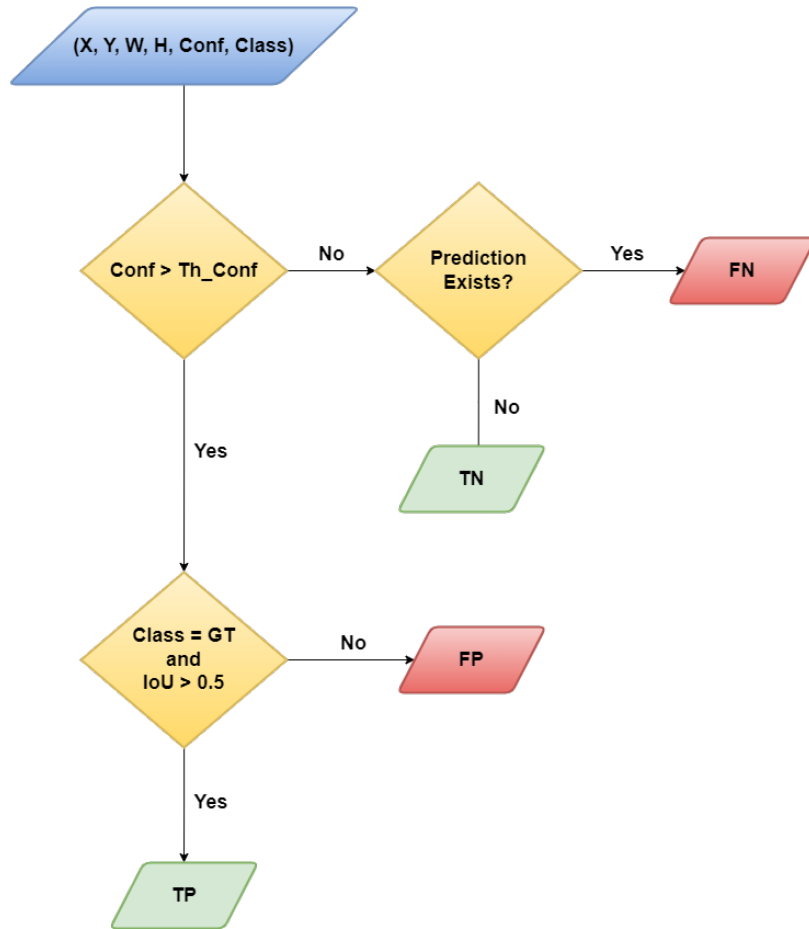


Figure 3.5: Flowchart to evaluate the boxes predicted

Algorithm 2 Algorithm to evaluate true positive detections [33]

- 1: **for all** detection that has a confidence score $> Th_Conf$ **do**
 - 2: choose one that belongs to the same class and has the highest IoU
 - 3: **if** no ground-truth can be chosen or $IoU < Th_IoU$ **then**
 - 4: the detection is a False Postive
 - 5: **else**
 - 6: the detection is a True Positive
-

That said, we can go on to define the **Precision** (P) as the number of true positives divided by the sum of true positives and false positives:

$$Precision = \frac{TP}{TP + FP} \quad (3.6)$$

The **Recall** (R) as the number of true positives divided by the sum of true

positives and false negatives:

$$Precision = \frac{TP}{TP + FN} \tag{3.7}$$

By setting the threshold for confidence score and IoU at different levels, we get different pairs of precision and recall. These two classical metrics are not sufficient to evaluate the models in the mod, so a new metric is introduced.

These two classical metrics are not sufficient to evaluate the models in the mod, so a new metrics is introduced. The precision-recall curve is a useful tool for evaluating detector performance, but it can be difficult to compare different detectors when their curves intersect. A numerical metric, such as **Average Precision** (AP), can be used instead. AP calculates the average precision across all unique recall levels and is based on the precision-recall curve. AP can then be defined as the area under the interpolated precision-recall curve, for the interpolation 11 levels of recall are considered. This metric is calculated as following formula [33]:

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interp}(r_{i+1}) \tag{3.8}$$

where the p_{interp} is the interpolated precision at a certain recall level $r' \geq r$:

$$p_{interp}(r) = \max_{r' \geq r} p_{r'} \tag{3.9}$$

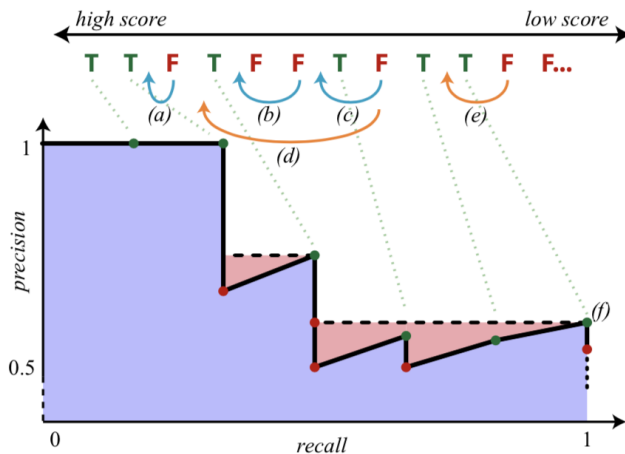


Figure 3.6: Average Precision

The calculation of AP only involves one class, in this task of MOD it will be necessary to find the average across all classes so the Mean Average Precision (**mAP**) will be used.

$$mAP = \frac{1}{K} \sum_{i=1}^K AP_i \quad (3.10)$$

For the VisDrone Challenge will be used 2 variants of mAP:

- $mAP^{IoU=0.5}$, obtained by setting an IoU threshold of 0.5
- $mAP^{IoU=0.5:0.05:0.95}$, which is mAP averaged over 10 IoU thresholds from 0.5 to 0.95 with 0.05 step. This will be the most important evaluation metric of the challenge

3.3 VisDrone Dataset

Object detection is a fundamental aspect of many advanced computer vision applications such as self-driving vehicles, facial recognition, and activity recognition. Despite recent advancements, these algorithms often focus on general scenarios rather than those captured by drones. This is due to the lack of publicly available large-scale benchmarks or datasets for drone-captured scenes, which has limited the research in this area. This challenge is still active today and the **Dataset VisDrone 2019** [34] was used for this work. VisDrone is a dataset that captures real-life scenarios from drones and its particular difficulties are a high class imbalance and targets that can be even less than 1% of the image. It is composed by 8.599 images divided into **6.471** for **training**, **548** for **validation** and **1.580** for **testing**. There are **10** object **categories** of interest including pedestrian, people, car, van, bus, truck, motor, bicycle, awning-tricycle, and tricycle.

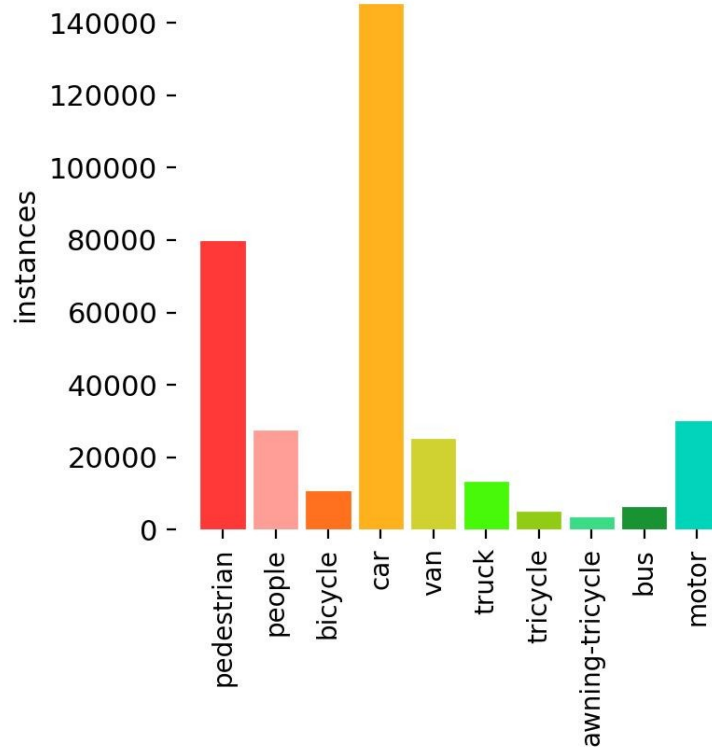


Figure 3.7: Instances of VisDrone

From the histogram in Figure 3.7, it is possible to see one of the main problems of this challenge, the categories of objects to be detected are very unbalanced; for example, there are about 150,000 instances of cars and only 2,000 of tricycles.

The other difficulty is related to the size of the targets to be detected, as shown in the Figure 3.8, at the top the histogram (x, y) has an approximately normal distribution, so the detection model can detect the discriminant parts more flexibly. From the (x, width) and (y, height) patches can be seen that the local parts are evenly distributed on both axes. Interesting to see in in the $(\text{width}, \text{height})$ patch that almost all parts only cover a rather small area of the whole image.

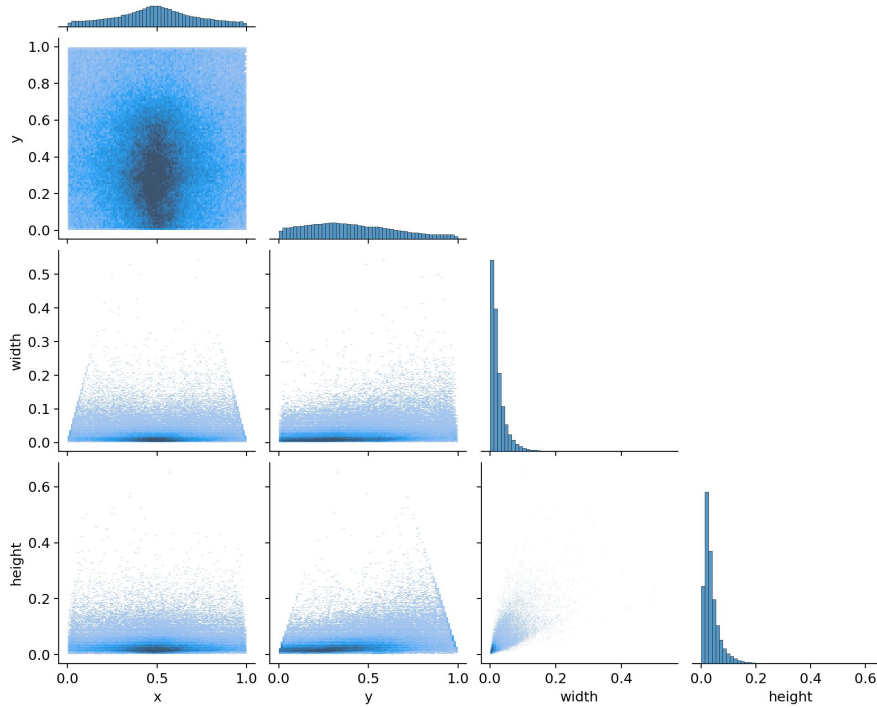


Figure 3.8: Labels Correlogram VisDrone

In the results of the VisDrone Challenge 2019 [35], the best 33 algorithms are presented in order of AP. In the table 42 some models are shown, the AP metrics is $mAP^{Iou=0.5:0.05:0.95}$ while the $mAP^{Iou=0.5}$ is AP50.

Position	Model	AP [%]	AP50 [%]	GPU	Speed [fps]
1	DPNet	29,62	54	Titan XP	6
2	RRNet	29,13	55,82	RTX2080ti	1,5
3	ACM-OD	29,13	54,07	Tesla V100	0,5
32	DBCL	16,78	31,08	Titan XP	2,5

Table 3.1: Challenge Results VisDrone 2019

3.3.1 DPNet

The architecture of **DPNet** [36] is dual-path and allows the parallel extraction of high-level semantic features and low-level object details. Although DPNet has an almost duplicated form compared to single-path detectors, the computational costs do not increase significantly. To improve representation capability, they introduced a light autocorrelation module (LSCM) for dependencies between neighbouring scale features.

The main difference with the YOLOv5 architecture is in the backbone. Both networks divide the input into two paths for feature extraction at different levels. An important difference that makes YOLOv5 faster is the initial Focus Layer, which allows a path to only analyse the regions proposed by the latter. Furthermore, it uses the CSP strategy in the backbone, concatenating features at various points in the network at different levels and using the SSP pyramid block finally allows max pooling for each proposed region at different scales.

3.4 Implementation

For this development phase was used a **GPU NVIDIA Quadro P4000** [16] which combines a 1792 CUDA core Pascal GPU, large 8 GB GDDR5 memory and advanced display technologies to deliver the performance and features that are required by demanding professional applications. Using the tools and technology described in 2, a docker container was prepared which the GitHub that containing the original code of YOLOv5 network was pulled. As described in section 3.1 are available 5 models of YOLOv5 trained on the COCO Dataset. The results of models validated on this dataset are as follow:

Therefore pre-trained models were used using the transfer learning technique for this task. Transfer learning is an effective technique that lets us utilize existing models to quickly adapt to new tasks with limited resources. This can be facilitated by manipulating certain layers of the target model and leaving others intact. To make this process as efficient as possible, some layers are fine-tuned to fit the new dataset and task, while others remain unchanged.

Model	Size [Pixel]	mAP:50-95 [%]	mAP:50 [%]
YOLOv5n	640	28	45,7
YOLOv5s	640	37,4	56,8
YOLOv5m	640	45,4	64,1
YOLOv5l	640	49	67,3
YOLOv5x	640	50,7	68,8

Table 3.2: Validation Results on COCO

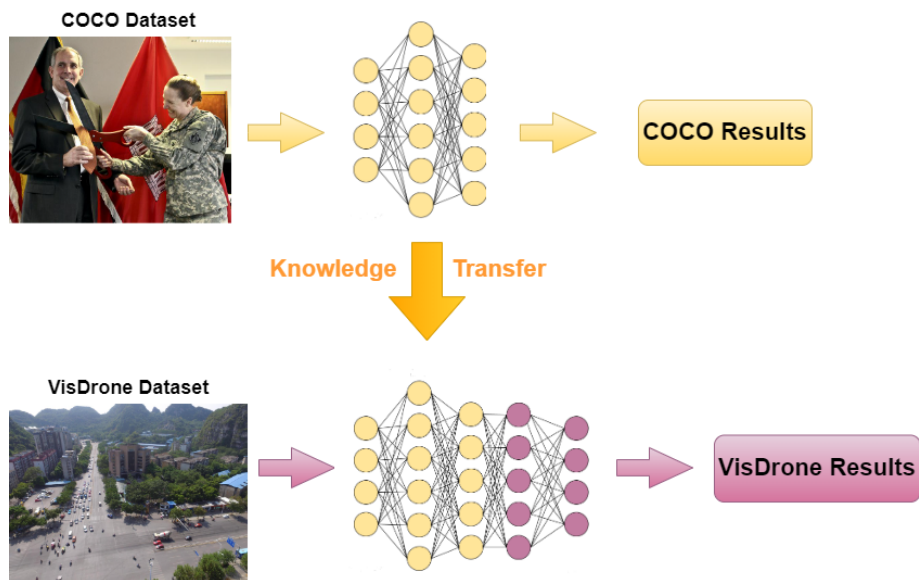


Figure 3.9: Transfer Learning

3.5 Training

After studying the the limits of VisDrone Dataset (sec. 3.3), proceeded with train phase of five models. The first training was carried out in order to choose the confidence threshold most suitable of this challenge. The results shown in Figure 3.10a indicate how increasing confidence improves accuracy but worsens recall 3.10b. The importance of mAP, a metric taken into account by the creators of the challenge, was explained in section 3.2, so the aim is to increase the air under the blue curve in the Precision-Recall graph 3.10c. The blue lines represents the

mean of all classes.

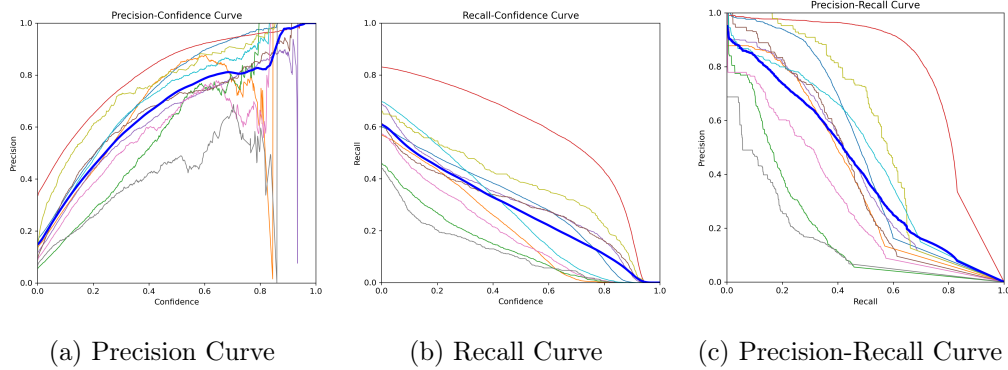


Figure 3.10: Precision and Recall as the confidence changes

By increasing the confidence threshold, we could increase the mAP metric by significantly reducing the recall, but this would lead to insensitivity. After training on the simplest model (**YOLOv5n**) was validated at different confidence thresholds, it can be seen from the figure 3.11 that the choice of 0.25 has a good relationship between precision and recall generating an increase in mAP.

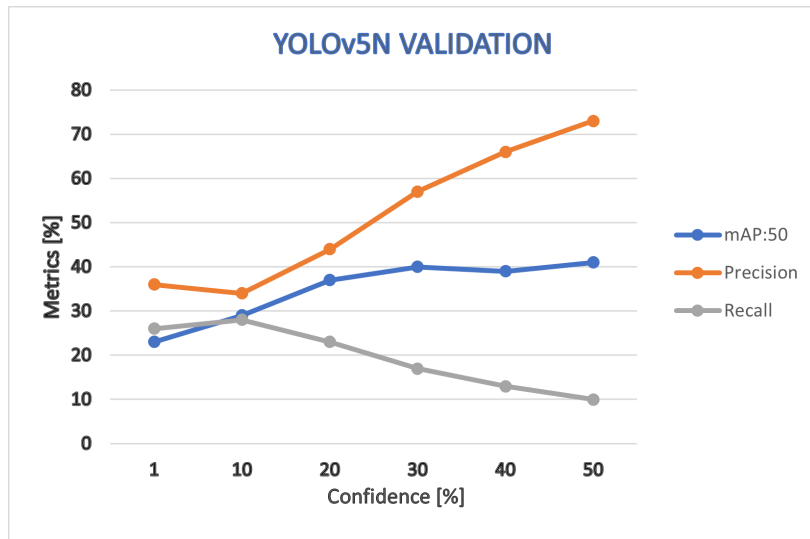


Figure 3.11: Validation YOLOv5n

Having chosen a **confidence threshold = 0.25** and an **IoU threshold = 0.5** all models were trained keeping the configuration unchanged:

- Image size = (640 x 640)
- Batch size = 16
- Epochs = 100
- Optimizer = SGD (Stochastic Gradient Descent)
- Learning Rate = 0,01
- Weight Decay = 0,0005

Gli iperparametri sono stati scelti dopo un'ottimizzazione attraverso un algoritmo genetico (GA). The Genetic Algorithm [37] is an optimisation algorithm based on principles inspired by evolutionary biology, such as natural selection and genetic mutation. This algorithm is used to solve optimisation problems that require finding an optimal solution in a space of potential solutions. The evolution is performed on a base scenario that is attempted to be improved. In this case, the base scenario is the tuning of VisDrone for 10 epochs, using pre-trained YOLOv5s.

Below are the results of the validation of the models on 548 images, note how the choice of the confidence of 0.25 shows a good trade off between precision and recall in the **YOLOv5x** model, which will also prove to be the best for the test challenge. These may not seem like good results, but it should be remembered that the model trained on COCO had **50,7%** of **mAP:50-95 3.2** in validation.

YOLOv5 [size]	Precision [%]	Recall [%]	mAP:50 [%]	mAP:50-95 [%]
N	51,6	22,3	37,2	20,8
S	52,9	27,8	40,3	25,1
M	53,3	35,9	44,7	27,9
L	54,1	40,1	47,1	30,3
X	55,2	42,4	49,2	32,3

Table 3.3: Validation Results on VisDrone

3.6 Testing

Having performed the training and validation phase of the models, this section will report the results obtained in testing on the VisDrone Dataset. The aim is to achieve the accuracy of the models evaluated in the challenge [35] studying the trade-off between accuracy and speed of inference. Below are the results obtained on the 1.580 test images from each model:

YOLOv5 [size]	Th_Conf [%]	Precision [%]	Recall [%]	mAP:50 [%]	mAP:50-95 [%]	Speed [fsp]
N	25	44,6	20,4	32,3	18	104
S	25	47	28,1	37,3	21,7	95
M	25	46,9	33,6	39,7	23,9	57
L	25	48,3	36,3	41,5	25,7	32
X	25	49,4	37,7	43,3	27,4	18
X	45	66,3	28,6	45,8	29,9	18

Table 3.4: Test Results on VisDrone

Checking the results of the challenge shown in Table 3.4, the first place winner shows a **mAP:50-95 = 29.6%**. The tests show that with the thresholds set in the validation phase ($Th_conf = 0.25$ and $Th_IoU = 0.5$), the model is comparable with the first runners-up in the challenge. By increasing the confidence threshold we can see that our **mAP:50-95 = 29.9%**, exceeds that of the DPNet model (first runner-up) penalising the recall of the model. Particular attention should be paid to the model's inference speed, which is three times faster than that of DPNet (**6 fps**), despite the fact that the GPU they use is much more powerful than the NVIDIA Quadro P4000 used. A further test to increase accuracy by reducing the speed was done by increasing the image resolution from (640x640) to (1028x1028) achieving a **2%** increase in mAP:50-95 but doubling the inference time. The speed evaluated in fps is the sum of three times evaluated

in *ms*:

$$speed = \frac{1000}{t_PreProcess + t_inference + t_NMS} \quad (3.11)$$

t_PreProcess is the time to resize the input image, the **t_inference** is the actual detection time while **t_NMS** is the time taken by the NMS algorithm 3.1.2 to evaluate the boxes.



(a) Label

(b) Prediction

Figure 3.12: Prediction on the test frame

As shown in Figure 3.12 the greatest difficulty is the detection of small objects in the background that represent less 1% of whole image.

3.7 Optimization

The objective of this section is to seek a solution to reduce the model's inference time while maintaining constant accuracy. For this reason, the technique used is the quantization of weights through the TensorRT framework. **Quantization** is a technique used to reduce the memory and computational requirements of deep learning models by representing the model's parameters and activations with fewer bits. **TensorRT** is a library developed by NVIDIA that optimizes models for deployment on NVIDIA GPUs. The library includes support for quantization, allowing developers to quantize their models and run them on devices with reduced memory and computational requirements. This can help to speed up the performance of deep learning models and make them more suitable for deployment on embedded. In particular, a **Post Training Quantisation** was used, which uses a calibration dataset to determine the quantisation parameters, such

as the scale factor and zero point, and does not require re-training, making it easier to include in a pipeline. The goal of quantisation is to reduce the memory footprint and computational requirements of a model without sacrificing too much accuracy. This technique is one of the most widely used to reduce the memory and computational requirements of neural networks, making it possible to implement them on devices with limited resources, such as smartphones and embedded systems. TensorRT applies post training quantization to compute quantization elements for each layer and utilizes the KL divergence (Kullback-Leibler divergence), is a measure of the difference between two probability distributions. The process of training involves using FP32 precision for parameters and activations. Optimizing these results by converting them to FP16 or INT8 precision not only reduces latency, but also gives a significant decrease in the size of the model.

	Dynamic Range	Min. Positive Value
FP32	$[-3,4 \cdot 10^{38}, +3,4 \cdot 10^{38}]$	$1,4 \cdot 10^{-45}$
FP16	$[-65.540, +65.504]$	$5,96 \cdot 10^{-8}$
INT8	$[-128, +127]$	1

As the activity diagram 3.13 shows, the models were first exported as graphs in ONNX (Open Neural Network Exchange), allowing to be trained and deployed with any framework, using any hardware, and in any language, then converted to FP32 (Floating Point 32 bit) engine with TensorRT and then performed a quantization, first in **FP16** (Floating Point 16 bit) and then in **INT8** (Integer 8 bit), finally the testing phase to study the trade-off between speed and accuracy achieved.

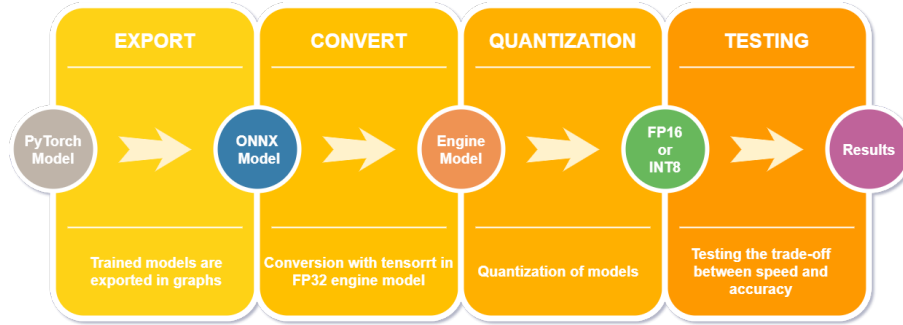


Figure 3.13: Diagram for quantization of models

As explained in the article [38], converting some of our weights to FP16 (a lower precision) can cause overflow due to the lower dynamic range of FP16 compared to FP32 (Table 3.5). However, experiments have demonstrated that this does not have a major effect on accuracy. In general, weights and activation values are robust to noise. Training a model involves preserving the features needed for prediction, which is something the model does naturally. This approach of limiting exceeding weights won't be successful when changing to INT8 precision. Since INT8 values are very limited [-127 to +127], most of the weights will be adjusted and overflow when using a lower precision, leading to a large decrease in accuracy of the model (Table 3.5).

YOLOv5	TensorRT	Size [Mb]	mAP:50 [%]	mAP:50-95 [%]	Speed [fps]
N	FP32	10	32,3	18,1	157
N	FP16	10	31,7	17,6	181
N	INT8	4	24,3	12,6	227
S	FP32	34,6	37,3	21,7	98
S	FP16	34,4	36,6	20,9	105
S	INT8	9,4	29,4	15,9	169

Table 3.5: Quantization Results (MOD)

The models *YOLOv5n* and *YOLOv5s* in PyTorch had a speed of 104 fps and

95 fps, respectively (Table 3.4), after export and quantization with TensorRT; from the table 3.5 it can be seen that the representation with FP32 also increases the speed while leaving the accuracy (evaluated in mAP) unchanged. Quantization in INT8 is very fast but not suitable as a solution as it greatly decreases accuracy, while quantisation of the weights in FP16 shows a good trade-off between speed and accuracy. Quantisation was only performed for the first two simpler models, as the other three were composed of larger layers and matrices are restricted by their bandwidth (memory limitations). This implies that their execution devotes most of its time to reading and writing data, thus making it impossible to reduce their overall runtime by decreasing computation time.

Chapter 4

Object Tracking adding strongSORT

In section 2.3, we explain the importance of Multi Object Tracking (MOT) in the military sector and beyond. This is also an open challenge that many researchers are working on. Doing reconnaissance from UAV means being able to detect and track objects at the same time. Therefore, the goal of this chapter is to look for a solution that can perform tracking with detection using the *YOLOv5* model seen in Chapter 3.

4.1 Architecture

Therefore, the proposed architecture is as shown in figure 4.1

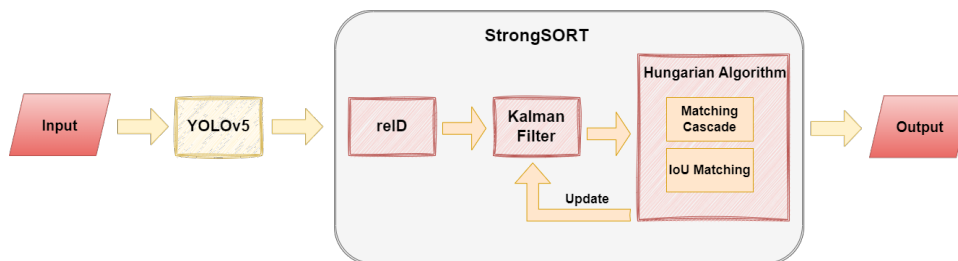


Figure 4.1: Multi Object Tracking Architecture

Let us look in detail at the blocks into which it is composed:

- **Input**, is a video sequence frames
- **YOLOv5**, the network is that explain in chapter 3, it deals with Multi Object Detection, and its output consisting of the co-ordinates of the boxes and the predicted classes enters the **strongSORT** algorithm
- **reID (Re-Identification)** is used to match the tracked objects over multiple frames, which helps to maintain the identity of an object even if it has moved away or disappeared in some frames. It is a technique used to identify objects in multiple frames of video footage. It works by comparing the features of the objects, such as color, shape, and texture, and assigning them a score to decide if they are the same object. This helps to keep track of the identity of an object even when it moves out of sight or is occluded, so that they can still be monitored. A convolutional network called OSNet (Omni-Scale Network) was used for this task.
- **Kalman Filter**, it propagates the detections from the current frame to the next which is estimating the position to appraise the condition of a system based on the data currently accessible. It is used to minimize the disturbance in the measurement data by using a weighted mean of the preceding measurement and the existing measurement. The Kalman filter is made up of two equations: the forecast equation, which is used to predict the state of the system, and the update equation, which is used to adjust the assessment of the system state based on the new measurement data.
- **Hungarian Algorithm**, it is an efficient method of solving assignment problems, by means of finding the ideal pairing between two sets of elements. It works through associating the lowest possible cost to each element, based on a cost matrix. This cost matrix is derived from the expense of assigning each element to its counterpart.
- **Output**, it is represented by the previously detected bounding box coordinates and assigned tracking id.

4.1.1 Re-Identification

The task of reidentification, also known as re-ID, is a problem of recognizing individuals at the instance level. It requires features that are discriminative and can capture various spatial scales, as well as combinations of multiple scales. These features, known as omni-scale features, can be of both homogeneous and heterogeneous scales. In the study by Zang et al. [39], a deep re-ID CNN called the omni-scale network (**OSNet**), was designed. It is used for learning omni-scale features, to improve efficiency and prevent overfitting. The OSNet utilizes point-wise and depthwise convolutions. Despite its small size, the OSNet has achieved state-of-the-art performance on six person re-ID datasets and outperforms many larger models by a significant margin.

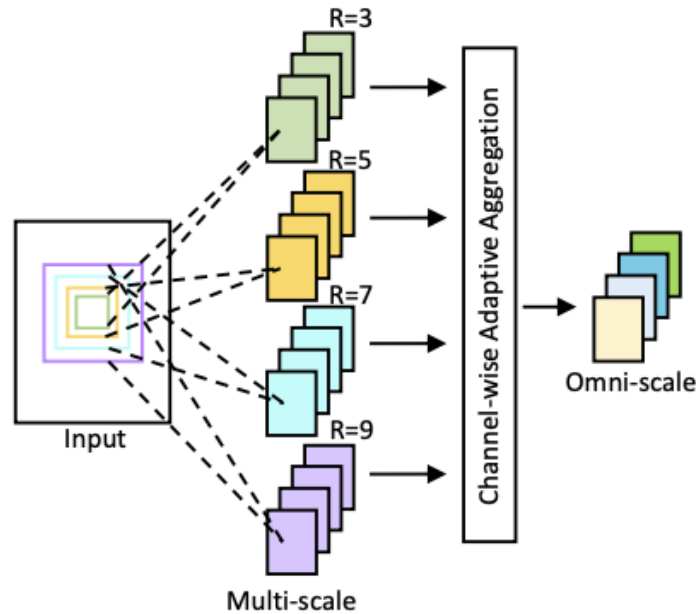


Figure 4.2: OSNet Architecture by Zhou et al. [39]

The CNN incorporates multiple scale features within each building block using dynamic channel-wise weights that allow it to learn various multi-scale combinations. As a result, the OSNet has the ability to learn omni-scale features, with each channel potentially identifying distinctive features of a single scale or a combination of multiple scales. The fundamental parameters (called Max_Age)

that must be set for this model is the maximum number of missing frames for each trajectory before it is deleted. It is clear that increasing the number will also increase the speed of inference in extracting the characteristics of each bounding box.

4.1.2 Hungarian Algorithm

The Hungarian algorithm, also known as the Kuhn-Munkres algorithm, can link an obstacle from one frame to another based on a score. There are various scores that can be used, such as IoU (Intersection Over Union) which compares the overlap of bounding boxes to determine if they are the same object. Another score is the Shape Score, which takes into account the similarity in shape or size between two consecutive frames. Additionally, a Convolution Cost can be used, which involves running a CNN on the bounding box and comparing it to a previous frame. If the convolutional features are similar, it is likely the same object, even if there is partial occlusion. This matching mechanism based on IoU cost matrix into a mechanism of Matching Cascade and IoU matching, specifically the core idea of Matching Cascade is to give greater priority to track matching to the targets that appear more frequently in the long-term occluded targets. As explained by Tithi et al. [40], the Hungarian algorithm is used in a matrix format. Given a non-negative matrix with dimensions of $n \times m$, where each element represents the cost of assigning a predicted object to a measured object, the Hungarian algorithm finds the best assignment of predictions to measurements such that each prediction is matched to one measurement and each measurement is matched to one prediction, while minimizing the total cost of the assignment. The algorithm works by subtracting the minimum value of each row and column, and then identifying the unique $[i, j]$ entries with a value of 0, which are then assigned as a match between prediction j and measurement i . If there are multiple 0s in a row or column, the process is repeated by subtracting the global minimum from all remaining entries.

For example [41], we have 3 detections and need to associate these detections to the 3 tracks based on IoU score. A detection will belong to a track that has the

maximum IoU. We will solve the assignment optimization problem between the detections and tracks using the Hungarian algorithm by finding the maximum IoU.

	Track1	Track2	Track3
Detection1	80	20	35
Detection2	30	15	19
Detection3	60	48	78

To solve the maximization problem, it was converted into a minimization problem by finding the maximum value in the matrix and subtracting it from each cell.

	Track1	Track2	Track3
Detection1	0	40	45
Detection2	50	65	61
Detection3	20	32	2

Now we apply the Hungarian algorithm to the grid step by step:

1. Step 1: Subtract row minima

	Track1	Track2	Track3
Detecion1	0	40	45
Detecion2	0	15	11
Detection3	18	30	0

2. Step 2: Subtract column minima

	Track1	Track2	Track3
Detection1	0	40	45
Detection2	0	15	11
Detection3	18	30	0

3. **Step 3: Cover all of the zeros in the matrix using the minimum number of lines**, Draw lines through the row and columns that have the 0 entries such that the fewest lines possible are drawn.

	Track1	Track2	Track3
Detection1	0	40	45
Detection2	0	15	11
Detection3	18	30	0

4. **Step 4: Create Additional Zeros**, To find the optimal solution using the Hungarian algorithm, the first step is to locate the smallest entry that is not covered by any line. Then, this entry is subtracted from each row that is not crossed out and added to each column that is crossed out. This process is repeated until each row and column only has one zero selected. This final solution is the one that associates detections with tracks using the maximum IoU.

	Track1	Track2	Track3
Detecion1	80		
Detecion2	30		
Detection3			78

Then the IoU threshold set (in our case 0.5) will be checked to see if it is a true positive.

4.2 Evaluation Metrics

New metrics were used to evaluate the multi object tracking model in addition to the traditional ones [24]. Let us explain some acronyms that will be used by the metric, as **IDTP** (True Positive ID trajectory), it is the longest associated trajectory matching to a ground truth trajectory is regarded as the gt's true ID. Then other trajectories matching to this gt is regarded as a **IDFP** (False Positive

ID trajectory) or **IDFN** (False Negative ID trajectory). **GT** is the number of reference (GroundTruth) trajectories and **MT** is the number of trajectories that have over 80% target tracked. **FM** is the Number of Fragmentations, ID switch is the special case of fragmentation when ID jumps.

- **ID Precision**, it shows the precision in tracking the trajectories of objects

$$IDP = \frac{IDTP}{IDTP + IDFP} \quad (4.1)$$

- **ID Recall**, it shows the precision in tracking the trajectories of objects

$$IDR = \frac{IDTP}{IDTP + IDFN} \quad (4.2)$$

- **Recall**, it shows the recall in detection

$$R = \frac{TP}{TP + FN} \quad (4.3)$$

- **Precision**, it shows the precision in detection

$$P = \frac{TP}{TP + FP} \quad (4.4)$$

- **PT**, the number of trajectories that have 20% to 80% target tracked

$$PT = GT - MT - ML \quad (4.5)$$

- **ML**, the number of trajectories that have less than 20% target tracked.
Total false positive number among all frames

$$FP = \sum_t \sum_i fp_{i,t} \quad (4.6)$$

- **FN**, total false negative number among all frames

$$FN = \sum_t \sum_i fn_{i,t} \quad (4.7)$$

- **IDs**, ID switch number, indicating the times of ID jumps

$$IDs = \sum_t ids_{i,t} \quad (4.8)$$

- **MOTA** (Multiple Object Tracking Accuracy), is a metric reflects the tracking accuracy. It has intergrated consideration of FN, FP, and IDs

$$T = \sum_t \sum_i gt_{i,t}$$

$$MOTA = 1 - \frac{FN + FP + IDs}{T} \quad (4.9)$$

- **MOTP**, (Multiple Object Tracking Precision), is a metric reflects the tracking precision

$$MOTP = \frac{\sum_{i,t} IoU_{t,i}}{TP} \quad (4.10)$$

4.3 Testing

Various tests were performed on different video sequences, like P-Destre Dataset [42]. Two tests inherent to our challenge for images from drones are reported in this section. In the first case we will analyse a video sequence with a single person to see how the algorithm works in the case of single object tracking, then we will analyse a video from the VisDrone Tracking 2019 dataset. In the first test of **Single Object Tracking** the detection’s model was limited to predict only one box for each frame.



(a) Tracking predicted $t = 0$

(b) Tracking predicted $t = 6$

Figure 4.3: Single Object Tracking

As shown in Figure 4.3, the same person, the one with the highest confidence score assigned by *YOLOv5*, is detected in both frames with the same ID (1) assigned by the *strongSORT* algorithm for its entire trajectory To evaluate the accuracy of this algorithm was calculated:

Models	GT	MT	Speed [fps]	P [%]	R [%]	IDP [%]	IDR [%]
YOLOv5N + OSNet	1	1	37	91,7	86,7	90,6	87,5
YOLOv5S + OSNet	1	1	39	93,8	89,9	91,7	88,9
YOLOv5X + OSNet	1	1	14	96,2	92,4	93,2	89,9

Table 4.1: Single Object Tracking Results

The results on the table 4.1 show that the GroudTruth trajectory is only one (GT) and the algorithm tracks one for more than 80% of its duration. The P (Precision), R (Recall) metrics indicate the accuracy of the detection model while IDP and IDR indicate the accuracy of the tracking algorithm. The results are satisfying, which is attributed to the analysis of the pedestrian class where both the detection and re-identification models exhibit high accuracy.

The most important test is the following, which shows how the algorithm behaves in the MOT task when analysing several classes. The figure 4.4 shows the two frame of one sequence video taken to VisDrone Tracking 2019. You can see how YOLOv5 manages to detect all objects present and also to distinguish the van class from car, while keeping the tracking IDs unchanged.

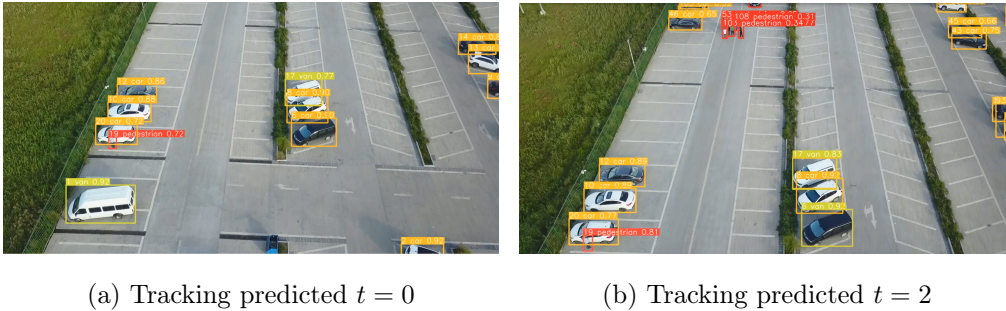


Figure 4.4: Multi Object Tracking

Subsequently, the frames pick up a much larger number of objects and the network makes some errors. Table 4.2 summarizes the obtained results.

Models	GT	MT	PT	Speed [fps]	P [%]	R [%]	MOTP [%]	MOTA [%]
YOLOv5N + OSNet	122	80	36	15	83,5	81,2	85,3	59,2
YOLOv5S + OSNet	122	82	37	13	85,3	82,3	85,8	61,9
YOLOv5X + OSNet	122	98	16	4	90,1	85,8	85,9	61,8

Table 4.2: Multi Object Tracking Results

As we can see from the results obtained, taking the last line describing the results obtained using YOLOv5x as detector, out of 122 trajectories we find 98 more than 80% of their actual length, 16 are partially detected and 3 are not detected. This time the MOTP and MOTA [sec.4.2] were considered to evaluate the tracking algorithm, and it can be seen that by changing the YOLOv5 model we only succeed in increasing the P and R of the detection but the tracking metrics remain unchanged.

4.3.1 Optimization

We saw from table 4.2 that the algorithm for the MOT has a high computational cost, so both models (YOLOv5 [3.1] and OSNet [4.1.1]) were optimised independently. A quantization of the tensor weights of the networks was performed using *tensorRT* as seen in sec 3.7. The speed this time was calculated by adding the tracking time of the strongSORT algorithm to the sum in the equation 3.11, which is generally 3 times longer than the detection time.

Models	tensorRT	GT	MT	PT	Speed [fps]	P [%]	R [%]	MOTP [%]	MOTA [%]
YOLOv5N + OSNet	FP32	122	80	36	18	83,5	81,2	85,3	59,2
YOLOv5N + OSNet	FP16	122	80	36	19	83,3	81,3	85,1	58,9
YOLOv5N + OSNet	INT8	122	60	14	29	58,1	45,8	55,9	29,3
YOLOv5S + OSNet	FP32	122	82	37	16	85,3	82,3	85,8	61,9
YOLOv5S + OSNet	FP16	122	82	37	17	85,1	82,2	85,4	61,3
YOLOv5S + OSNet	INT8	122	63	15	25	59,2	47,8	56,2	30,6

Table 4.3: Quantization Results (MOT)

As we saw in the optimisation results in Chapter 3, for this task too, the FP32 and FP16 quantisation shows an increase in speed without reducing the accuracy of the models, unlike the cast at INT8, which is still the fastest but the least accurate.

Chapter 5

Deploy

In this chapter i will be described how to deploy object detection models on **NVIDIA Jetson TX2** and the results of the models on the VisDrone test dataset performed on this board will be compared. In addition, some tests done in real time through the camera on the board will be reported. For OS installation on Jetson TX2 card, Linux Ubuntu 18.04LTS on host machine is required to be able to download and install NVIDIA SDK Manager [43] which offers an end-to-end development environment configuration solution for the NVIDIA DRIVE SDKs. After installation is complete, the board will use a custom distribution called Linux For Tegra (L4T) derived from Ubuntu 18.04. SDK Manager allows you to choose the frameworks to be installed, in our case PyTorch, TensorRT and DeepStream were selected. Subsequently, object detection models exported in ONNX universal format (graphs) were deployed in the board via SSH (Secure Shell Protocol).



Figure 5.1: Secure Shell Protocol

5.1 NVIDIA Jetson TX2

The NVIDIA Jetson TX2 is a computer on a single board that is specifically built to handle artificial intelligence and machine learning tasks. It boasts of powerful GPU, CPU, memory and other components that are optimized to deliver top-notch AI performance, with support for multiple AI development frameworks and programming languages, as well as a range of connectivity options for sensors and other peripherals, the Jetson TX2 is a versatile AI solution. The main modules that make it very power are [44]:

- **256 core NVIDIA Pascal GPU**, supports all the same features as discrete NVIDIA GPUs, including extensive compute APIs and libraries including CUDA.
- **ARMv8 (64-bit) Multi-Processor CPU**: two cluster connected, the Denver 2 (Dual-Core) CPU clusters is optimized for higher single-thread performance while the ARM Cortex-A57 (Quad-Core) clusters is better suited for multi-threaded applications and lighter loads.
- **Memory**, 8GB LPDDR4 integrated on the module.
- **Storage**, 32 GB eMMC memory integrated on the module.
- **Camera**, recording of 4K ultra-high-definition video at 60fps.

5.2 Testing

After installation of the operating system and the necessary frameworks, a virtual environment was prepared containing all the libraries required for testing the networks. The first tests were performed on the VisDrone dataset, as done in sec.3.6, using the two models with the shortest inference time (YOLOv5n and YOLOv5s) in order to analyze their execution speed on the board.

YOLOv5	TensorRT	mAP:50 [%]	mAP:50-95 [%]	GPU [fps]
N	FP32	32,1	18	39
N	FP16	31,6	17,6	42
N	-	32,2	18,1	26
S	FP32	37,1	21,4	24
S	FP16	37,1	20,8	29
S	-	37,2	21,4	21

Table 5.1: Test Results on Jetson TX2

It can be seen from the 5.1 table that the mAP results are unchanged, but we experience an increase in inference speed due to lower GPU power on the card. In the results seen in table 3.5 performed with a P4000 GPU, the optimised *YOLOv5n* analysed between 150 and 220 frames per second, in this case we arrive at a maximum of 40. It should be noted that NVIDIA Jetson TX2 does not support quantisation in INT8 so the results of the models with weights in FP32, FP16 in TensorRT and those of the original PyTorch model in FP32 have been reported. This is the maximum power the board can achieve by putting it in MAXN mode, i.e. turning on all CPU and GPU clusters without saving on battery efficiency.

5.3 Real-Time Testing

First the camera on board was configured with **GStreamer**, a framework for creating multimedia pipelines. Then a GStreamer pipeline was created which captures a video from the camera by executing the commands *v4l2src* to capture the video, *decodebin* to decode the video and *xvimagesink* to display the output. After checking the operation, the GStreamer pipeline has been inserted in the YOLOv5 code to acquire images in real time. For the first real-time test, a YOLOv5n [sec.3.1] model trained on COCO dataset was used, the one from which we started for the development. A real-time inference rate of 40 fps was ob-

tained using the COCO optimized model by filming a laboratory room where the measured classes were: chair, TV, bottle and person. A real time inference rate of 40 fps was achieved using the COCO-optimised model by filming a laboratory room in which the classes measured were: chair, TV, bottle and person.

To test our networks trained on visdrone in real time, it would be necessary to mount the board on a drone, the **DeepStream** framework was used to solve this problem. The **NVIDIA DeepStream SDK** is a high-performance and scalable AI framework for video and image analysis. Developed by NVIDIA, it is designed for multisensor processing and streaming analytics. With a unified API between Python and C, DeepStream offers a flexible development environment for building efficient video analytics pipelines on NVIDIA GPUs. The framework features a plugin-based architecture for integrating various video and image analysis algorithms, including object detection, classification, segmentation, and optical flow.

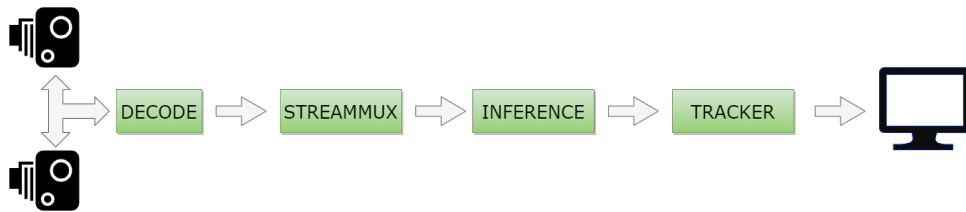


Figure 5.2: DeepStream SDK Architecture

The DeepStream SDK is rooted in the GStreamer multimedia framework and includes a GPU-accelerated plugin pipeline, with plug-ins for video inputs, decoding, preprocessing, TensorRT-based inference, object tracking, and display included to simplify the application development process. These capabilities allow for the creation of versatile multistream video analytics solutions. The *Decode* Block decodes data from a URI into unprocessed media, **Streammux** Block forms a batch of frames from multiple input sources while **Inference** and *Tracker* Block are executed by models with PyTorch or TensorRT frameworks.

To test our fastest model (YOLOv5n) with this framework, the on board camera was used to record another display which showed a video taken by a drone, while another locally downloaded video was sent in the streammux block

to create a second dummy camera and get a multi-stream architecture.



(a) Real-Time prediction at $t = 0$

(b) Real-Time prediction at $t = 5$

Figure 5.3: Real-Time Object Detection with DeepStream

Figure 5.3 shows the output of the two cameras, at the top the objects detected by the dummy camera (the locally downloaded video) while at the bottom the on board camera which detects objects on another display. We can see from the two frames how object detection is also robust to camera rotations, although it is analyzing a screen and not real objects. The inference rates of the *YOLOv5n* model trained on VisDrone are shown in Table 5.2.

Model	TensorRT	Top-Stream [fsp]	Bottom-Stream [fps]
YOLOv5n	-	24	23
YOLOv5n	FP32	30	28
YOLOv5n	FP16	31	30

Table 5.2: Real-Time Detection Results

Chapter 6

Conclusion

The aim of this work was to study the **Multi Object Detection** and **Multi Object Tracking** tasks with Deep Learning approaches, paying particular attention to the trade-off between accuracy and inference speed. In conclusion, it can be said that for the first part of the project (Multi Object Detection), we achieved a good accuracy of the **YOLOv5** models on the VisDrone-2019 Dataset, comparable to the first runners-up in the challenge, with a remarkable improvement on the speed of inference, they are able to triple the number of frames analyzed per second despite using a GPU with lower capabilities than the first runner-up. For the second part of the work (Multi Object Detection), the choice of this new architecture proved to be an excellent solution. The cascaded use of the **YOLOv5** for detection and the **strongSORT** algorithm for tracking showed excellent results. In the last VisDrone-2021 Challenge, the new version of **DPNet**, the first classified algorithm, achieved 39% accuracy (mAP), but we are still a long way from a possible application in reality.

6.1 Future works

From my point of view, greater attention should be directed towards resolving the issues of the VisDrone dataset rather than searching for and implementing new architectures in order to improve the training of existing algorithms. I thought of two future solutions, one for the MOT task and one for the MOD task:

1. **Balancing the VisDrone dataset**, is a very tough challenge, but doing an initial test run, in which i cut out all the boxes containing the class to be increased, and then applied transformations to increase the number of instances.

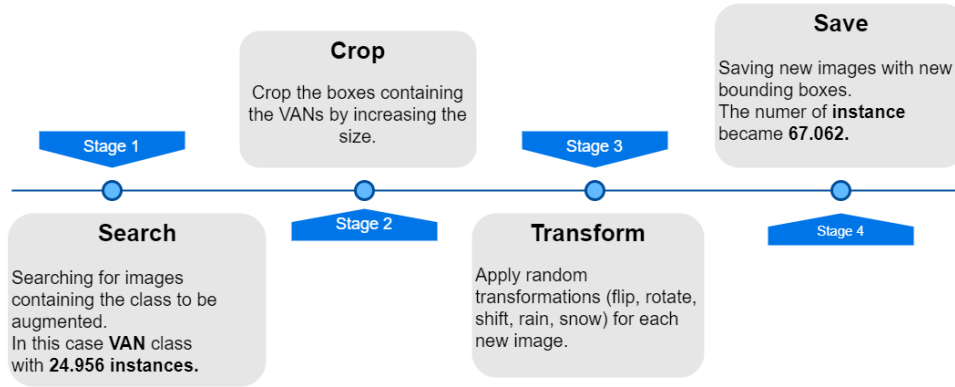


Figure 6.1: Workflow to increase instances of a class

Re-training the model (*YOLOv5n*) with the class '**van**' increased from about 25,000 instances to 67,000, we are able to increase the **AP** (Average Precision) of the respective class by almost 2% on validation; by repeating the same process for each class we could improve the accuracy of the network. Obviously the basic idea is not to create an image with only one instance cropped, but to overlay them on the other original images.

2. **Fine-Tuning OSNet**, the other future development is about the MOT task, in this case in the tests performed we obtain good results, but it can be seen in table 4.2, that the tracking metric (MOTA) does not increase even when using the most accurate YOLOv5 model for detection, this is because the Re-ID network (OSNet) used was pre-trained on *Market1501*, a dataset containing only people filmed from above. Therefore, a dataset would need to be created from VisDrone to retrain the network to extract features on the classes of interest to create more accurate tracks.

Chapter 7

Sommario

Il lavoro presentato è stato svolto durante un tirocinio curricolare presso l'azienda **MBDA Italia s.p.a**, leader in Europa per la progettazione e produzione di missili e sistemi missilistici. In particolare è stato svolto nella divisione di Missile Design Italia, nel reparto di guida, navigazione e controllo. Lo scopo della tesi era studiare soluzioni per il rilevamento e l'inseguimento di oggetti ripresi da droni con approcci di deep learning. Particolare attenzione è stata volta allo studio del rapporto tra accuratezza e velocità di inferenza dei modelli, al fine di poterli eseguire su schede embedded e testarne i benefici in Real Time. Il dataset scelto per questo lavoro è **VisDrone 2019**, un dataset pubblico con 10 categorie di oggetti (pedestrian, people, car, van, bus, truck, motor, bicycle, awning-tricycle, and tricycle). Esso rappresenta una sfida, ancora oggi del tutto aperta, in cui i maggiori ricercatori cercano di trovare modelli più adatti a rilevare e classificare piccoli oggetti ripresi da droni. Dai correlogrammi riportati sul dataset si possono intuire le sue due enormi difficoltà, la prima derivata dall'elevato sbilanciamento delle classi e la seconda derivata dalla dimensione degli oggetti all'interno delle immagini, in cui molto spesso sono meno dell'1% dell'immagine stessa. La rete scelta per essere addestrata su VisDrone, per studiarne la capacità di rilevamento di piccoli oggetti, è la **YOLOv5** (You Only Look Once), una rete a single-step progettata da *Ultralytics* [22] e pre-allenata sul dataset COCO, ampiamente sfruttata per la sua leggerezza e velocità di esecuzione. Nella **prima parte** del pro-

getto sono stati utilizzati i 5 modelli di YOLOv5 (Nano, Small, Medium, Large, X-Large) classificati in ordine di profondità e quindi del numero di layer utilizzati. E' intuibile dai risultati sulla validazione dei modelli utilizzando il dataset COCO, effettuata dai creatori delle reti, che all'aumentare del numero di layer la rete risulta più accurata a discapito della velocità di inferenza. Dopo avere preparato un container Docker contenente tutte le librerie necessarie all'implementazione e al testing, è stato eseguito l'allenamento di tutte e 5 le reti utilizzando gli stessi parametri, descritti nel capitolo 3. I modelli sono stati validati utilizzando le metriche necessarie per questo tipo di sfida (**mAP** [sez. 4.2]), come richiesto nel paper della challenge VisDrone 2019 [35]. Da una prima valutazione delle metriche potrebbe sembrare che i modelli non siano adatti a risolvere questo tipo di task, in realtà nel paper vengono menzionati i primi 33 algoritmi classificati in ordine di mAP, 29,62% per il primo. La rete utilizzata in questo lavoro raggiunge perfettamente i risultati (29,9%), triplicando il numero di frame analizzati al secondo, nonostante l'utilizzo di una **GPU NVIDIA Quadro P4000**, con prestazioni inferiori rispetto alla **Titan XP** utilizzata dai vincitori della challenge. Un aspetto importante abbiamo detto essere il tempo di inferenza, dunque è stata applicata una tecnica di quantizzazione dei tensori della rete per ottimizzare la velocità. I risultati ottenuti sono sorprendenti per la quantizzazione in FP32 e FP16, ottenuta attraverso il framework **TensorRT**, riuscendo a mantenere le metriche di valutazione pressochè identiche e quasi raddoppiando la velocità di esecuzione.

La **seconda parte** del progetto si concentra a risolvere il problema del tracciamento di più oggetti ripresi da droni. Per questo obiettivo si è pensati ad un approccio a cascata tra due algoritmi, il primo appena descritto (*YOLOv5*) che si occupa di effettuare il rilevamento degli oggetti, mentre il secondo (*strongSORT*) prende in input l'uscita della *YOLOv5* e crea delle tracce per ogni oggetto rilevato. **StrongSORT** è un algoritmo complesso, composto da una rete per la re-identificazione ID degli oggetti, in questo caso è stata utilizzata una **OSNet** (Omni-Scale Network) pre-addestrata sul dataset Market1501, che si occupa di estrapolare delle caratteristiche fondamentali da ogni box (come la dimensione,

la forma dell'oggetto, i colori, ecc.), poi passa l'informazione ad un **Filtro di Kalman** il quale stima la posizione futura degli n oggetti presenti nel frame e attraverso l'**Algoritmo Ungherese** viene creata una matrice di costo per associare ogni oggetto rilevato la rispettiva traccia, dopo avere aggiornato il filtro di Kalman, in uscita avremmo sia le coordinate del box predetto ,la classe e l'ID assegnato dall'algoritmo di *strongSORT*. Questo approccio è stato testato prima su un task di inseguimento di una singola persona ripresa da drone, e successivamente su sequenze video del dataset VisDrone Tracking. Nel primo caso i risultati sono sorprendenti, raggiungendo il 90% di precisione sia nel rilevamento che nell'inseguimento, questo è dovuto dal fatto che entrambi i modelli che formato il nostro algoritmo completo (*YOLOv5* e *OSNet*) sono stati ampiamente allenati sulla classe "persona". Successivamente nel secondo test mostrato [sez.4.1], su una sequenza video con 122 oggetti e tracce, sono state introdotte nuove metriche di valutazione, ma anche in questo caso i risultati sono buoni, arrivando a tracciare 98 tracce su 122 con una precisione nel rilevamento del 90% e una precisione nel tracciamento dell'85%. Ovviamente i tempi di inferenza aumentano dato che dobbiamo aggiungere alla somma dei tempi anche quello dell'algoritmo di *strongSORT*, notevolmente più lento rispetto al precedente, e più aumentano le tracce e il numero di oggetti più diminuisce la velocità di esecuzione. Anche in questo caso sono stati mostrati i risultati eseguendo una quantizzazione indipendente anche nel modello *OSNet*, riuscendo raddoppiare la velocità senza alterare le metriche.

La **parte finale** del progetto è incentrata sul deploy degli algoritmi di rilevamento sulla scheda **NVIDIA Jetson TX2**, al fine di confrontare i tempi di esecuzione e studiare il comportamento delle reti in Real-Time. Come previsto eseguendo i due modelli più semplici (*YOLOv5n* e *YOLOv5s*), la velocità diminuisce notevolmente anche spingendo la scheda alla massima potenza di calcolo a discapito dell'efficienza energetica. Successivamente è stata settata la telecamera disponibile sulla scheda testando la velocità in Real-Time del modello più semplice allenato sul dataset COCO, notando che lavora a circa 40fps. Purtroppo per l'impossibilità di eseguire dei test in Real-Time da drone è stato

utilizzato il framework **DeepStream** di NVIDIA che consente di analizzare più flussi di stream in Real-Time contemporaneamente utilizzando delle reti neurali. Dunque è stata utilizzata la telecamera a bordo scheda per riprendere un'altro monitor, il quale mostrava una video ripreso da drone, e contemporaneamente è stata creata una seconda telecamera fittizia attraverso un video già registrato. I risultati sono molto buoni e per entrambi i video si riesce a raggiungere una velocità di 30 fps in contemporanea. Possiamo concludere affermando che seppur i risultati ottenuti siano buoni da un punto di vista teorico, c'è ancora molto da lavorare per questo tipo di sfida. Nell'ultima challenge di VisDrone 2021 il primo algoritmo classificato ha raggiunto il 39% di accuratezza, ma siamo ancora molti lontani da una possibile applicazione nella realtà. Secondo me i futuri sviluppi che andrebbero portati avanti sono due, uno per ogni challenge. Dal mio punto di vista bisognerebbe concentrarsi più nel risolvere le problematiche del dataset VisDrone piuttosto che cercare e implementare nuove architetture, al fine di migliorare l'allenamento degli algoritmi già esistenti. Bilanciare tutte le classi è una sfida molto dura, ma facendo un primo test di prova, in cui ho ritagliato tutte i box che contenevano la classe da aumentare, e successivamente applicato della trasformazioni per aumentarne il numero di istanze. Riaddestrando il modello con la classe "van" aumentata da circa 25.000 istanze a 65.000, riusciamo ad aumentare l'Average Precision della rispettiva classe di quasi il 2%; ripetendo lo stesso procedimento per ogni classe potremmo migliorare l'accuratezza della rete. Ovviamente l'idea di fondo non è quella di creare un' immagine con una sola istanza ritagliata, ma saprebbe quella di andare a sovrapporre sulle altre immagini originali. L'altro sviluppo per migliorare l'inseguimento di oggetti è quello di preparare un dataset VisDrone per riaddestrare il modello OSNet, attualmente specializzato nel rilevamento di persone.

List of Figures

1.1	Computer Vision Challenges [5]	10
2.1	Basic CNN Architecture [6]	14
2.2	Object Detection [7]	16
2.3	A road map of object detection [8]	17
2.4	Samples of tracking	19
2.5	MOT Functional Architecture by Xie et al. [11]	21
2.6	Docker Architecture [18]	23
2.7	Deep Learning Software NVIDIA [19]	27
3.1	Comparison between the object detection models	29
3.2	Types of YOLOv5 models [22]	30
3.3	Architecture YOLOv5 (Kim et al., 2021 [28])	31
3.4	Intersection Over Union	34
3.5	Flowchart to evaluate the boxes predicted	38
3.6	Average Precision	39
3.7	Instances of VisDrone	41
3.8	Labels Correlogram VisDrone	42
3.9	Transfer Learning	44
3.10	Precision and Recall as the confidence changes	45
3.11	Validation YOLOv5n	45
3.12	Prediction on the test frame	48
3.13	Diagram for quantization of models	50

LIST OF FIGURES

4.1	Multi Object Tracking Architecture	52
4.2	OSNet Architecture by Zhou et al. [39]	54
4.3	Single Object Tracking	59
4.4	Multi Object Tracking	60
5.1	Secure Shell Protocol	63
5.2	DeepStream SDK Architecture	66
5.3	Real-Time Object Detection with DeepStream	67
6.1	Workflow to increase instances of a class	69

List of Tables

3.1	Challenge Results VisDrone 2019	42
3.2	Validation Results on COCO	44
3.3	Validation Results on VisDrone	46
3.4	Test Results on VisDrone	47
3.5	Quantization Results (MOD)	50
4.1	Single Object Tracking Results	60
4.2	Multi Object Tracking Results	61
4.3	Quantization Results (MOT)	62
5.1	Test Results on Jetson TX2	65
5.2	Real-Time Detection Results	67

Bibliography

- [1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
- [2] Òscar Lorente, Ian Riera, and Aditya Rana. Image classification with classic and deep learning techniques, 2021.
- [3] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [4] Thomio Watanabe and Denis Wolf. Instance segmentation as image segmentation annotation, 2019.
- [5] Computer vision. <https://desupervised.io/computer-vision>.
- [6] MK Gurucharan. Cnn architecture. <https://www.upgrad.com/blog/basic-cnn-architecture>.
- [7] How to detect people using computer vision. <https://learn.alwaysai.co/object-detection>.
- [8] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey, 2019.
- [9] Ch Murthy, Mohammad Farukh Hashmi, Neeraj Bokde, and Zong Woo Geem. Investigations of object detection in images/videos using various

- deep learning techniques and embedded platforms—a comprehensive review. *Applied Sciences*, 05 2020.
- [10] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *NIPS*, pages 91–99, 2015.
- [11] Daniel Izadia Sean Bangera Thayne Walkera Ryan Ceresania Christopher Guaglianoa Henry Diaza Wanlin Xiea, Jaime Idea and Jason Twedta. Multi-object tracking with deep learning ensemble for unmanned aerial system applications. 10 2021.
- [12] Julie Dequaire, Peter Ondrůška, Dushyant Rao, Dominic Wang, and Ingmar Posner. Deep tracking in the wild: End-to-end tracking using recurrent neural networks. *The International Journal of Robotics Research*, 37(4-5):492–512, 2018.
- [13] Ricardo Pereira, Guilherme Carvalho, Luís Garrote, and Urbano J. Nunes. Sort and deep-sort based multi-object tracking for mobile robotics: Evaluation with new data association metrics. *Applied Sciences*, 12(3):1319, Jan 2022.
- [14] Matt Rabinovitch. Comparing state of the art region of interest trackers. <https://medium.com/teleidoscope/comparing-state-of-the-art-region-of-interest-trackers>, 2019.
- [15] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. 04 2016.
- [16] Nvidia quadro p4000 documentation. <https://www.nvidia.com/>.
- [17] Ubuntu 20.04 lts documentation. <https://releases.ubuntu.com/focal/>.
- [18] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

- [19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [20] Venkatesh Wadawadagi. Tensorflow vs pytorch: Deep learning frameworks. 01 2023.
- [21] Docker-hub yolov5 image. <https://hub.docker.com/r/ultralytics/yolov5/dockerfile>.
- [22] Yolov5 github repository. <https://github.com/ultralytics/yolov5>.
- [23] strongsort github repository. https://github.com/mikel-brostrom/Yolov5_StrongSORT_OSNetforStrongOSNettracking.
- [24] Evaluation mot github repository. https://github.com/shenh10/mot_evaluationforevaluatetrackingmodel.
- [25] Model quantization github repository. https://github.com/Wulingtian/yolov5_tensorrt_int8_toolstensorrtquantizationint8.
- [26] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.
- [27] Amitabha Banerjee. Yolov5 vs yolov6 vs yolov7. <https://www.learnwitharobot.com/p/yolov5-vs-yolov6-vs-yolov7>, 2022.
- [28] Munhyeong Kim, Jongmin Jeong, and Sungho Kim. Ecap-yolo: Efficient channel attention pyramid yolo for small object detection in aerial image. *Remote Sensing*, 13:4851, 11 2021.
- [29] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

- [30] Laurens van der Maaten Kilian Q. Weinberger Gao Huang, Zhuang Liu. Densely connected convolutional networks. 10 2020. <https://arxiv.org/pdf/1608.06993.pdf>.
- [31] Ge H Zhang Z Zang X. Xu Q, Zhu Z. Effective face detector based on yolov5 and super-resolution reconstruction. *Comput Math Methods Med*, 2021.
- [32] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, (1):100–108.
- [33] Apurva Badithela, Tichakorn Wongpiromsarn, and Richard M. Murray. Evaluation metrics for object detection for autonomous systems, 2022.
- [34] Dataset visdrone 2019 download. <https://github.com/VisDrone/VisDrone-Dataset>.
- [35] Dawei Du and et. al Zhu. Visdrone-det2019: The vision meets drone object detection in image challenge results. pages 213–226, 2019.
- [36] Quan Zhou, Huimin Shi, Weikang Xiang, Bin Kang, Xiaofu Wu, and Longin Jan Latecki. Dpnet: Dual-path network for real-time object detection with lightweight attention, 2022.
- [37] Keshav Ganapathy. A study of genetic algorithms for hyperparameter optimization of neural networks in machine translation, 2020.
- [38] Abhay Chaturvedi. Understanding nvidia tensorrt for deep learning model optimization. 11 2020.
- [39] Andrea Cavallaro Tao Xiang Kaiyang Zhou, Yongxin Yang. Omni-scale feature learning for person re-identification. 12 2019.
- [40] et al. Jesmin Jahan Tithi, Sriram Aananthakrishnan. Online and real-time object tracking algorithm with extremely small matrices. 03 2021.
- [41] Renu Khandelwal. Hungarian algorithm. 03 2022.

BIBLIOGRAPHY

- [42] S. V. Aruna Kumar, Ehsan Yaghoubi, Abhijit Das, B. S. Harish, and Hugo Proença. The p-destre: A fully annotated dataset for pedestrian detection, tracking, and short/long-term re-identification from aerial devices. *IEEE Transactions on Information Forensics and Security*, 16:1696–1708, 2021.
- [43] Nvidia sdk manager. <https://developer.nvidia.com/drive/sdk-manager>.
- [44] Data sheet nvidia jetson tx2. <https://download.kamami.pl>.
- [45] Robertson SJ. *Robertson SJ. Disarthria Profile*. Winslow Press, 1982. Versione italiana a cura di Fossi F. e Cantagallo A. Ediz. Omega(1999).
- [46] Liyao Wu Wenying Chen Lu Tan, Tianran Huangfu. Comparison of retinanet, ssd, and yolo v3 for real-time pill identification. 10 2020.
- [47] Metrics - average precision and map. <https://innerpeace-wu.github.io/2018/03/22/Metrics-Average-Precision/>.