SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Multi-Vehicle Detection and Tracking in Aerial Image Sequences based on Deep Learning

Somesh Khandelia

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Multi-Vehicle Detection and Tracking in Aerial Image Sequences based on Deep Learning

Multi-Vehicle Erkennung und Verfolgung in Luftbild-Sequenzen basierend auf Deep Learning

Author:	Somesh Khandelia
Supervisor:	Prof. Dr.-Ing. habil. Alois Christian Knoll
Advisor:	Walter Zimmer, M.Sc.
Submission Date:	April 17, 2023

Disclaimer

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, April 17, 2023 Somesh Khandelia

Acknowledgement

On the professional front, I am indebted to Mr. Walter Zimmer of the TUM AIR Chair, for supervising my thesis and providing valuable guidance from the first day. He always motivated me to strive for better. I am grateful to Mr. Franz Kurz, Mr. Maximilian Langheinrich, Mr. Seyed Majid Azimi and Mr. Reza Bahmanyar of DLR (the German Aerospace Center) for giving me the opportunity and resources to work on a very interesting thesis. On the personal front, words alone cannot express my gratitude towards my partner in the TUM master's journey, Ms. Ashika Manjunath. I am also grateful to my parents Mr. Anil Khandelia and Mrs. Anita Khandelia, my little sister Ms. Yukty Khandelia and my grandmother Mrs. Lakshmi Dhanuka for keeping me in their prayers and being my pillars of support. On the spiritual front, none of this would have been possible, but for my God, Lord Shiva, from whom I derive my very name Somesh. In times of anxiety and stress, I chanted the most powerful mantra of meditation that ever existed: *Om Namah Shivaya*!

Abstract

Multi-object detection is a classical challenge in the computer vision community which involves the identification of up to several objects of interest in an image and constructing bounding boxes around them to demarcate them from the background. Multi-object tracking (MOT) takes this a step further by constructing the course followed by each of the detected objects in every image frame of a video sequence. The research community works actively on the task of MOT via the MOT benchmark that offers the most popular datasets which are generally centered around pedestrian detection and tracking on the ground. However, in this work, we tackle the relatively less worked-upon problem of detecting and tracking vehicles in aerial imagery and 11 classes of objects on the road in ground image sequences. We pick certain state-of-the-art (SOTA) algorithms from the MOT benchmark and apply them to our domain which consists of two sharply contrasting datasets, the low FPS high resolution DLR dataset containing aerial images captured from a helicopter with large camera motion and the high FPS medium resolution A9 dataset containing ground images captured from traffic monitoring systems with no camera motion. We train several YOLOv7 based detection models and test several SOTA tracking algorithms on the two datasets to conclude that intersection-over-union (IoU) and Kalman Filter work well on the A9 dataset but not on the DLR dataset, whereas appearance features and camera motion compensation make more sense for the DLR dataset and not so much for the A9 dataset. We therefore propose a new robust tracking algorithm called Byte-De-SORT that lacks the Kalman Filter and is a combination of the IoU based ByteTrack and the appearance features based DeepSORT. Byte-De-SORT achieves a competitive (HOTA, MOTA, IDF1) score of (0.56, 0.75, 0.63) on the DLR dataset and (0.48, 0.68, 0.51) on the A9 dataset, making it the best overall method that can be applied to both the datasets. It achieves an average inference speed of 2.82 FPS on the DLR dataset and 14.34 FPS on the A9 dataset, making it also suitable for real-time tracking.

Kurzfassung

Die Erkennung mehrerer Objekte ist eine klassische Herausforderung in der Computer-basierten Bilderkennung. Diese beinhaltet die Identifizierung von mehreren Objekten in einem Bild mit einer 2D Box, um sie vom Hintergrund abzugrenzen. Das Verfolgen von mehreren Objekten (*multi-object tracking*, kurz MOT) geht noch einen Schritt weiter, indem es den Verlauf von jedem erkannten Objekt in einer Videosequenz konstruiert. Die Forschungsgemeinschaft arbeitet aktiv an der MOT-Aufgabe. Hierzu gibt es den MOT-Benchmark, der die beliebtesten Datensätze im Bereich der Erkennung und Verfolgung von Fußgängern bietet. In dieser Arbeit fokussieren wir uns jedoch auf das relativ weniger bearbeitete Problem, der Erkennung und der Verfolgung von Fahrzeugen in Luftbildaufnahmen, sowie in Bildern aufgenommen aus Infrastrukturperspektive. Wir wählen neuartige Algorithmen aus dem MOT-Benchmark aus und wenden sie in unseren zwei Domänen an, die aus zwei Datensätzen bestehen. Der DLR Datensatz wurde aus einem Hubschrauber aus aufgenommen und beinhaltet hochaufgelöste Luftbilder, die mit einer niedrigen Bildwiederholungsrate aufgenommen wurden. Bei der Aufnahme hat sich die Kamera mitbewegt, was eine große Herausforderung an die Verfolgung der Fahrzeuge darstellt. Der zweite Datensatz (A9 Datensatz) hat eine höhere Bildwiederholungsrate, jedoch eine geringere Bildauflösung. Dieser Datensatz enthält Aufnahmen von statischen Verkehrsüberwachungskameras. Wir trainieren mehrere YOLOv7-basierte Erkennungsmodelle und testen mehrere Algorithmen zur Verfolgung von Fahrzeugen auf beiden Datensätzen. Letztenendes kommen wir zu dem Schluss, dass der Schnittpunkt über die Vereinigung (*Intersection-over-Union*, kurz IoU) und der Kalman-Filter gut auf dem A9 Datensatz funktionieren, jedoch weniger gut auf dem DLR Datensatz. Die Verwendung von Erscheinungsmerkmalen sowie die Kompensation der Kamerabewegung macht auf dem DLR Datensatz mehr Sinn als auf dem A9 Datensatz. Wir schlagen daher einen neuen robusten Algorithmus zur Verfolgung von Fahrzeugen mit geringen Bildwiederholungsraten vor und nennen diesen Byte-De-SORT. Aufgrund der geringen Bildwiederholungsrate kommt der Kalman-Filter hier nicht zum Einsatz. Byte-De-SORT ist eine Kombination aus dem IoU-basierten *ByteTrack* und dem auf Erscheinungsmerkmalen basierenden *DeepSORT*. Byte-De-SORT erreicht ein wettbewerbsfähiges Ergebnis (HOTA=0,56, MOTA=0,75, IDF1=0,63) auf dem DLR Datensatz sowie ein etwas niedrigeres Ergebnis (HOTA=0,48, MOTA=0,68, IDF1=0,51) auf dem A9 Datensatz, was ihn zur besten Methode macht, die auf beiden Datensätzen angewandt werden kann. Erreicht wird eine durchschnittliche Inferenzgeschwindigkeit von 2,82 FPS auf dem DLR Datensatz und 14,34 FPS auf dem A9 Datensatz, sodass er auch für eine eine Verfolgung von Fahrzeugen in Echtzeit geeignet ist.

Contents

Disclaimer **iii**

Acknowledgement **v**

Abstract **vii**

Kurzfassung **ix**

1 Introduction **1**
1.1 Problem Statement 1
1.2 Motivation 1
1.3 Contribution 2
1.4 Structural Outline 3

2 Essential Background **5**
2.1 Goals 5
2.2 Object Classification 5
2.2.1 What is Object Classification? 5
2.2.2 How is it done? 5
2.2.3 Metrics of evaluation 8
2.3 Object Detection 9
2.3.1 What is Object Detection? 9
2.3.2 How is it done? 9
2.3.3 Metrics of evaluation 12
2.4 Object Tracking 14
2.4.1 What is Object Tracking? 14
2.4.2 How is it done? 15
2.4.3 Metrics of evaluation 18

3 Related Work **25**
3.1 Goals 25
3.2 Previous Theses 25
3.2.1 Beheim et al., 2021 25
3.2.2 Kraus et al., 2020 27
3.3 Methodology based 29
3.3.1 MPNTrack 29
3.3.2 SiamMOT 31

3.3.3 TrackFormer 33
3.4 State of the Art 35
3.4.1 ByteTrack 35
3.4.2 UAVMOT 37
3.4.3 BoT-SORT 40

4 Working on the DLR dataset **43**
4.1 Dataset overview 43
4.2 Detection experiments 45
4.2.1 640 X 640 46
4.2.2 1280 X 1280 47
4.2.3 2560 X 2560 48
4.2.4 3200 X 3200 49
4.3 Tracking experiments 50
4.4 Conclusion 55

5 Working on the A9 dataset **57**
5.1 Dataset overview 57
5.2 Detection experiments 67
5.3 Tracking experiments 69
5.4 Conclusion 74

6 Conclusion **75**

7 Future Work **77**

List of Figures **79**

List of Tables **83**

Bibliography **85**

1 Introduction

1.1 Problem Statement

Most of the research done today, in the domain of Multiple Object Tracking (MOT), is focused around pedestrian tracking, at the ground level. Most algorithms developed in recent times attempt to climb the leaderboard on the popular MOT benchmarks like MOT15 [1], MOT16 [2], MOT17 [3], and MOT20 [4]. Consequently, state-of-the-art (SOTA) architectures end up being developed for pedestrian tracking, whereas the research community is relatively less active in developing algorithms for other domains. However, other domains do have their own relevance, importance and applications. Therefore, it would make sense to leverage the SOTA architectures and algorithms, developed for the domain of pedestrian tracking, in these other domains in order to establish good baselines. Establishing baselines gives us a quantitative insight into the difference between different domains. For example, from a qualitative point of view, the DLR dataset (see Chapter 4) has extremely small objects (GSD: 3-6 cm, image resolution: 5472 X 3648) captured from a helicopter, in comparison to the conventional MOT datasets which generally consist of image sequences on the ground level. One can easily argue that tracking multiple objects simultaneously is going to be relatively difficult on this dataset, but one of the goals of this thesis is to explain the reasoning behind that argument, quantitatively, rather than leaving it to intuition. Once we become objectively aware of the challenges posed by these different domains, the next step is to build upon the SOTA algorithms further, to overcome their shortcomings (e.g. we found IoU based algorithms do not work well with low FPS data) while being applied to these domains. Naturally, the ideal goal would be to obtain an approach that is one-size-fits-all, and therefore works well for multiple domains. To summarise, the goal of this thesis is to **further develop upon a tracking method, that used to originally work well on popular MOT benchmarks, such that it can now very well be applied to track both: "Vehicles in aerial imagery" (the DLR dataset, see chapter 4) as well as "Multiple (11) classes of objects, in traffic monitoring systems, on the ground level" (the A9 dataset, see Chapter 5).**

1.2 Motivation

Vehicle tracking has several applications including surveillance systems, traffic monitoring, autonomous driving, and accident prevention. The Providentia project [5] and consequently the A9 dataset [6] (see Chapter 5) are aimed at creating a digital twin of the traffic based on the data collected from the A9 highway in Garching (a little north of Munich, Germany). This digital twin can be streamed to autonomous vehicles on the network so that they are better aware of the road and traffic conditions, and thus accidents can be prevented much better. In the near future, it is planned that the 3D perception pipeline of the Providentia project will use a detector based on YOLOv7 [7] pretrained on MS COCO [8] and a tracker based on the SORT [9] algorithm. At the time of writing this thesis, SORT was published more than 5 years back and therefore more SOTA trackers should be tested in order to obtain potentially better alternatives to SORT. The DLR [10] dataset (see Chapter 4), which is yet to be published, consists of very high resolution aerial image sequences of vehicles in different scenarios like crowded city streets and highways, and from a research perspective it is desirable to see how effective can this dataset be, to serve as a benchmark for vehicle tracking in aerial imagery. Since two very contrasting datasets presented themselves as candidates for vehicle tracking, it was natural through intellectual curiosity to attempt to find a one-size-fits-all tracking method that can be, in general, universally applied.

1.3 Contribution

The **primary** contribution of this thesis is a tracking algorithm called **Byte-De-SORT**. We derive the name of this algorithm from two existing methods on which it is based, namely ByteTrack [11] and DeepSORT [12]. Byte-De-SORT is largely based on the SOTA tracking method ByteTrack [11], which has recently been used as a basis for several tracking methods developed across domains, because of its simple yet promising ideology of not throwing away low-score detections (confidence score of these detections is a hyperparameter and can typically be between 0.15 and 0.5) while doing tracking. However, the primary mode of data association in ByteTrack was IoU (see Section 2.4.2), which has been shown in this thesis to not work well for low FPS datasets with large camera motion. Therefore, it had to be removed. Similarly, Kalman Filter (see Section 2.4.2), which is a popular choice for motion modelling using a constant velocity assumption, did not prove to be effective for datasets with low FPS and large camera motion. Therefore, we removed the Kalman Filter based motion prediction from ByteTrack as well. Camera motion compensation proved effective while working with the DLR dataset, when we tried out the BoT-SORT [13] algorithm that uses OpenCV [14] based image registration techniques to correct the Kalman Filter predictions. However, the technique was too slow at inferencing (0.01 FPS) to be part of a real-time tracker and hence we discarded the idea. For the purpose of data-association, inspired from DeepSORT, we decided to use cosine distance between feature embeddings of detections in the previous and current image frame. These feature embeddings are basically 512 dimensional vectors, obtained by using a ResNet-50 backbone trained on the task of re-ID (see Section 2.4.2). Therefore, we obtained **an online tracking algorithm for real-time usage, based on the tracking-by-detection paradigm that can be directly used off-the-shelf without training, because of the absence of a motion model, by**

plugging in a pretrained object detector and a re-ID model. Byte-De-SORT works well on both a low FPS high resolution dataset containing very small objects with large camera motion (DLR dataset, see Chapter 4) and a high FPS medium resolution dataset containing large objects with no camera motion (A9 dataset, see Chapter 5).

The **secondary** contribution of this thesis work is the comprehensive study and comparison of several existing SOTA tracking algorithms and their application to two very contrasting datasets. We trained and tested several YOLOv7 based detectors on the two datasets, since in tracking-by-detection paradigm, the performance of the detector hugely influences the performance of the tracker. We trained with different image sizes (640 X 640, 1280 X 1280, 2560 X 2560, 3200 X 3200), and discovered that resolution has an impact on inference accuracy. We tested several tracking algorithms on the two datasets in order to ascertain which characteristics of tracking algorithms worked best with different properties of datasets. The tracking algorithms were pretrained on VisDrone2020 [15] dataset, which has medium-sized vehicles captured from a drone, since the DLR dataset has very small vehicles captured from a helicopter, while the A9 dataset has large vehicles captured from cameras hosted on traffic monitoring systems.

1.4 Structural Outline

In Chapter 1, we introduce the topic of the thesis, the necessity and motivation behind conducting research on this topic and our contributions to the research community as part of this thesis. Chapter 2 provides some conceptual background of multi-object tracking to seamlessly walk through this thesis work. We start with the most basic task of object classification and proceed to the more challenging tasks of detection and tracking. We also explain the most popular metrics of evaluation for these tasks. Chapter 3 discusses certain noteworthy research works on multi-object tracking. We look at two previous theses works which are immediate predecessors of this thesis, a few methods that tackle the problem of MOT with extremely different architectures and finally some relatively recent SOTA algorithms. In Chapter 4, we briefly take a look at the DLR dataset, which is yet to be published, that contains aerial image sequences of vehicles. We discuss the data preprocessing that was essential to bring the data in a format suitable to be fed in to the detectors and trackers. Finally, we perform detection and tracking on the dataset and report the results, while gathering some insights behind the observations. In Chapter 5, we present a brief overview of the A9 dataset, most sequences of which have not yet been published. This dataset contains image sequences of mostly different kinds of vehicles on traffic highways and intersections. We talk about the data preprocessing, detection and tracking experiments that were performed on the A9 dataset, and we report the results and the insights behind those results. In Chapter 6, we present a summary of the knowledge gained through several experiments that were performed as part of this thesis. In Chapter 7, we present some ideas that can be built on top of the results of this thesis, in order to hopefully obtain even better outcomes in future.

2 Essential Background

2.1 Goals

The goal of this chapter is to acquaint the reader with the concepts that will be discussed throughout this work. To progress seamlessly through this work, a good understanding of the concepts presented in this chapter is essential.

2.2 Object Classification

2.2.1 What is Object Classification?

Given an image and a set of known classes or object-types, the goal is to identify which class the image belongs to or basically to find what is the type of the object present in the image. For example, if you have a binary classifier that classifies images into two classes e.g. cars or trucks, you can pass an image to this classifier and ask what is the probability that the image is of a car (or truck). An example for further clarification is given in Figure 2.1.

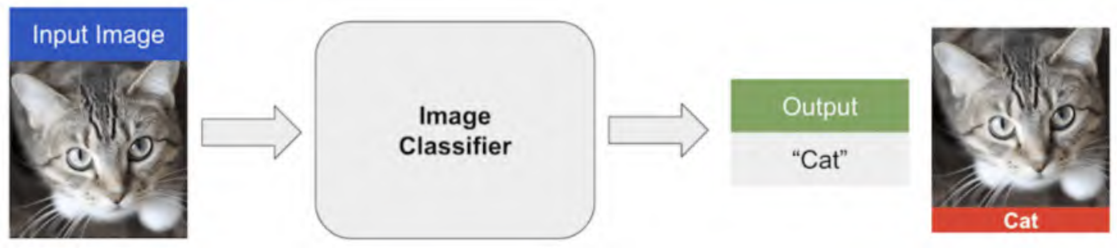


Figure 2.1: Example of Object Classification (image source [16])

2.2.2 How is it done?

With the advent of Deep Learning and Neural Networks, in the last decade, most of the work revolving around object classification has been done with the help of Convolutional Neural Networks (CNN). CNN based architectures generally work in two basic steps:

1. Feature extraction: For the purpose of feature extraction, a filter/kernel of a given size e.g. 5 X 5 is convolved with the image which is of a given resolution e.g. 1080 X 1080. Previously, manually defined kernels would be used to extract particular

features e.g. edges in images. But in CNNs, these kernels are learnt (the 25 weights in this example) with the help of back-propagation.

2. Feature selection: The extracted feature maps are passed through a Pooling operation, which is basically an aggregation. Popular Pooling techniques include Max Pooling and Average (Mean) Pooling. The goal is to obtain the overall/strongest feature signal.

The output of the steps above is called a feature embedding. This embedding is flattened (brought into 1 channel, e.g. the image was in 3 channels originally namely RGB, and then the number of channels kept changing depending on the number of convolutional filters in each layer) and passed through Linear (Dense) Neural Network layers, the output of which in turn can be passed through a Softmax layer to get the probabilities of the object, in the image, to belong to each of the given classes.

AlexNet

AlexNet [17] was one of the first models to produce excellent results on the ImageNet [18] challenge, whereby the goal is to correctly classify millions of images across thousands of classes. AlexNet showed the potential of CNNs for the task of object classification, and was followed by several models that were based on the same paradigm.

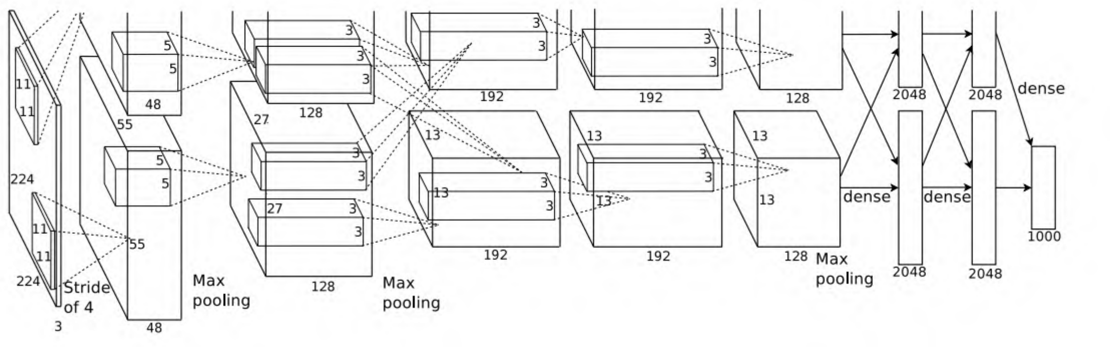
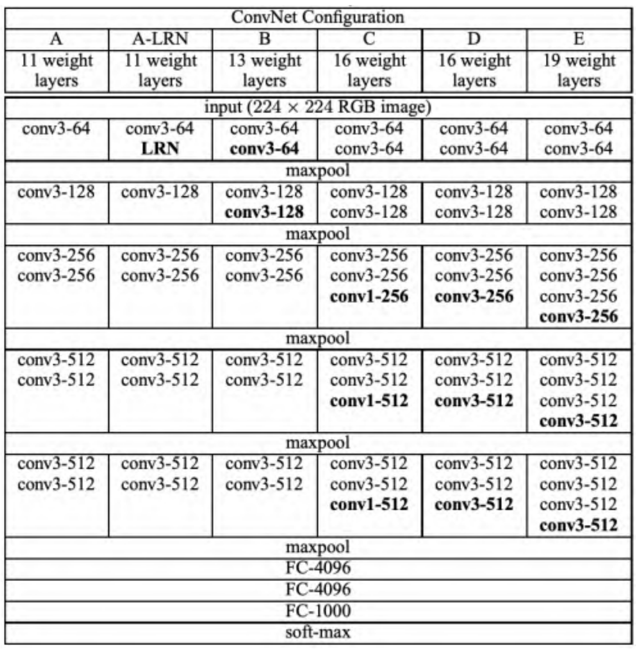


Figure 2.2: AlexNet architecture (image source [17])

VGGNet

It can be seen from Figure 2.2 that AlexNet [17] used convolutional filters of various sizes, 11 X 11, 5 X 5, 3 X 3 etc. VGGNet [19] simplified this architecture and used only 3 X 3 filters and obtained even better results on the ImageNet [18] challenge. It can be seen in Figure 2.3 that the authors of VGGNet [19] also experimented with the number of layers across several models, because that directly influences the number of learnable parameters and hence the capacity to learn.

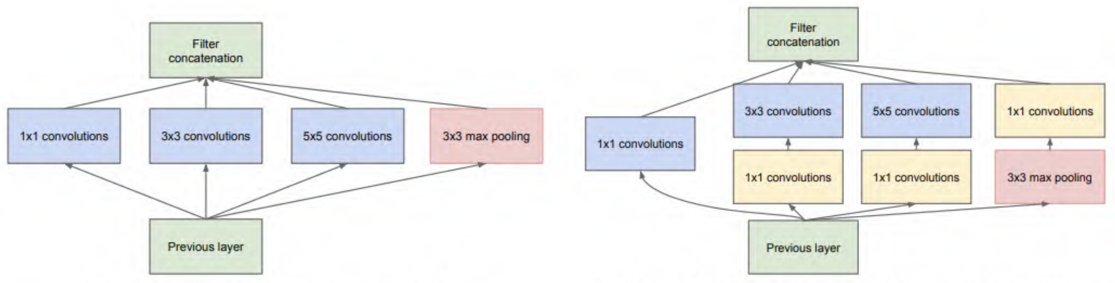


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.3: VGGNet architecture (image source [19])

GoogLeNet

With increasing number of learnable parameters (weights), training the model started to become a computationally intensive task. VGGNet [19] already had more than a hundred million trainable parameters. GoogLeNet (Inception) [20] introduced 1 X 1 convolutions to reduce these number of computations. It can be seen in Figure 2.4, 1 X 1 convolutions are performed to first reduce the number of channels (dimensions) in the feature maps, before applying the more expensive 3 X 3 or 5 X 5 convolutions on them. This saves many computations and makes the training efficient.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2.4: Inception module architecture (image source [20])

ResNet

It is logical to think that deeper neural networks will produce better accuracy, than the not so deep ones, because they have more trainable parameters and therefore a higher capacity to learn. However, it was found that merely increasing the number of layers in a network was saturating the accuracy or in some cases having even a negative impact on the training loss, and the cause was not overfitting. In order to train deeper networks, ResNet [21] introduced the concept of Residual blocks, as can be seen in Figure 2.5. A residual connection enables the network to decide whether it wants to keep the feature maps produced by a layer or skip the layer. Therefore, very deep architectures can be constructed and the network itself learns which layers are useful and which are not. Deeper architectures can now enjoy accuracy gains, as we had assumed at first, thanks to residual connections. ResNet [21] backbones are very popular today to obtain feature embeddings, in order to work with appearance based features.

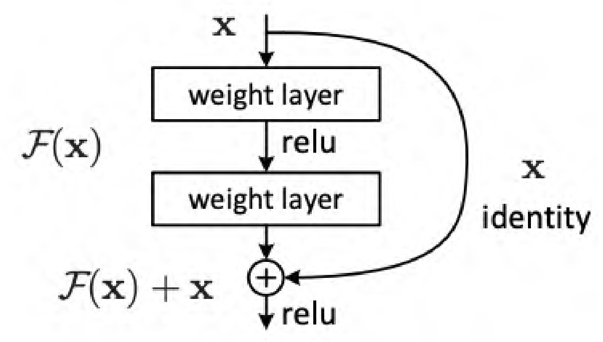

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.5: Residual block architecture (image source [21])

2.2.3 Metrics of evaluation

In order to compare the performance of different models, quantitative metrics are essential. For the purpose of object classification, a widely used metric is called F_1 score. Let us understand the calculation of F_1 score with an example.

- Suppose we have 4 images of cars and 6 images of trucks, and we have a classifier whose job is to identify cars i.e. it outputs a probability >= 0.5 if it thinks that the image is that of a car.
- Assume that for 3 actual car images and 2 actual truck images, the classifier outputs the probability >= 0.5 of being a car. These are called *Positives (P)*.
- Therefore, $(4+6) - (3+2) = 5$ images received a probability of < 0.5 of being a car by the classifier. These are called *Negatives (N)*.
- Out of 5 *Positives*, 3 are actually cars and are called *True Positives (TP)*, while 2 are actually trucks and not cars and are called *False Positives (FP)*.

- Out of 5 *Negatives*, 4 are actually trucks and are correctly not classified as cars, and are therefore called *True Negatives (TN)*, while 1 is actually a car but not classified as one and therefore falls under the category of *False Negatives (FN)*.
- *Precision* is a measure of the quality of the classification. Our classifier is able to classify positives, but how good is it at doing so and whether it misclassified a negative as a positive (*FP*) is answered by *Precision*.

$$Precision = \frac{TP}{TP + FP} = \frac{3}{3+2} = \frac{3}{5} = 0.6 \tag{2.1}$$

- *Recall* is a measure of the quantity of the classification. Our classifier is able to classify positives, but how many was it able to correctly classify and whether it missed out on any positive by classifying it as a negative (*FN*) is answered by *Recall*.

$$Recall = \frac{TP}{TP + FN} = \frac{3}{3+1} = \frac{3}{4} = 0.75 \tag{2.2}$$

- *Precision* or *Recall* alone are not enough to judge the performance of a classifier, and for a classifier to be good, both these measures should evaluate to as high as possible (≈ 1). Therefore, a unified metric called F_1 score, which is the harmonic mean of *Precision* and *Recall*, is generally used. Higher the F_1 score (≈ 1), better is the performance of the classifier.

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * 0.6 * 0.75}{0.6 + 0.75} = \frac{0.9}{1.35} = 0.67 \tag{2.3}$$

2.3 Object Detection

2.3.1 What is Object Detection?

Object Detection is a more generic form of Object Classification. Basically, it is Object Classification + Localization, where Localization means to pinpoint the location of an object in an image. Generally, Localization is done with the help of a bounding box (*bbox*). This can be easily clarified from Figure 2.6. We can also see an example of Instance Segmentation in Figure 2.6, which is to take things a step further, whereby each pixel of an image individually undergoes Object Classification.

2.3.2 How is it done?

An Object Detection model needs to not only give the probability of an object to belong to a class, but also give the coordinates of the bounding box (*bbox*). Therefore, the model needs to have two heads: a classification head and a regression head. A *bbox* can generally be represented by a quadruple as given in Equation 2.4 or Equation 2.5.

$$bbox = (x_{\text{min}}, y_{\text{min}}, width_{\text{bbox}}, height_{\text{bbox}}) \tag{2.4}$$

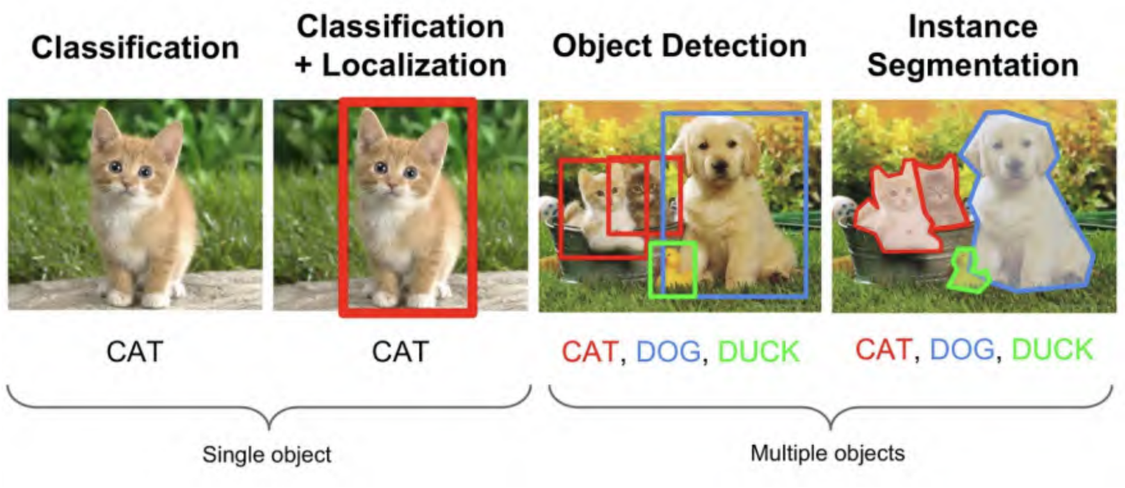


Figure 2.6: Example of Object Detection (image source [22])

$$bbox = (x_{\text{center}}, y_{\text{center}}, width_{\text{bbox}}, height_{\text{bbox}}) \tag{2.5}$$

(x_{min}, y_{min}) in Equation 2.4 represent the top-left corner coordinates of the *bbox*. (x_{center}, y_{center}) in Equation 2.5 represent the center coordinates of the *bbox*. In these equations, $width_{\text{bbox}}$ is the width of the *bbox* and $height_{\text{bbox}}$ is the height of the *bbox*.

There are, broadly speaking, two types of object detectors. We can now take a look at them, taking a very prominent example for each one of them.

Two-stage detectors

The R-CNN family of detectors, namely, R-CNN [23], Fast R-CNN [24] and Faster R-CNN [25] are very good examples of two-stage detectors. We can take the example of R-CNN for the purpose of conceptual understanding, since the others are performance improvements built on top of it. R-CNN works in three steps, as illustrated in Figure 2.7:

1. Use Selective Search [26] to give region-proposals. The 'R' in R-CNN stands for region-proposals, whereby a region-proposal is a section of the image where an object can probably be. Around 2000 region-proposals of different shapes and sizes are made.

2. A CNN is run on top of each region-proposal, to obtain a feature embedding.

3. The obtained feature embedding is passed in to a classification head (SVM, Support Vector Machine) to identify the class of the object in the region-proposal, and a regression head (Linear Regression) to obtain the *bbox* offsets (the difference in size and shape of the *bbox* from the original region-proposal).

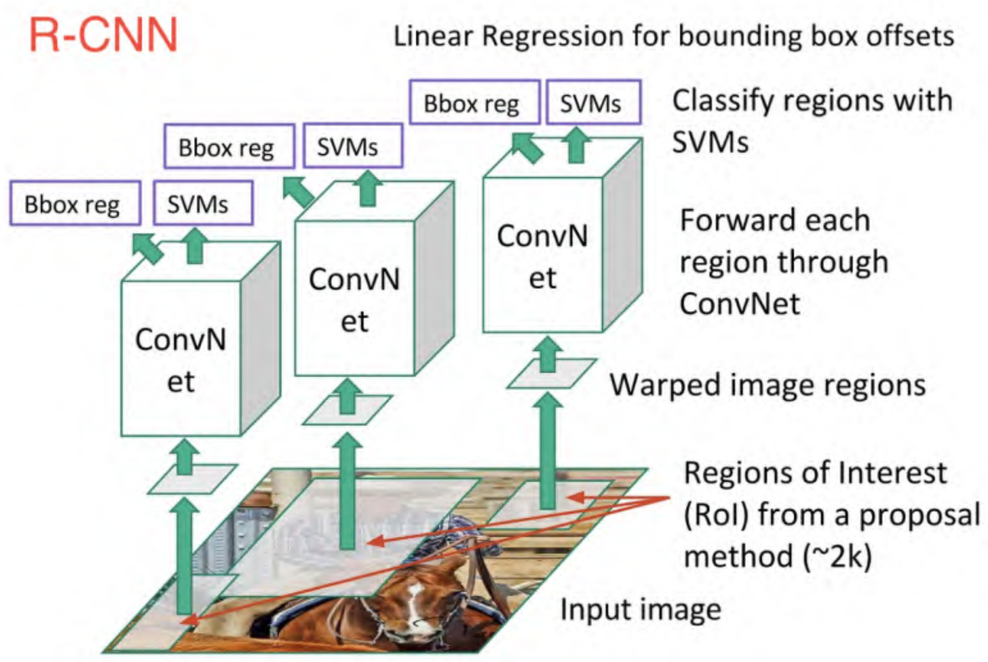

R-CNN
Linear Regression for bounding box offsets
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Figure 2.7: R-CNN (image source [27])

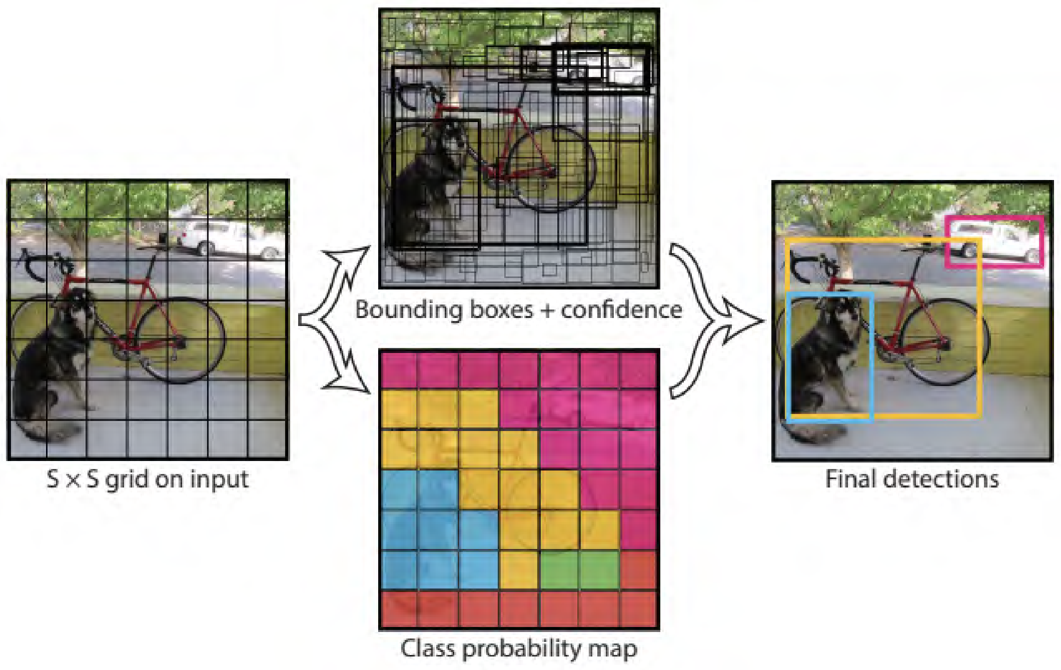


Figure 2.8: YOLO (image source [28])

One-stage detectors

Unlike two-stage detectors, where first prospective regions are proposed and then a CNN is run on these regions to obtain object classes and bounding boxes, in one-stage detectors, class probabilities and bounding box predictions are made directly from the image instead of first proposing regions-of-interest (RoI). YOLO (You Only Look Once) [28] based models are good examples of one-stage detectors. YOLO basically works in the following way, as illustrated in Figure 2.8:

1. We first divide the image into a grid. Each grid-cell symbolises whether it contains the mid-point of an object. Therefore, for every grid-cell we predict a confidence score, which is the probability that this grid-cell contains the mid-point of an object. Apart from the confidence score, for each grid-cell, we also predict a bounding box which should be containing the object whose mid-point is contained by the grid-cell. Apart from the confidence score and bounding box, we also predict the class of the object contained by the grid-cell.

2. Many grid-cells can predict that the mid-point of an object belongs to them, and correspondingly attach a confidence score. We pick the one with the highest confidence score and call it a *Prediction*. All the other bounding boxes with a sufficiently high IoU (refer section 2.4.2) with the *Prediction's* bounding box are discarded. This is called Non-Maximum Supression (NMS). It helps to get rid of multiple detections for the same object.

3. It is important to note that we assumed that a grid-cell can contain the mid-point of only one object, which is not going to be true for many cases. Thus, we come to the concept of anchor boxes, whereby each grid-cell has a predetermined number of anchor boxes of different shapes and sizes associated with it. Now an object is assigned to a grid-cell that contains its mid-point and to an anchor box of that grid-cell that has the largest IoU with the object. Therefore, any grid-cell can now contain the mid-points of as many objects as the number of anchor boxes.

In this current work, we have used YOLOv7 [7] as the object detector, because of two main reasons:

- One-stage detectors are faster than two-stage detectors during inferencing, and are therefore better suited to real-time systems.

- YOLOv7 was the state-of-the-art YOLO model available at the time of writing this thesis (refer Figure 2.9). YOLOv8 was introduced but was still undergoing active development, had many issues and bugs, and the research paper for YOLOv8 was still not published.

2.3.3 Metrics of evaluation

The most popular metric to evaluate Object Detection performance is called *mAP*, or Mean Average Precision. Let us now understand how to calculate *mAP*.

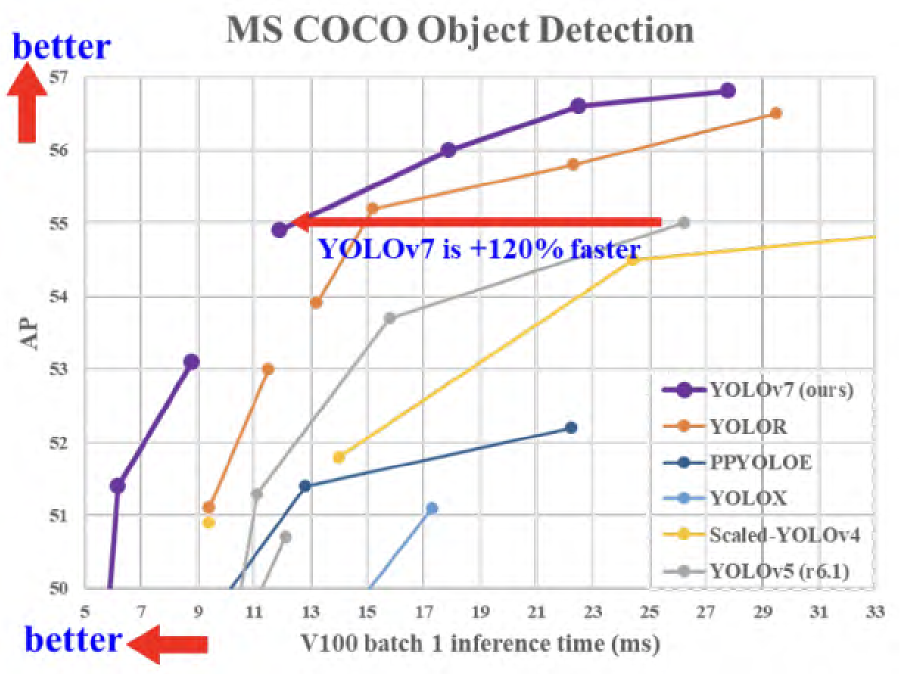


Figure 2.9: YOLOv7 performance (image source [7])

- *Positives*: All the bounding boxes predicted by a detector.
- *True Positives* (TP): All *Positives* that have an IoU (refer Section 2.4.2) >= 0.5 (or another threshold, this value is a hyperparameter) with a ground truth (GT) *bbox*.
- *False Positives* (FP): All *Positives* that are not *True Positives*, i.e. all predicted bounding boxes that have an IoU < 0.5 with a GT *bbox*.
- *False Negatives* (FN): All those GT bounding boxes, that did not get predicted, i.e. they had an IoU < 0.5 with any *Positive*.
- *True Negatives* (TN): This basically corresponds to the entire background of the image where there was not an object and we did not predict any either. This metric is not useful and hence ignored in Object Detection.
- *Precision* is given by Equation 2.1 and *Recall* is given by Equation 2.2. However, they are now calculated class-wise, and a TP or FP is included in the formula only when it has a confidence score >= 0.5 (or another threshold, this value is a hyperparameter) for that object class.
- *Average Precision* (AP): This is the area under the Precision-Recall (PR) curve. AP is calculated class-wise, to unmask class-imbalance in the data and to transparently show whether our detector performs well on certain object classes while not so well on others. To plot the PR curve, *Precision* and *Recall* need to be cumulatively

calculated for every *Positive* in an object-class arranged in descending order of confidence-score. Then, to calculate the area under the curve, *Precision* values can be interpolated over 11 (PASCAL VOC 2007 [29]) or (more recently) 101 (MS COCO 2014-17 [8]) equidistant *Recall* points between 0 and 1. The sum of *Precision* values, at all equidistant *Recall* points, is then divided by the number of equidistant *Recall* points (11 or 101) to obtain AP. Please refer [30] for a good example.

- *Mean Average Precision* (mAP): Once we have calculated AP for every object class, we can sum them up and divide by the number of classes to obtain *mAP*, which is a holistic metric that signifies how good our detector performed over all the classes. *mAP_0.5* is the usually used metric which is defined such that *True Positives* are considered for IoU >= 0.5. However, MS COCO 2014-17 [8] also introduced a more difficult metric *mAP_0.5:0.95*, which calculates *mAP* by changing IoU thresholds in steps of 0.05, from 0.5 to 0.95, and then takes the average of all the *mAP* values.

2.4 Object Tracking

2.4.1 What is Object Tracking?

Object Detection dealt with classifying an object and finding its location in an image. Object Tracking goes a step further, to obtain the trajectory an object follows in an image sequence (e.g. a video). Therefore, Object Tracking is Object Detection + Data Association, whereby Data Association means to match the same object's detections across multiple frames (images). The problem of Single-object tracking involves detecting one particular object of interest and tracking it through a sequence of images. However, the more popular and challenging problem is that of Multi-object tracking, whereby we detect and track multiple objects, through an image sequence, simultaneously.

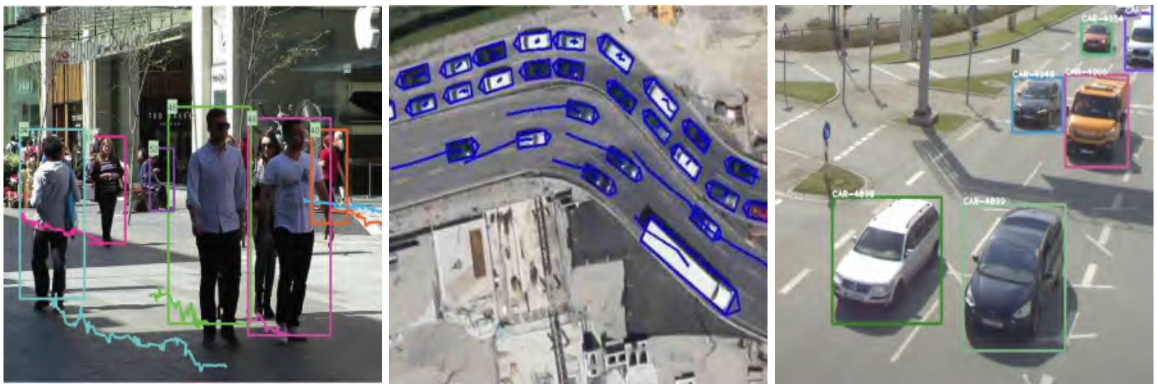


Figure 2.10: Different domains of MOT. (From left to right: MOT-16 (source [31]), KIT AIS (source [32]), A9 (source [6]))

2.4.2 How is it done?

Most of the research and consequently new and better algorithms for tracking objects are developed for the domain of pedestrian tracking thanks to popular MOT (Multi-object tracking) benchmark datasets like MOT-15 [1], MOT-16 [2] and MOT-20 [4]. However, this thesis focuses mainly on tracking vehicles and other traffic participants and that too in two very different scenarios (domains):

- Aerial imagery: The dataset used for this thesis has not been published by the German Aerospace Center (DLR) [10] so far, but a good example of a popular published dataset, that is similar to but not as challenging as the dataset used for this thesis, is the KIT AIS Vehicle Dataset [32] whose image sequences were also provided by DLR.
- Real-time traffic monitoring: For this purpose, the dataset used by this thesis is the A9 dataset [6].

One of the primary goals of this thesis was to adapt the algorithms, that are primarily built for pedestrian tracking, to the two domains that we are interested in (refer Figure 2.10). This is because, as previously mentioned, most of the state-of-the-art research is focused in the pedestrian tracking domain and we wanted to benefit from that research in our domains. Since this thesis deals primarily with Object Tracking, a whole separate chapter (Chapter chapter 3) was deemed necessary to go over the works that have preceded ours and on which we have laid the foundations of this thesis. However, in this section, we can discuss some commonly used techniques/concepts in tracking algorithms, which would be helpful in understanding the algorithms themselves later.

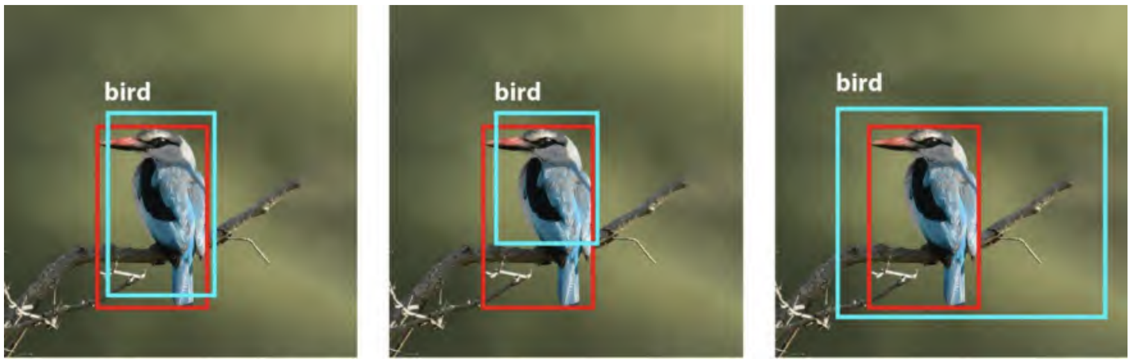


Figure 2.11: IoU: a comparison of different bounding boxes. Red *bbox* is GT and Blue *bbox* is *Prediction* (image source [33])

Kalman filter

- Kalman filter is a popular and complicated algorithm to model an object's motion. To talk about it in detail, would be out of scope for this current thesis. However, [34] is a good reference to understand the working of Kalman filter in depth.

- Basically, the Kalman filter can predict, using a covariance matrix, the new position of the *bbox* and infer the velocity of motion of an object, based on the past measurements of the position of an object's bounding box.
- It generally works on the assumptions that the noise in measurements can be modelled by Gaussian Probability Distribution Function and in a short interval of time (between two frames, for example), the velocity of an object remains constant.
- There are two primary steps involved, in the *learning* process of the Kalman filter:
  - *Prediction*: A new state (position/velocity of the object) is predicted.
  - *Update*: The *Prediction* is combined with the new incoming measurement.
- During training, the new incoming measurement in the *Update* step is the ground truth (GT) *bbox*.
- During inference, the new incoming measurement in the *Update* step is a Detection *bbox*, selected by the Hungarian algorithm (see Section 2.4.2), that had either of the following characteristics:
  - It had a sufficiently high IoU (above a certain threshold, e.g. 0.5, which is a hyperparameter), with the *Prediction* (see Section 2.4.2).
  - Its feature embedding had a sufficiently low cosine distance from the feature embedding of the *Prediction* (see Section 2.4.2).

IoU

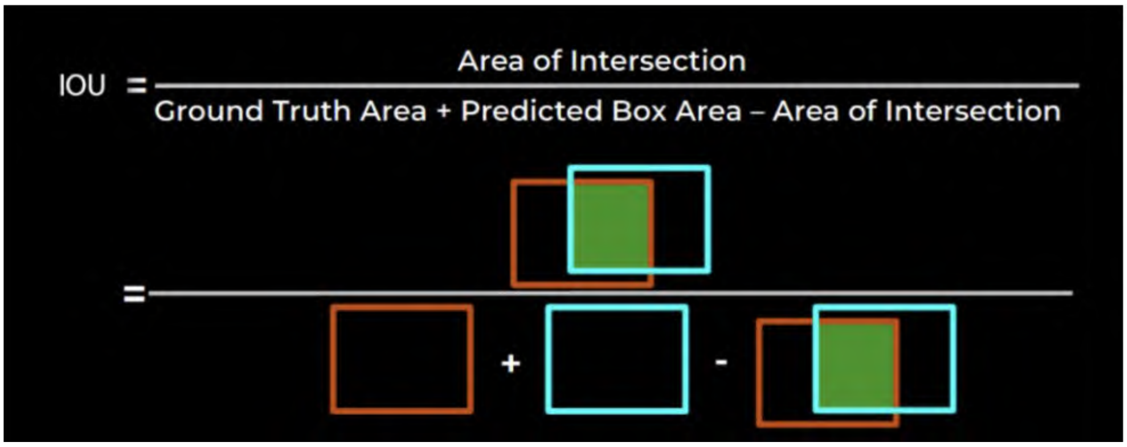

IOU =
Area of Intersection
Ground Truth Area + Predicted Box Area – Area of Intersection
=
+
-

Figure 2.12: IoU: an illustration of how it is calculated. Red *bbox* is GT and Blue *bbox* is *Prediction* (image source [33])

- IoU is the most popular measure of discerning a *Positive* in the world of Object Detection (and consequently Object Tracking).

- The question we are trying to answer is: given a GT *bbox* and several predicted bounding boxes, which *Prediction* best explains/matches the GT. Figure 2.11 basically asks this question, and Figure 2.12 provides the answer.
- The *Prediction* which has the highest overlap with the GT *bbox*, and therefore the highest IoU, is the best choice.

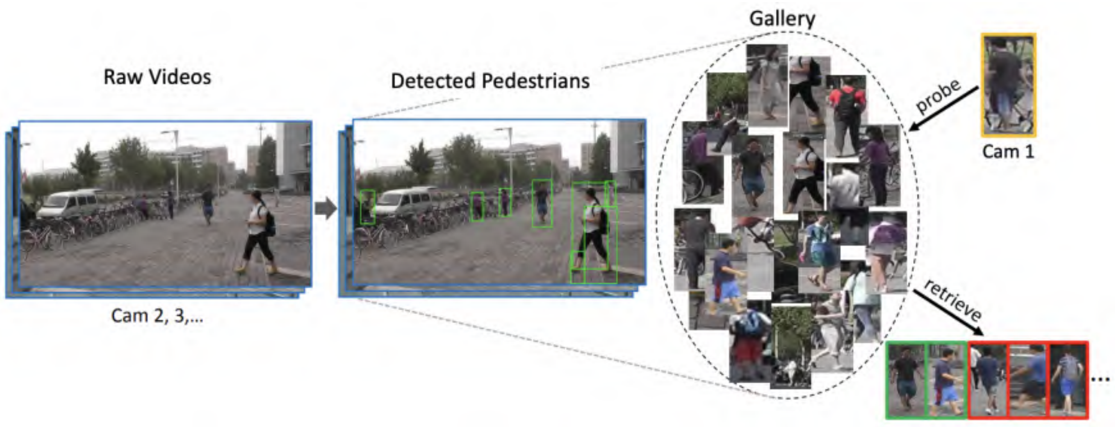


Figure 2.13: Example of re-ID in the domain of person re-identification (image source [35])

re-ID

- re-ID is an abbreviation for re-identification. This technique is generally used in the domain of person re-identification [35]. However, just like Object Tracking, the concept of re-ID can be borrowed from the domain of person re-identification and applied to vehicle re-identification.
- The task of re-ID essentially entails that, given a **query**/**probe** *bbox* object, find one or more *bbox* objects from a **gallery** of *bbox* objects that are basically the same object captured at a different point in time by the same camera or even a different camera at the same point in time or a different point in time (see Figure 2.13).
- The feature embeddings of the **query** and **gallery** bounding boxes are obtained using a ResNet-50 [21] backbone.
- Generally, the top 5, 10, 20, 50, or even 100 closest findings (called **retrieve**) are obtained. The metric of measuring closeness can be the **cosine distance** between the feature embeddings of the **query** and a particular **gallery** *bbox*.
- *Precision*, *Recall* and F_1 *score* can now be calculated based on how many retrievals were actually correct and how many correct ones were not retrieved at all.

- Triplet Loss is regarded as a good loss function to train a model on the task of re-ID [36].
- Offline models trained in the task of re-ID have proven very useful in occlusion handling, and tracking in general. These models can be used to obtain feature embeddings during inference time for tracking by the Kalman filter (see Section 2.4.2).
- However, the operation of obtaining a feature embedding using a deep ResNet-50 [21] backbone is much more computationally expensive than calculating IoU and therefore, re-ID based tracking models are generally slower than IoU based models at inferencing.

Hungarian algorithm

- Consider a bipartite graph, where on one side the nodes represent the GT bounding boxes, and on the other side the nodes represent the predicted bounding boxes.
- We construct an edge (bidirectional, one-to-one mapping, since only one prediction can be assigned to one GT *bbox* and vice versa) from a node on one side to a node on the other side, if there is an IoU >= 0.5 (or any other suitable threshold, hyperparameter) between the bounding boxes represented by these nodes.
- In a crowded situation, several predicted bounding boxes will have an IoU >= 0.5 with several GT boxes. And thus we will end up with a many-to-many mapping, but our goal is to ideally obtain a one-to-one mapping (in an ideal balanced case, where number of detections is equal to the number of GT bounding boxes).
- This conversion of a many-to-many mapping to a one-to-one mapping is generally called the assignment problem, and the goal is to have a one-to-one mapping such that a predetermined cost function is minimised.
- Hungarian algorithm [37] does this job in an efficient manner in polynomial time. As mentioned in the section for Kalman filter (see Section 2.4.2), low cosine distance between feature embeddings of bounding boxes can also be used, instead of high IoU, as a condition to construct edges in the bipartite graph.

2.4.3 Metrics of evaluation

In this thesis, we use the three most popular and widely accepted Object Tracking evaluation metrics namely MOTA [38], IDF1 [39] and HOTA [40]. The information in this section has been distilled and consolidated from the following sources: [38], [39], [40], [41], [2] and [42].

Common errors in MOT

- *False Negative* (FN): When GT exists but we did not predict it.
- *False Positive* (FP): When we make a prediction but a corresponding GT does not exist.
- *Identity Switch* (IDSW): An object's track-id is switched with another object's track-id. Basically, one object is confused for another object (see Figure 2.14(a)). Generally, this happens when the objects move close to each other (IoU based), or they look very similar (re-ID based).
- *Fragmentation* (Frag): A GT track is predicted for some frames, then fails to be predicted for some frames, and then again gets predicted (see Figure 2.14(b)). Simply put, fragmented tracks are basically broken tracks. However, if a track is broken and is never predicted again, then it is not counted as fragmented (see Figure 2.14(c)).

It is expected that a metric to evaluate Object Tracking, takes into account these very common errors.

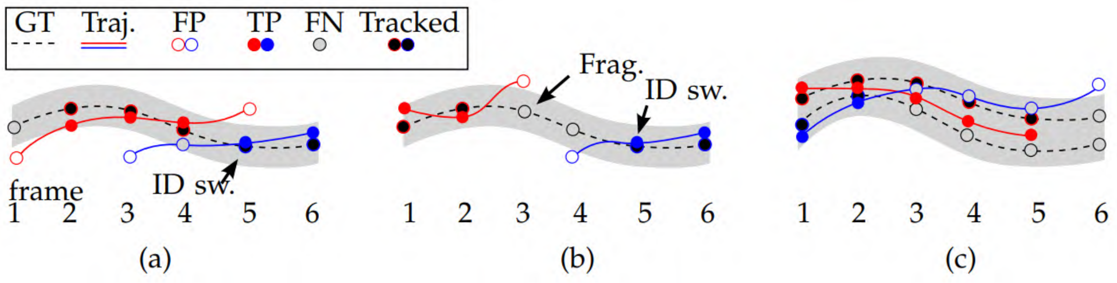


Figure 2.14: Common errors in MOT. (a) An ID switch occurs when the mapping switches from the previously assigned red track to the blue one. (b) A track fragmentation is counted in frame 3 because the target is tracked in frames 1-2, then interrupts, and then reacquires its 'tracked' status at a later point. A new (blue) track hypothesis also causes an ID switch at this point. (c) Note that no fragmentations are counted in frames 3 and 6 because tracking of those targets is not resumed at a later point. (image and caption source [2])

MOTA

- Multiple Object Tracking Accuracy (MOTA) is the most widely used metric for evaluating Object Tracking performance, since it is able to take into account 3 very common sources of errors (as given in section 2.4.3), namely, FN, FP and IDSW.

- The errors made by a model are accounted for in each frame $t \in T$ as given by Equation 2.6.

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t} \tag{2.6}$$

- However, it is debated that MOTA alone is perhaps not a sufficient measure of performance, since it gives a lot of significance to detection performance over data-association performance.

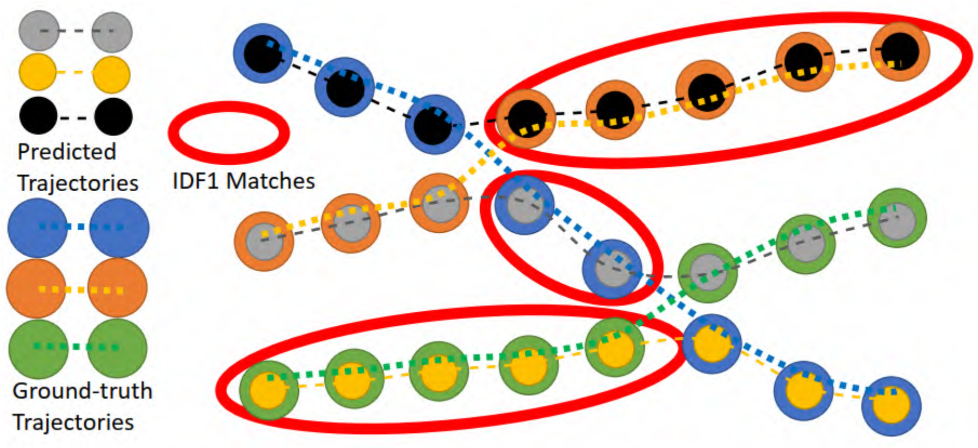


Figure 2.15: IDF1 calculation example (image source [40])

IDF1

- The Identity metrics focus on measuring data-association performance rather than detection performance and are therefore often used together with MOTA.
- Given a set of GT tracks and a set of predicted tracks, the goal is to have a one-to-one matching between them.
- This matching is done using the Hungarian algorithm (see Section 2.4.2), whereby we try to obtain a single best set of matching trajectories (portions of tracks) between the GT and predicted tracks (see Figure 2.15).
- However, any trajectory that does not end up in this matching set is counted as a *Negative* irrespective of whether it contributes to correct detections. This leads to a decrease in the score. Thus, a high IDF1 score is indicative of a good estimation of unique objects in the image-sequence but not good detections. Also, IDF1 fails to evaluate the localization accuracy of tracking algorithms.
- The conventional metrics, as we saw in Section 2.2.3, like TP, FN, FP, Precision, Recall and F_1 are now referred to as IDTP, IDFN, IDFP, IDP (see Equation 2.7), IDR

(see Equation 2.8) and IDF1 (see Equation 2.9) respectively.

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{2.7}$$

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{2.8}$$

$$IDF1 = \frac{2 * IDP * IDR}{IDP + IDR} = \frac{2 * IDTP}{2 * IDTP + IDFP + IDFN} \tag{2.9}$$

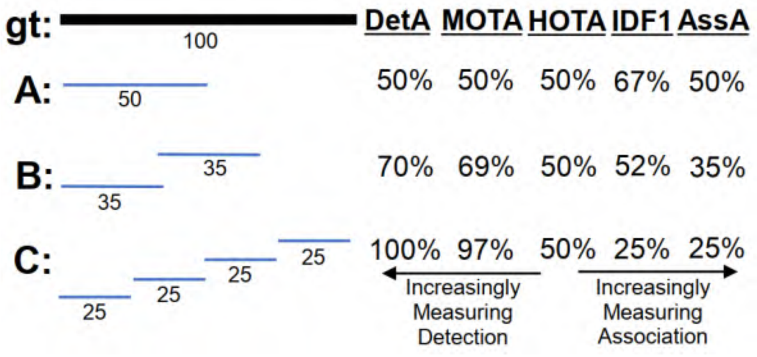


Figure 2.16: A simple tracking example highlighting one of the main differences between evaluation metrics. Three different trackers are shown in order of increasing detection accuracy and decreasing association accuracy. MOTA and IDF1 overemphasize the effect of accurate detection and association respectively. HOTA balances both of these by being an explicit combination of a detection score DetA and an association score AssA. (image and caption source [40])

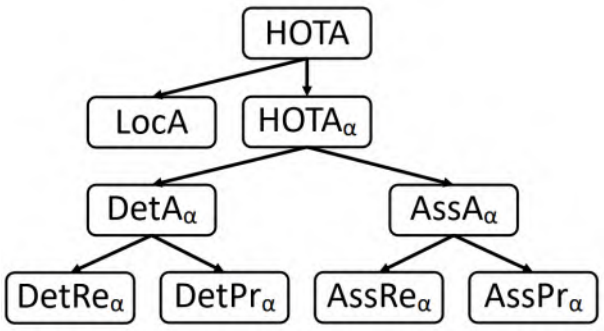


Figure 2.17: Diagrammatic representation of how HOTA can be decomposed into separate sub-metrics which are able to differentiate between different types of tracking errors. (image and caption source [40])

HOTA

- Higher Order Tracking Accuracy (HOTA) is a more recent, unified and holistic metric, to evaluate tracking performance, in the sense that it does not bend towards detection performance like MOTA or data-association performance like IDF1 (refer Figure 2.16).

- HOTA takes into account all three (detection, association, localisation) performances by measuring and incorporating LocA (Localisation Accuracy), DetA (Detection Accuracy) and AssA (Association Accuracy), as can be seen in Figure 2.17.

- In Figure 2.17, α signifies a particular localisation threshold, e.g. 0.5, basically like IoU (see Section 2.4.2). We calculate $HOTA_\alpha$ over 19 values of α ranging from 0.05 to 0.95 in steps of 0.05, and then integrate to find the final HOTA, as given by Equation 2.10.
$$HOTA = \int_0^1 HOTA_\alpha \, d\alpha \approx \frac{1}{19} \sum_\alpha HOTA_\alpha \tag{2.10}$$

- *Localisation Accuracy* (LocA): Localisation Accuracy is basically the average Localisation-IoU over all *True Positives* (TP) in the entire sequence, and is given by the Equation 2.11, where $S(c)$ is the spatial similarity score (basically the IoU) between a predicted detection (prDet) and a GT detection (gtDet) that make up a TP c.
$$LocA = \int_0^1 \frac{1}{|TP_\alpha|} \sum_{c \in \{TP_\alpha\}} S(c) \, d\alpha \tag{2.11}$$

- *Detection Accuracy* (DetA): DetA can easily be obtained from the commonly used metrics, namely, DetRe (Detection Recall) and DetPr (Detection Precision), which have already been explained in Section 2.3.3. DetA basically gives a percentage of how many predicted and GT detections could be aligned together, and is given by the Equation 2.12.
$$DetA_\alpha = \frac{DetRe_\alpha * DetPr_\alpha}{DetRe_\alpha + DetPr_\alpha - DetRe_\alpha * DetPr_\alpha} = \frac{|TP|}{|TP| + |FN| + |FP|} \tag{2.12}$$

- *Association Accuracy* (AssA): Association Accuracy(AssA) is the average alignment between matched trajectories (predicted and GT), averaged over all detections. For a given TP c, we can now define:
  - *True Positive Association* (TPA): A TPA for a given TP c is a TP which has the same gtID (GT track-id) and prID (predicted track-id) as c.
  - *False Negative Association* (FNA): A FNA for a given TP c is a gtDet with the same gtID as c, but was assigned a different prID than c or no prID at all if it was missed.

- *False Positive Association* (FPA): A FPA for a given TP c is a prDet with the same prID as c, but was assigned a different gtID than c or no gtID if it did not actually correspond to an object.

AssA can now be obtained from the Equation 2.13.

$$AssA_\alpha = \frac{1}{|TP|} \sum_{c \in \{TP\}} \frac{|TPA(c)|}{|TPA(c)| + |FNA(c)| + |FPA(c)|} \tag{2.13}$$

- *Higher Order Tracking Accuracy* (HOTA): HOTA can now easily be obtained from DetA and AssA, as given by Equation 2.14.

$$HOTA_\alpha = \sqrt{DetA_\alpha * AssA_\alpha} \tag{2.14}$$

3 Related Work

3.1 Goals

The goal of this chapter is to discuss several prominent and preceding works, which have been an inspiration and have been used as a foundation for this current work. We can classify the related works in three ways, namely, previous theses, methodology based (different primary architectures) and state of the art. For every work, we discuss its key contributions, architecture, results on popular benchmarks, perceived advantages and disadvantages when applied to the field of vehicle tracking in aerial imagery, and overall remarks in general.

3.2 Previous Theses

This thesis work was carried out in 2022-2023 and was a direct successor of two other theses ([42] (2021) and [43] (2020)), that were based on object detection and tracking in aerial imagery, and were also done in the collaboration of TUM I6 Chair [44] and DLR [10].

3.2.1 Beheim et al., 2021

Contributions

- [42] compared the performance of several tracking algorithms on the KIT AIS Vehicle Dataset [32] (see Figure 3.2).
- Used the joint-tracking-and-detection paradigm (instead of the more usual tracking-by-detection paradigm), with TransTrack [45] as a baseline.
- Attempted to improve tracking performance using Angle Prediction (orientation of vehicles), re-ID (see section 2.4.2) and Motion Prediction using an LSTM [46] based motion model.

Architecture

- The primary architecture is based on the TransTrack [45] architecture (see Figure 3.1), which in turn uses a Transformer [47] based encoder-decoder architecture to perform detection and tracking simultaneously instead of treating them as two disjoint tasks.

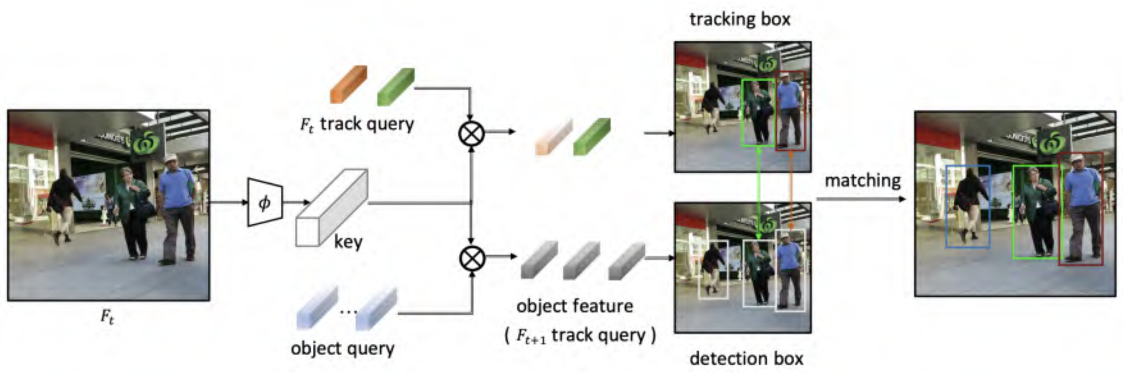


Figure 3.1: Pipeline of TransTrack (image and caption source [45])

Results

- The work [42] reported that **TransTrack [45], without modifications, achieved a MOTA score of 78.1 and an IDF1 score of 86.8 on the KIT AIS Vehicle Dataset [32]**.
- With the application of **re-ID** on top of TransTrack [45], **a MOTA score of 78.5 and an IDF1 score of 86 were achieved on the KIT AIS Vehicle Dataset [32]**, which were the best reported results that were achieved overall.
- Angle Prediction and Motion Prediction led to worse results than the baseline TransTrack results.

Advantages

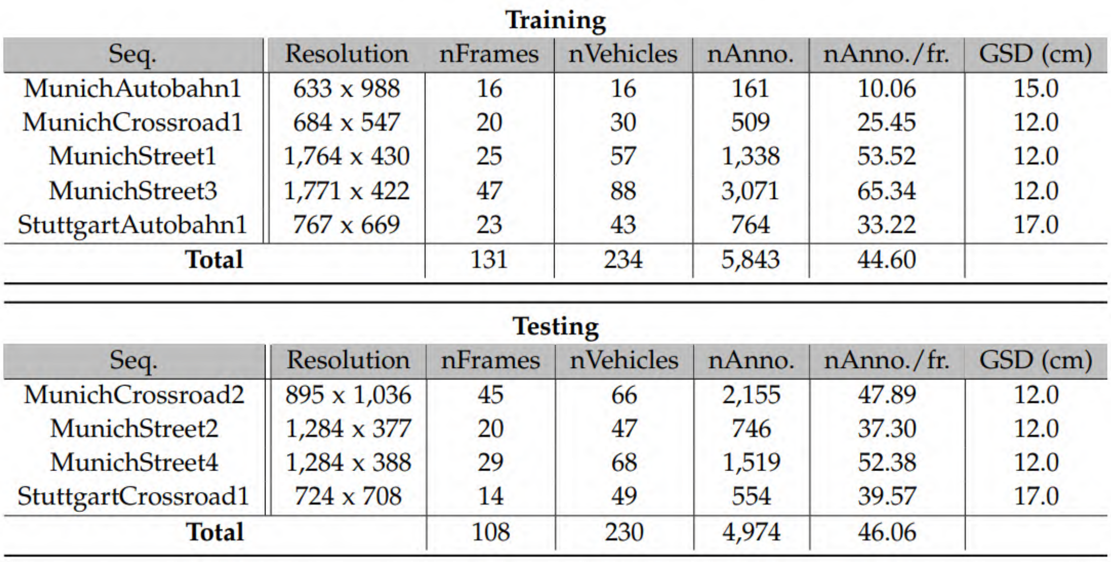
- It showed that **re-ID is useful in object tracking**.

Disadvantages

- **The joint-tracking-and-detection paradigm cannot leverage the power of the new state-of-the-art object detectors** (e.g. YOLOv7 [7]) that can be independently trained on much larger object detection datasets (e.g. MS COCO [8]), to obtain better tracking performance on small datasets (e.g. KIT AIS Vehicle Dataset [32]).

Remarks

- Transformer [47] based architectures are difficult to train in the sense that they require a lot of training data, which is unfortunately not available in the domain of vehicle tracking in aerial images.
- A Transformer [47] based architecture coupled with re-ID (see Section 2.4.2) and LSTM [46] based motion prediction, is not suitable for inferencing in real-time systems because it is going to be extremely slow and memory-intensive.



Training

Seq.	Resolution	nFrames	nVehicles	nAnno.	nAnno./fr.	GSD (cm)
MunichAutobahn1	633 x 988	16	16	161	10.06	15.0
MunichCrossroad1	684 x 547	20	30	509	25.45	12.0
MunichStreet1	1,764 x 430	25	57	1,338	53.52	12.0
MunichStreet3	1,771 x 422	47	88	3,071	65.34	12.0
StuttgartAutobahn1	767 x 669	23	43	764	33.22	17.0
Total		131	234	5,843	44.60	

Testing

Seq.	Resolution	nFrames	nVehicles	nAnno.	nAnno./fr.	GSD (cm)
MunichCrossroad2	895 x 1,036	45	66	2,155	47.89	12.0
MunichStreet2	1,284 x 377	20	47	746	37.30	12.0
MunichStreet4	1,284 x 388	29	68	1,519	52.38	12.0
StuttgartCrossroad1	724 x 708	14	49	554	39.57	17.0
Total		108	230	4,974	46.06	

Figure 3.2: An overview of the KIT AIS Vehicle Dataset (image source [43])

3.2.2 Kraus et al., 2020

Contributions

- [43] introduced the AerialMPT dataset [48], which has a comprehensive collection of pedestrian data in various crowded situations.
- Achieved state-of-the-art tracking results in aerial image based pedestrian datasets like KIT AIS Pedestrian [32] and AerialMPT [48], by proposing AerialMPTNet [48].
- Evaluated AerialMPTNet [48], which was developed primarily for pedestrian tracking in aerial image sequences, on KIT AIS Vehicle dataset [32] (see Figure 3.2) as well.

Architecture

- Used a Siamese Neural Network (SNN) architecture, baselined on the work [49], with tracking-by-regression paradigm.
- An LSTM module [46] was used as a motion vector predictor.
- A Graph Convolutional Neural Network (GCNN) [50] module was used to capture the relative motion between nearby objects.
- Refer Figure 3.3 for an architectural overview.

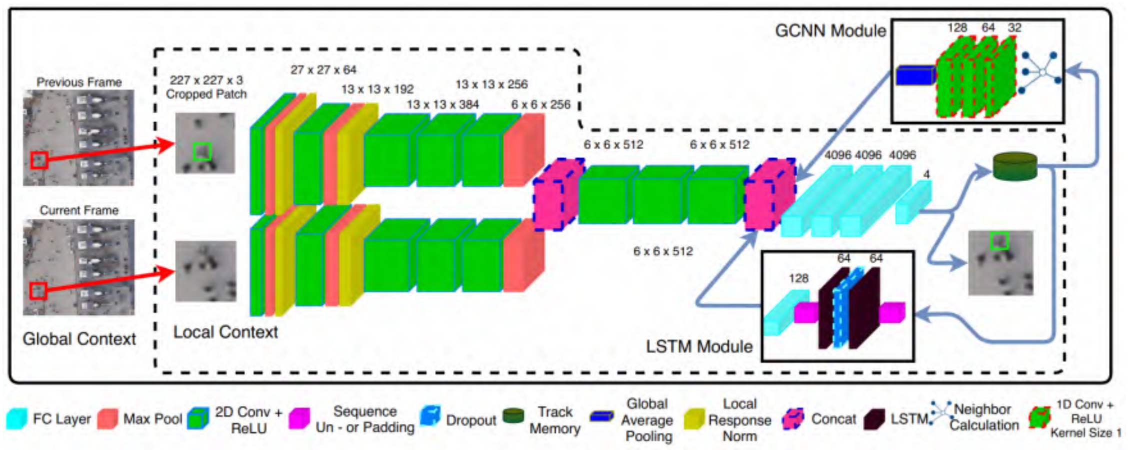


Figure 3.3: Overview of the AerialMPTNet architecture including an SNN, an LSTM, and a GCNN module. The inputs are two consecutive images in a sequence, cropped and centered to a target object, and the output is the object coordinates in the second snippet which is then mapped to the image coordinates. (image and caption source [48])

Results

- Since our thesis is focused on Vehicle Tracking, it makes sense to make a note of AerialMPTNet's performance on a Vehicle Tracking dataset, even though it was primarily designed for pedestrian tracking. **AerialMPTNet obtained a MOTA score of 42.0 and an IDF1 score of 70.0 [48] on the KIT AIS Vehicle Dataset [32]**.

Advantages

- The method works well on pedestrian datasets. It shows successfully that **Deep Learning visual features can be used for small objects**, if the bounding box area is rescaled and more contextual information is taken into account.

Disadvantages

- It does not work so well on Vehicle datasets. The neighborhood considered for GCNN to find 8 nearest objects is not enough, since vehicles can be quite far apart in comparison to pedestrians. Therefore, **local neighborhood is not sufficient for vehicle tracking**.

Remarks

- Vehicles travel faster than pedestrians. With small FPS, the neighborhood of a vehicle can change considerably (old vehicles may be gone, new vehicles may have entered etc.). Therefore, the GCNN approach of a limited neighborhood will not work.

- The Siamese network takes the current frame centered at the object from the previous frame. From the previous point it is again evident that with low FPS and a high-speed vehicle, detection cannot be regressed to a local neighborhood.
- The LSTM is used in a global fashion (to reduce computation), taking 5 previous positions to give a motion vector (x,y) which is used to predict (regress) the next position. However, this does not take into account that different objects are at different velocities (and acceleration) and therefore a single global LSTM cannot be used to predict motion for all.

3.3 Methodology based

We now shift our attention to several methods that were quite diverse in their primary architectures, and managed to produce competitive results on the MOT [1, 2, 3] benchmarks.

3.3.1 MPNTrack

Contributions

- [51] used a **Graph Neural Networks** [52] based approach to tackle the problem of Multiple Object Tracking (MOT) on MOT benchmarks that deal with pedestrian tracking.
- Proposed a MOT solver based on Message Passing Networks (MPN), which can exploit the natural graph structure of the problem to perform both feature learning as well as final solution prediction.
- Proposed a novel time-aware neural message passing update step inspired by classic graph formulations of MOT.

Architecture

While reading the following points, it is recommended to refer to the Figure 3.4:

- Receive as input a set of frames and detections.
- Construct a graph in which nodes represent detections, and all nodes at different frames are connected by an edge.
- Initialize node embeddings in the graph with a CNN, and edge embeddings with an MLP (multi-layered perceptron) encoding geometry information.
- The information contained in these embeddings is propagated across the graph for a fixed number of iterations through neural message passing.

- Once this process terminates, the embeddings resulting from neural message passing are used to classify edges into active (colored with green) and non-active (colored with red). During training, the cross-entropy loss of the predictions w.r.t. ground truth labels is computed and gradients are backpropagated through the entire pipeline.
- At inference, a simple rounding scheme is followed to binarize the classification scores and obtain final trajectories.

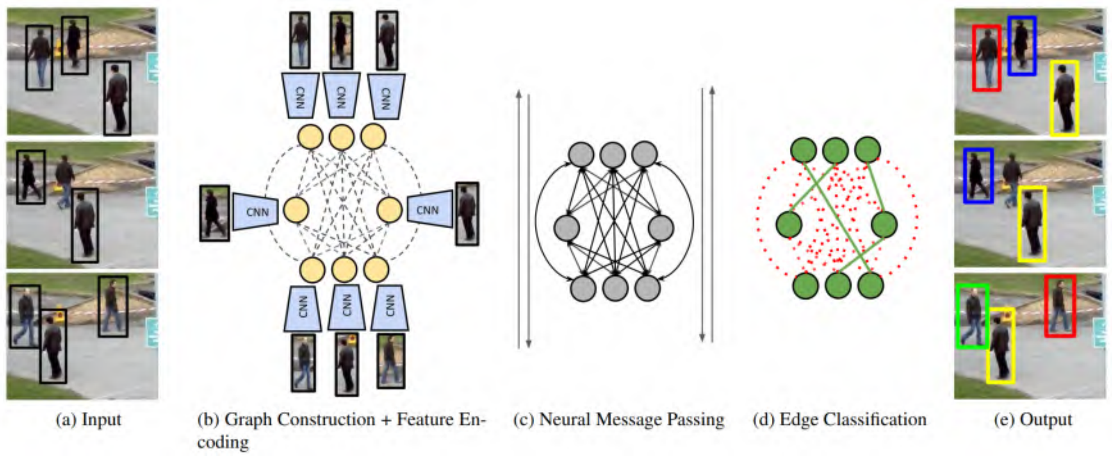

CNN
(a) Input
(b) Graph Construction + Feature Encoding
(c) Neural Message Passing
(d) Edge Classification
(e) Output

Figure 3.4: Overview of MPNTrack (image source [51])

Results

- An overview of MPNTrack's results on popular MOT benchmarks can be found in Table 3.1.

MPNTrack [51] Results		
Benchmark	**MOTA**	**IDF1**
MOT15 [1]	51.5	58.6
MOT16 [2]	58.6	61.7
MOT17 [3]	58.8	61.7

Table 3.1: An overview of MPNTrack's results on MOT benchmarks.

Advantages

- Method takes into account the **global picture** and gathers higher order information through message passing steps.

- **Handcrafted geometric features** in the edge embeddings offer opportunity to include other features as seen fit.

Disadvantages

- This method is an **Offline Tracker**, since it uses future frames, and therefore cannot be used for real-time systems.
- **No** specific enhancements to deal with **camera motion**.

Remarks

- The visual features are extracted using a ResNet50 [21] backbone which is pre-trained on ImageNet [18] and then retrained for the task of re-ID (see Section 2.4.2) on multiple public datasets. After that it is used in the pipeline, whereby each detection passes through it to be converted to a node embedding. However, there is no local context information in these detections, which might be useful for countering camera motion.
- A separate (independent) dedicated re-ID module can be used which is trained for re-ID on the current dataset itself, instead of other datasets, so that it has more domain specific knowledge.
- An edge always exists between detections of two different frames, and therefore how the camera moved between these frames (relative camera motion) could have been explicitly embedded in the edge.

3.3.2 SiamMOT

Contributions

- [53] proposed an implicit motion model (IMM) and an explicit motion model (EMM) (for tracking) on top of a **Siamese Neural Network** (SNN) architecture, which uses Faster-RCNN [25] architecture's Region Proposal Network (RPN) (for detections).
- Achieved competitive results on three popular benchmarks that offer a variety of challenges, namely, MOT17 [3] (occlusion and crowded scenes), TAO-person [54] (wide range of scene types and video corruption artifacts), and Caltech Roadside Pedestrians (CRP) [55] (large camera motion).

Architecture

While reading the following points, it is recommended to refer to the Figure 3.5:

- SiamMOT [53] takes as input two frames I^t and $I^{t+\delta}$, where the goal is to associate a set of detections $R^t = \{R_1^t, ..., R_i^t, ...\}$ in frame I^t to the detections $R^{t+\delta}$ in the next frame $I^{t+\delta}$.
- For this purpose, a motion model is used to propagate every detected *bbox* R_i^t at time t to $\tilde{R}_i^{t+\delta}$ at time $t+\delta$.
- The new detections $R_i^{t+\delta}$ are obtained by running a RPN based method [25] on search areas $S_i^{t+\delta}$ of the frame $I^{t+\delta}$. These search areas were obtained by expanding R_i^t by a factor $r(>1)$, while maintaining the same geometric center.
- A spatial matching process now associates the output of the motion model $\tilde{R}_i^{t+\delta}$ with the new detections $R_i^{t+\delta}$ at time $t+\delta$ such that the detected instances are linked from t to $t+\delta$.

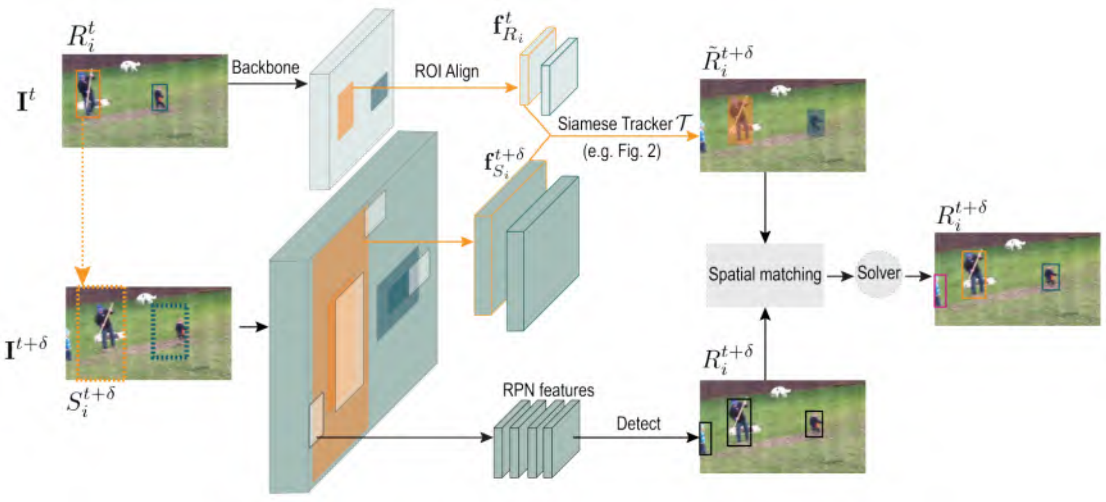


Figure 3.5: Overview of SiamMOT (image source [53])

Results

- **SiamMOT [53] achieved a MOTA score of 65.9 and an IDF1 score of 63.3 on MOT17 [3] benchmark**'s test set with public detections.

Advantages

- **Method was shown to work** on a dataset (CRP [55]) **with fast camera motion**.
- **Online tracking**.
- Ablation study done in the paper clearly shows that **motion modeling gives better online tracking results**.

Disadvantages

- Gives importance to **local features**, and thus **poor at long-term occlusion handling** and re-identification.

Remarks

- Just like many other Siamese Neural Network based architectures, this does not seem suitable for low FPS datasets, since the search region is always in the vicinity of the target.
- Most results were obtained on person-based high FPS datasets and people move slower in comparison to vehicles. Thus, local neighborhood may not be a good enough search region for low FPS vehicle-based datasets.
- Using an off-the-shelf one-stage detector (see Section 2.3.2) would be faster than the current RPN based two-stage detection approach (see Section 2.3.2).
- The spatial matching block could have been removed if instead of computing a location map, that encodes the offset from that location to the top-left and bottom-right *bbox* corners, they had used a more direct re-ID (see Section 2.4.2) based approach. However, in exchange for gaining speed, they would have had to throw away EMM which did give them good results.

3.3.3 TrackFormer

Contributions

- [56] proposed an end-to-end trainable multi-object tracking approach, which achieves detection and data association in a new tracking-by-attention paradigm, using a **Transformer** [47] based architecture.
- They also proposed the concept of autoregressive track queries which embed an object's spatial position and identity, thereby tracking it in space and time.
- TrackFormer [56] achieved competitive results on two popular benchmarks, namely, MOT17 [3] for multi-object tracking and MOTS20 [57] for tracking as well as segmentation.

Architecture

While reading the following points, it is recommended to refer to the Figure 3.6:

- The architecture consists of a common CNN backbone, e.g. ResNet50 [21], for image feature extraction at the frame-level.
- A Transformer [47] encoder is used for image feature encoding with self-attention.

- A Transformer decoder is used to produce output embeddings with *bbox* and class information, by applying self-attention as well as encoder-decoder attention.
- At frame $t = 0$, the decoder transforms N_{object} object queries (white) to output embeddings, either initializing new autoregressive track queries or predicting the background class (crossed).
- On subsequent frames, the decoder processes the joint set of $N_{object} + N_{track}$ queries to follow or remove (blue) existing tracks as well as initialize new tracks (purple).

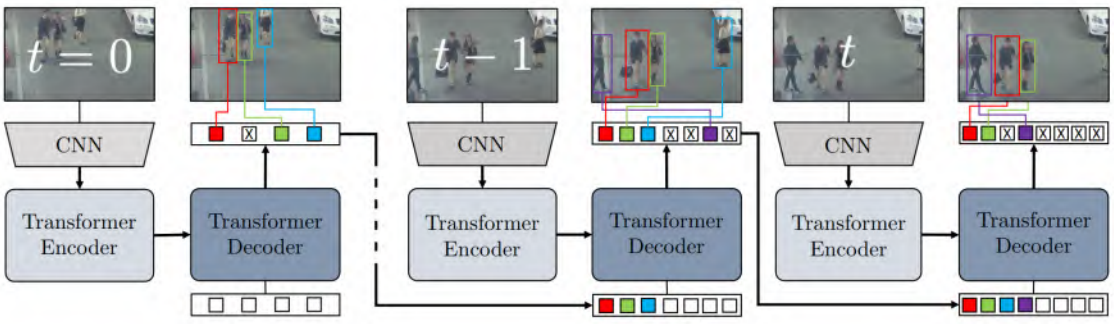


Figure 3.6: Overview of TrackFormer (image source [56])

Results

- An overview of TrackFormer's results on MOT17 [3] benchmark can be found in Table 3.2.

TrackFormer [56] Results		
Type of Detections	**MOTA**	**IDF1**
Public	62.3	57.6
Private	74.1	68.0

Table 3.2: An overview of TrackFormer's results on MOT17 test set.

Advantages

- **Online Tracking**.
- Can **handle short-term occlusion**, with a patience window during which inactive tracks are stored.

Disadvantages

- May not be good to track **fast moving objects** or **low FPS data**, because tracking queries have **spatial embeddings**.
- **Require huge amount of data** to be trained because of the Transformer architecture. Ablation study shows that without training on huge amount of data, the results are not competitive.
- **Requires a lot of training time and GPU resources**. To obtain results on MOT17 with Private Detections (see Table 3.2), it required 2 days of training on 7 GPU units, each of 32 GB capacity.

Remarks

- The strong point of this work is that they try to jointly do detection and tracking, based on feature level attention, to avoid any expensive graph optimization or inclusion of explicit motion modelling or appearance-based re-ID models.
- However, this is also its weak point because the transformer architecture by itself is still computation heavy, devoid of long-term re-ID (no graphs), incapable of handling long-term occlusions (no appearance-based re-ID), and unable to work with fast motion or low FPS data (no motion model).
- Therefore, the method seems quite tailored for pedestrian datasets with high FPS and with relatively slow moving objects.

3.4 State of the Art

In this section, we look at a few methods that have produced state-of-the-art results on the MOT [3, 4] benchmarks, and were therefore considered to be applied to the DLR dataset (Chapter 4) and the A9 dataset (Chapter 5), as part of this thesis.

3.4.1 ByteTrack

Contributions

- The main contribution of [11] is an algorithm that is based on tracking-by-detection paradigm and is built on the simple premise of using both high and low score detections during tracking, unlike previous methods that would generally throw away low score detections (confidence score < 0.5).
- The algorithm is capable of integrating any off-the-shelf detector and another tracking method within it.

Architecture

- ByteTrack [11] is more of an algorithm or a methodology, which can be applied to other methods in hopes of increasing their performance, or can even be used standalone as a tracking algorithm.
- Therefore, we do not have an architecture to discuss about as such, but we can discuss the steps of the algorithm itself.
- Step 1: Given an image (one frame of a video sequence), make object detections using any pretrained detector e.g. YOLOv7 (refer Section 2.3.2).
- Step 2: Construct two disjoint sets of detections, high-score detections and low-score detections, based on a given threshold (0.5).
- Step 3: Predict track location of existing tracks using Kalman Filter (see Section 2.4.2), and construct a set of existing tracks.
- Step 4: Associate detection boxes, from the set of high-score detections, with tracks from the set of existing tracks. Some detections and tracks may not be associated and they will be called remaining high-score detections and tracks respectively.
- Step 5: Associate detection boxes, from the set of low-score detections, with tracks from the set of remaining existing tracks. The low-score detections that are not matched are now thrown away and the tracks from the remaining tracks that did not get matched are also deleted (actually we can construct a set of lost tracks for a certain number of frames for re-ID purposes, but we will ignore that for simplicity here).
- Step 6: The remaining high-score detections are initialized as new tracks, and added to the set of existing tracks. The whole process can now be repeated for the next image in the video sequence.

Results

- An overview of ByteTrack's results on popular MOT benchmarks can be found in Table 3.3.

ByteTrack [11] Results				
Benchmark	**MOTA**	**IDF1**	**HOTA**	**FPS**
MOT17 [3]	80.3	77.3	63.1	29.6
MOT20 [4]	77.8	75.2	61.3	17.5

Table 3.3: An overview of ByteTrack's results on MOT benchmarks.

Advantages

- Good **utilization of low-score detections**.
- Can **handle occlusion** and motion blur, with track interpolation.
- Good **inference speed** (FPS, refer Table 3.3).

Disadvantages

- No explicit module to deal with **camera motion**.

Remarks

- In the paper, they performed detections using YOLOX [58]. We can use the more advanced YOLOv7 [7], which is almost guaranteed to better the performance, since this method's performance is highly dependent on the quality of the detections (tracking-by-detection paradigm).

3.4.2 UAVMOT

Contributions

- [59] tackles the problem of Multiple Object Tracking (MOT) in unmanned aerial vehicle (UAV) videos, which is more challenging than the conventional MOT in pedestrian datasets because of large and irregular camera motion and view changes in 3D directions.
- Proposed an ID feature update (IDFU) module to enhance object ID embedding features, which could update ID features adaptively with a UAV's changing views.
- Developed an adaptive motion filter (AMF) for complex motion tracking of objects in UAV videos, which adaptively switches motion filters to adapt to the movement of a UAV.
- Designed a novel gradient balanced focal (GBF) loss to supervise the learning of objects' heatmaps, which not only considers the imbalanced categories but also focuses on the small-scale objects in UAV videos.

Architecture

While reading the following points, it is recommended to refer to the Figure 3.7:

- We take the current image frame I_t and the previous image frame I_{t-1}, and pass them through a shared feature extractor and detection heads to perform object detection.

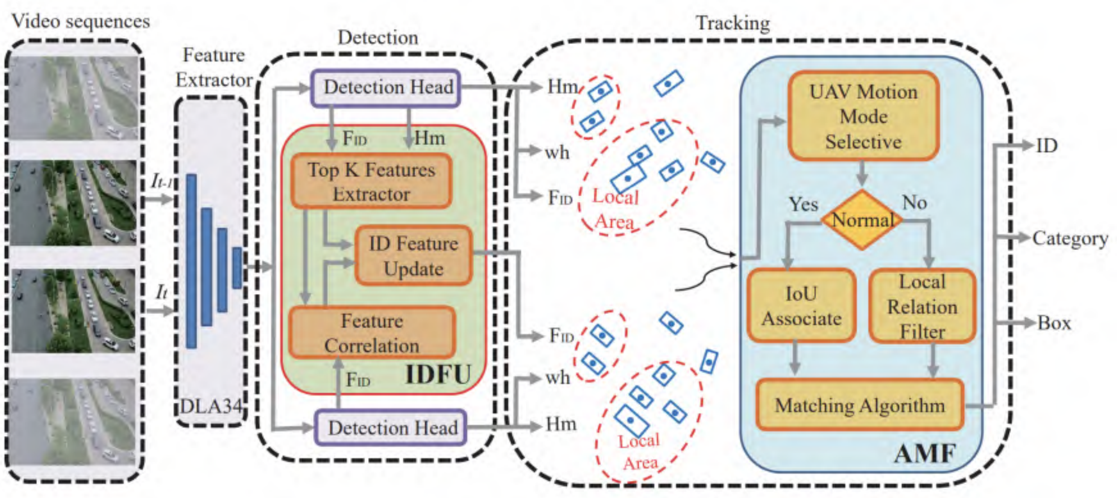


Figure 3.7: Overview of UAVMOT (image source [59])

- The detection head consists of object *bbox* size *wh*, heatmaps *Hm*, and tracking ID embedding features F_{ID}. ID embedding feature is a feature that can point to a particular ID/object.
- The IDFU module extracts the previous frame object features to associate with current frame features, which can adaptively update the ID embedding features in various UAV views.
- Kalman Filter (see Section 2.4.2) is first used on the previous frame's detections to predict object motion, and then an IoU (see Section 2.4.2) is performed with the current frame's detections. During this IoU, if many matches are found between existing tracks and new detections, it is assumed that the UAV camera did not undergo any sudden complex motion. However, if the number of matches is below a threshold, a Local Relational Filter is used to match tracks and detections assuming that the camera underwent some abrupt motion between the frames, and therefore Kalman Filter was not enough for association. This is basically the AMF principle introduced in this paper.
- Please refer to Figure 3.8 to understand the concept of Local Relational Filter. Basically, for every detection in the previous frame and the current frame, we calculate a relative relation vector $v = [l_{max}, l_{min}, \theta]$ where l_{max} is the distance to the farthest object, l_{min} is the distance to the nearest object, and θ is the angle between them. These three elements are all calculated in a local area around a particular detection. It is assumed that even if the UAV camera moved between frames, the relative relation vector for an object would have remained constant and this assumption is used for tracking instead of the usual Kalman Filter.

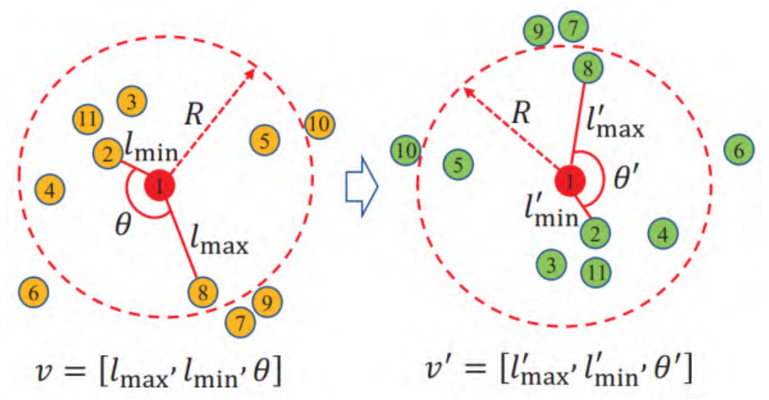

R
l_{min}
l_{max}
θ
$v = [l_{\text{max}}, l_{\text{min}}, \theta]$
R
l'_{max}
l'_{min}
θ'
$v' = [l'_{\text{max}}, l'_{\text{min}}, \theta']$

Figure 3.8: Schematic diagram of relative relation vector (image and caption source [59])

Results

- An overview of UAVMOT's results can be found in Table 3.4.

UAVMOT [59] Results		
Benchmark	**MOTA**	**IDF1**
VisDrone [15]	36.1	51.0
UAVDT [60]	46.4	67.3

Table 3.4: An overview of UAVMOT's results.

Advantages

- Gradient Balanced Focal (GBF) **loss function for small object detection** and class imbalance.
- Adaptive Motion Filter (AMF) to **deal with abrupt camera motion**.
- ID Feature Update (IDFU), which **uses appearance features for better data association**.

Disadvantages

- Does not explicitly address **occlusion handling**, or **long term re-ID**.
- The assumption for the working of Local Relational Filter, that the relative relation vector (refer Figure 3.8) remains constant from one frame to another would not hold for **low FPS dataset**.

Remarks

- In the Local Relational Filter, since angles are used, maybe the vehicle orientation can be used to improve results.

3.4.3 BoT-SORT

Contributions

- [13] builds on top of ByteTrack [11], and combines the advantages of IoU and re-ID, by combining them using a cosine-distance fusion.
- It provides camera motion compensation (CMC) by adopting conventional image registration techniques to correct the Kalman Filter.
- It improves upon the Kalman Filter state vector to obtain better accuracy.

Architecture

While reading the following points, it is recommended to refer to the Figure 3.9:

- The steps of the algorithm are based on ByteTrack, with a few changes.
- Correct the predictions of Kalman Filter, for camera motion compensation (CMC), using image registration techniques from the OpenCV library [14].
- Associate high-score detections and CMC corrected Kalman Filter predictions using a cosine-distance based fusion of IoU and re-ID.
- Associate low-score detections and CMC corrected Kalman Filter predictions using IoU only.

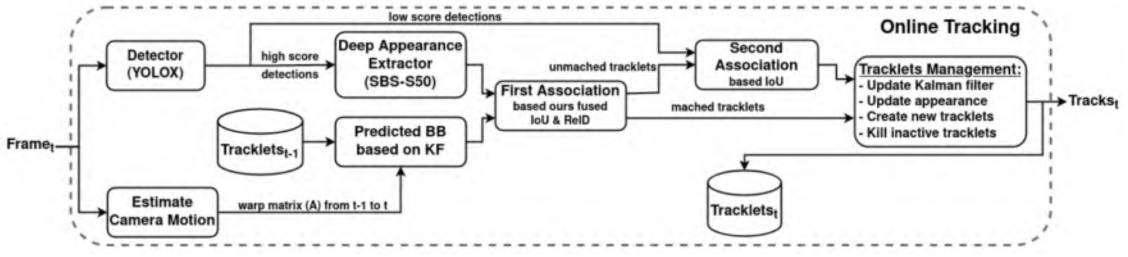


Figure 3.9: Overview of BoT-SORT (image source [13])

Results

- An overview of BoT-SORT's results on popular MOT benchmarks can be found in Table 3.5.

BoT-SORT [13] Results				
Benchmark	**MOTA**	**IDF1**	**HOTA**	**FPS**
MOT17 [3]	80.5	80.2	65.0	4.5
MOT20 [4]	77.8	77.5	63.3	2.4

Table 3.5: An overview of BoT-SORT's results on MOT benchmarks.

Advantages

- **Camera motion compensation**, using OpenCV sparse image registration for background motion estimation.
- **Combines re-ID and IoU** instead of using one of them case-wise.
- **Improved Kalman Filter** state vector (directly estimate height and width of box).
- Proposal of frame-dependent MOTA called Current-MOTA or **cMOTA**, which helps in **better debugging**.

Disadvantages

- **Much slower** at inferencing than ByteTrack, and may not be useful for real-time applications.

Remarks

- Maybe, the camera motion compensation module could be bettered using visual SLAM methods [61].

4 Working on the DLR dataset

4.1 Dataset overview

- The dataset consisted of several image sequences of vehicles that were captured from a helicopter. All images are of a very **high resolution**: **5472 X 3648**.
- We filtered out several sequences, based on FPS (frames per second) and GSD (ground sampling distance, basically the distance between two consecutive pixel centers measured on the ground), to maintain uniformity across the sequences. The dataset had not been previously baselined as to object tracking performance, and it made sense to apply algorithms to a subset of the original dataset that exhibited uniformity in important regards that affect detection and tracking performance.
- GSD affects the size of the objects in the images and FPS affects the perceived motion of the objects. Therefore, it was essential to fix certain ranges for these parameters in order to facilitate meaningful learning and operation by our models.
- Thus, we obtained **7 sequences**, which had a **GSD in the range of 3-6 cm**, and a **frequency in the range of 1-2 FPS**.
- Originally, the dataset boasted of multiple classes of vehicles. However, to baseline it for the first time, we decided to treat the problem as that of a single-class. Therefore, all vehicles would now belong to a **single class called vehicle**.
- An overview of the 7 sequences can be seen in Table 4.1. The number of unique vehicles in the sequence (number of unique tracks), the number of image frames in a sequence, and the total number of object annotations (labelled bounding boxes) for a sequence, have been clearly highlighted.
- The annotations were originally in a **.xml format with oriented bounding boxes**, and had to be brought into the **MS-COCO [8] YOLOv7 [7] format with rectangular bounding boxes**, for the detector and trackers.
- We are using the tracking evaluation metrics from the MOT benchmarks, namely HOTA, MOTA and IDF1. In order to do so, the **Ground Truth annotations** had to be converted to the **MOT benchmark compliant format** as well.
- Figure 4.1 clearly shows the **large camera motion** that is an important characteristic of the sequences in this dataset.

- The large camera motion, the high image resolution, low FPS, and very small objects (high GSD) make this dataset uniquely challenging to work with, for the purpose of vehicle tracking.
- Further details about the DLR dataset, e.g., the annotation procedure, original multiple object classes etc. cannot be provided in this work as there are plans to separately publish the dataset.

Sequence name	**#vehicles**	**#frames**	**#annotations**
2019-10-17-4k_134	73	53	349
2020-06-23-4k-Fahrzeugtracking	37	23	294
2020-06-19-A99-A8-Starnberg_24	26	10	77
2019-10-17-4k_130	77	13	202
2020-03-19-Kieswerke-A96_14	62	13	298
2021-07-21-ATM_43	173	60	5221
2019-10-17-4k_139	136	78	618

Table 4.1: Overview of the DLR dataset.

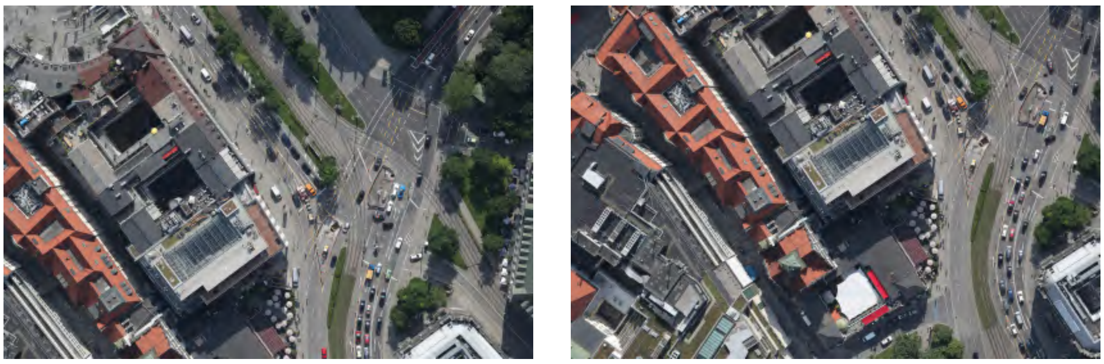


Figure 4.1: Frame 1 and Frame 20 of the **2021-07-21-ATM_43** DLR dataset sequence.

4.2 Detection experiments

We **trained** several **YOLOv7** detection models **from scratch**, using **different image-sizes**, instead of using pretrained YOLOv7 weights (on the MS-COCO dataset), due to the large difference in object size between the MS-COCO dataset and DLR dataset. An overview of the training experiments can be found in Table 4.2. A sample of the ground truth bounding boxes can be found in Figure 4.2. We always focus on three different scenarios in the same frame, namely, normal and clear visual of vehicles, partially occluded vehicles in shadows of trees and motorcycles, to check the robustness of the detection model. The different thresholds used for the experiments are default values used in the official YOLOv7 [7] code, and can be found in the hyperparameter files of the official YOLOv7 repository, e.g., the IoU threshold for training is always 0.2. During inference, the default values of confidence score threshold: 0.25 and IoU threshold: 0.45 are used.

Exp.	GPU	#GPUs	#workers	Time	#epochs	image-size	batch-size	pre-trained	#train images	#val images	#test images
1	NVIDIA TITAN RTX (24GB)	1 (50% used)	8	1.5 hours	100	640 X 640	16	No	231	20	6
2	NVIDIA TITAN RTX (24GB)	1 (90% used)	8	2 hours	100	1280 X 1280	8	No	231	20	6
3	NVIDIA TITAN RTX (24GB)	4 (50% used)	8	3 hours	100	2560 X 2560	4	No	231	20	6
4	NVIDIA TITAN RTX (24GB)	4 (80% used)	8	3.5 hours	100	3200 X 3200	4	No	231	20	6

Table 4.2: Overview of YOLOv7 detector's training experiments on the DLR dataset.

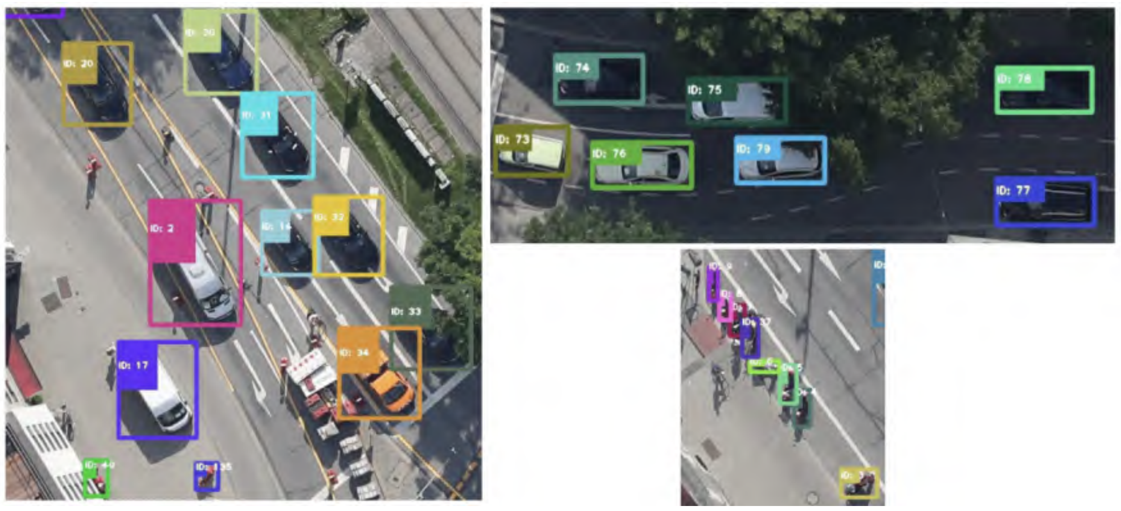


Figure 4.2: Ground truth visualisation of bounding boxes, on an image of the **2021-07-21-ATM_43** sequence.

4.2.1 640 X 640

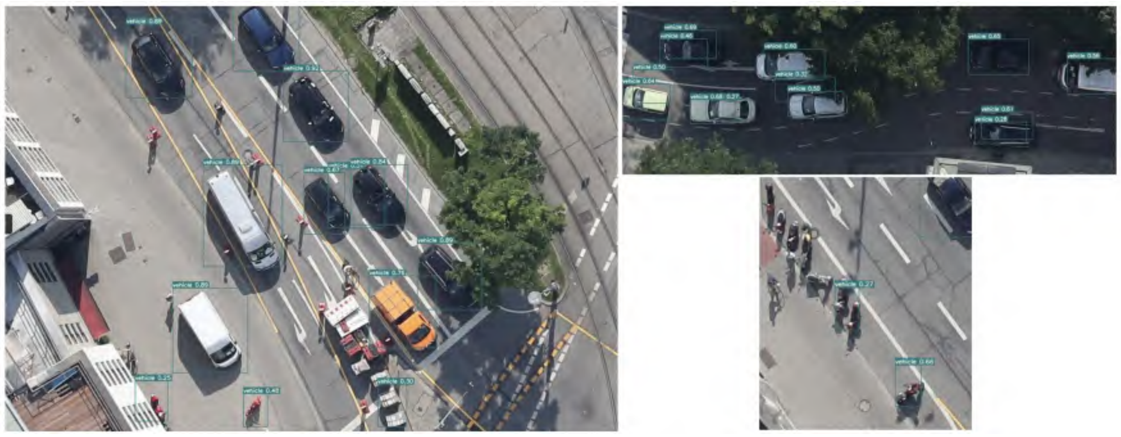


Figure 4.3: YOLOv7 inference result, with 640 X 640 image-size parameter value, on an image of the **2021-07-21-ATM_43** sequence.

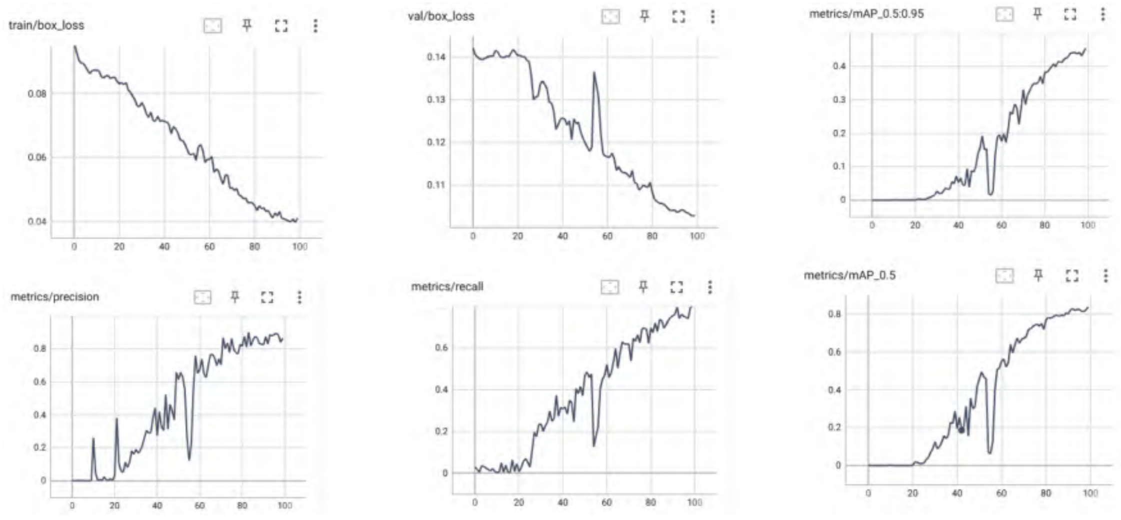


Figure 4.4: YOLOv7 training results, with 640 X 640 image-size parameter value.

Because of the high resolution of the input image, we started with a low image-size of 640 X 640 for training the detector, in order to keep the memory requirements for training in check. The quantitative results of the training can be found in Figure 4.4. The qualitative results obtained by inferencing the trained model on a test image, can be found in Figure 4.3. It can be seen from the inference that there are many false positives because the model tries to resize the image of a very high resolution 5472 X 3648 to 640 X 640 before inferencing, the average confidence score of detections is low at approximately 0.75 and several motor-cycles are not even detected.

4.2.2 1280 X 1280

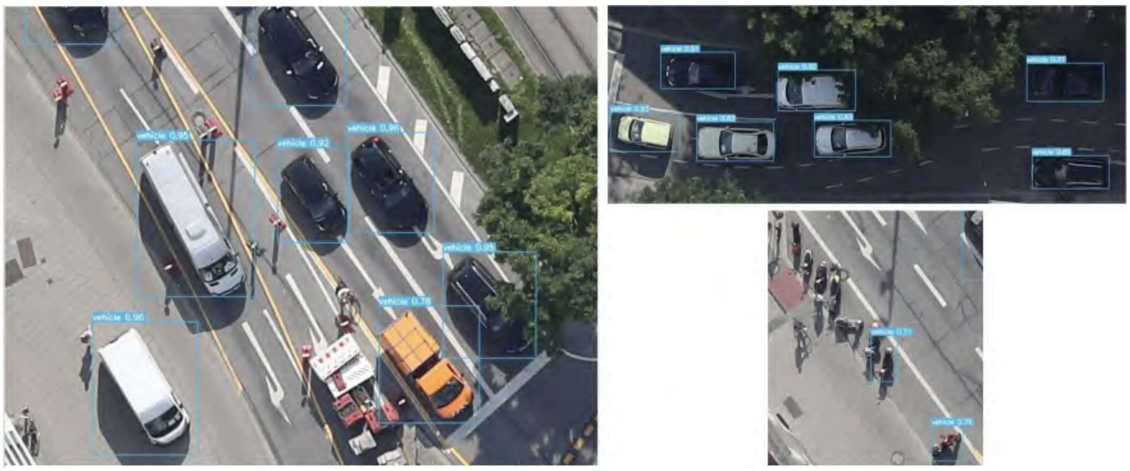


Figure 4.5: YOLOv7 inference result, with 1280 X 1280 image-size parameter value, on an image of the **2021-07-21-ATM_43** sequence.

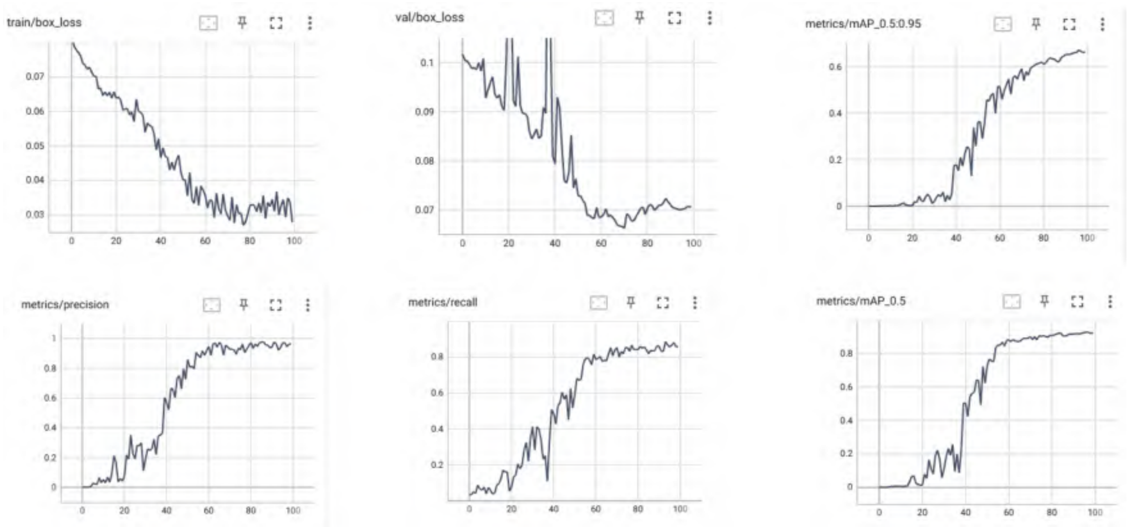


Figure 4.6: YOLOv7 training results, with 1280 X 1280 image-size parameter value.

To overcome the large number of false positives produced by the 640 X 640 model, we trained a new model from scratch with the image-size of 1280 X 1280. The quantitative results of the training can be found in Figure 4.6. The qualitative results obtained by inferencing the trained model on a test image, can be found in Figure 4.5. It can be seen from the inference that the false positives have now been reduced, since the image-resolution we are now working with has increased. However, the average confidence score is approximately 0.85 now which is higher than before but is still low and several motor-cycles are not even detected.

4.2.3 2560 X 2560

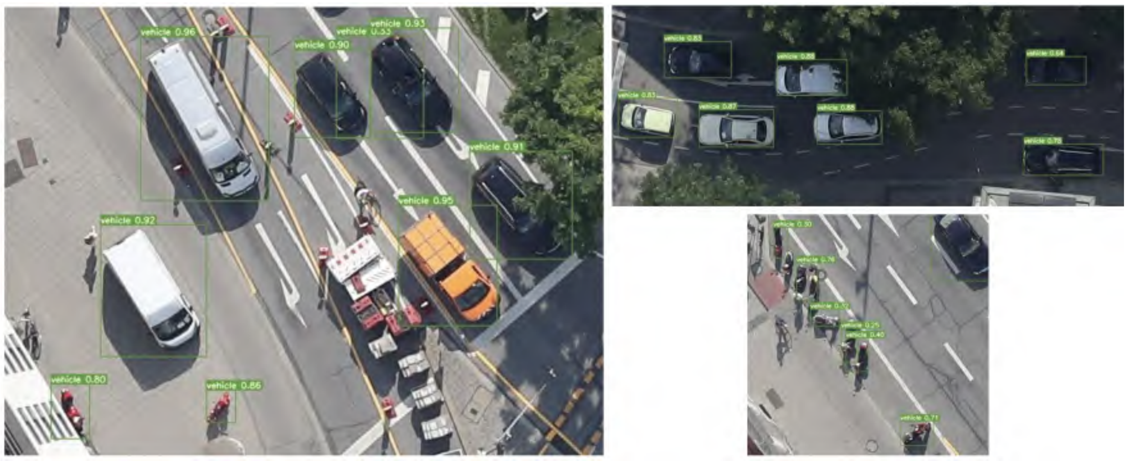


Figure 4.7: YOLOv7 inference result, with 2560 X 2560 image-size parameter value, on an image of the **2021-07-21-ATM_43** sequence.

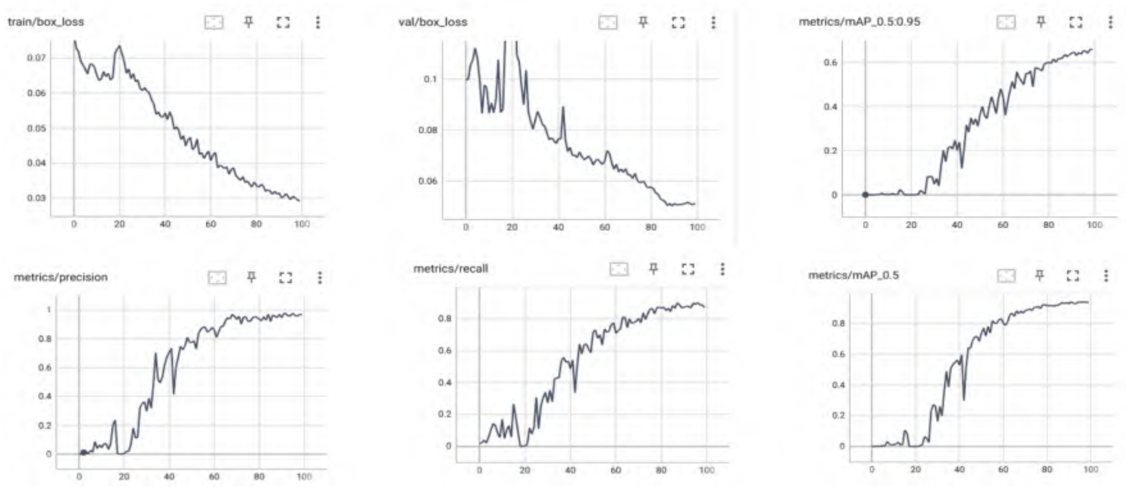


Figure 4.8: YOLOv7 training results, with 2560 X 2560 image-size parameter value.

In order to overcome the limitation of the 1280 X 1280 model, so as to detect smaller objects like motor-cycles, we trained a new model from scratch with the image-size of 2560 X 2560. The quantitative results of the training can be found in Figure 4.8. The qualitative results obtained by inferencing the trained model on a test image, can be found in Figure 4.7. It can be seen from the inference that many more motor-cycles are now detected (6 instead of 2 before) and the average confidence score of detections has also increased to approximately 0.9. However, the confidence score of detected motor-cycles is still low (approximately 0.5) and a few motor-cycles are still not detected.

4.2.4 3200 X 3200

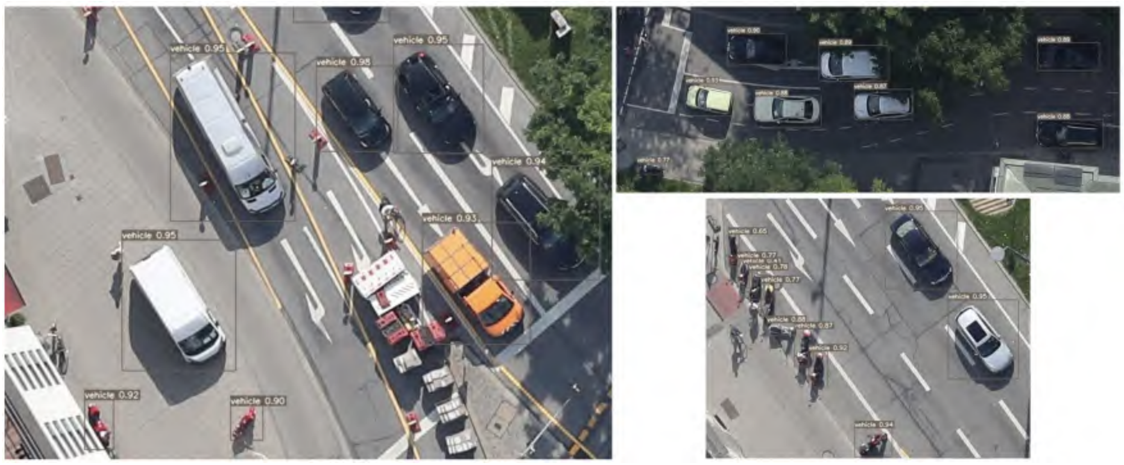


Figure 4.9: YOLOv7 inference result, with 3200 X 3200 image-size parameter value, on an image of the **2021-07-21-ATM_43** sequence.

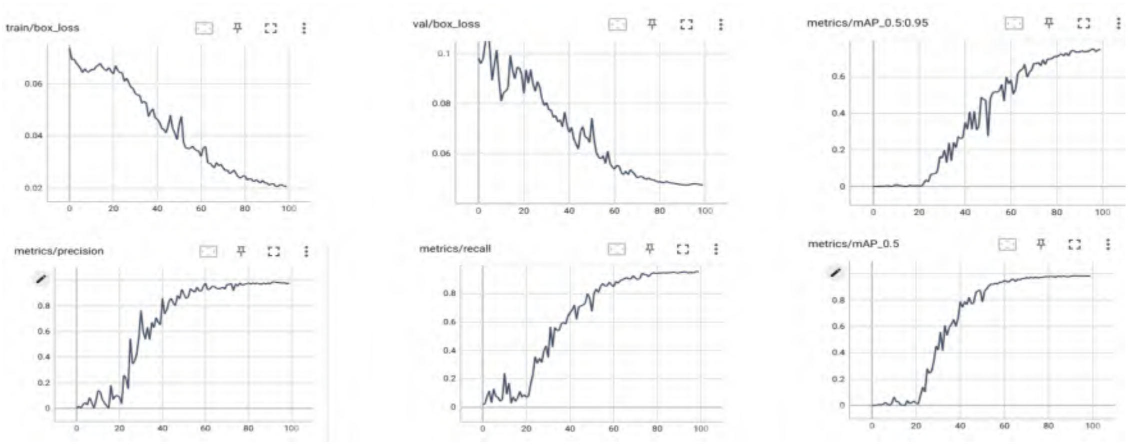


Figure 4.10: YOLOv7 training results, with 3200 X 3200 image-size parameter value.

To overcome the limitation of the 2560 X 2560 model, whereby the confidence score of detecting motor-cycles was low and to detect all of the motor-cycles, we trained a new model from scratch with the image-size of 3200 X 3200. The quantitative results of the training can be found in Figure 4.10. The qualitative results obtained by inferencing the trained model on a test image, can be found in Figure 4.9. It can be seen from the inference that not only all motor-cycles (10 of them) are now detected but also the average confidence score of detecting motor-cycles has increased significantly to approximately 0.8. The average confidence score of all detections has now attained a high value of approximately 0.95. Due to GPU memory restrictions, we did not train any further detection models.

4.3 Tracking experiments

In this section, we would like to present the tracking results of several algorithms, namely, ByteTrack [11], UAVMOT [59], DeepSORT [12], BoT-SORT [13] and our proposed Byte-De-SORT, on all the sequences of the DLR dataset. For all tracking experiments, we used NVIDIA TITAN RTX (24 GB) GPU and we used the same YOLOv7 detector, which was trained from scratch on the DLR dataset with an image-size of 3200 X 3200, since that gave us the best detection performance. For all qualitative results, we chose non-consecutive frames 1, 15 and 25 out of the 60 frames to see a tracker's performance over long-term tracking. If the track-id and color of the bounding box are maintained, it means that the object was tracked correctly. We do not display track history, because that is already indicated by the track-id and bounding box colour, and we focus on specific parts of the frame and not the entire frame over a long period of time (non-consecutive frames).

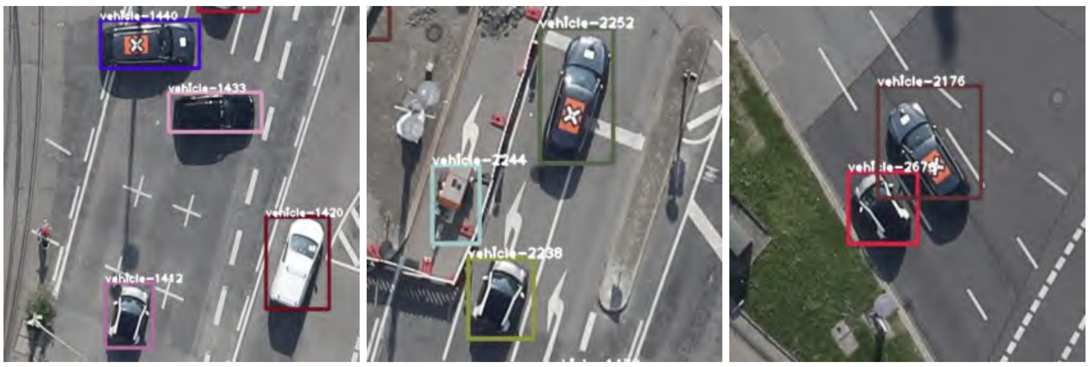


Figure 4.11: ByteTrack's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence.

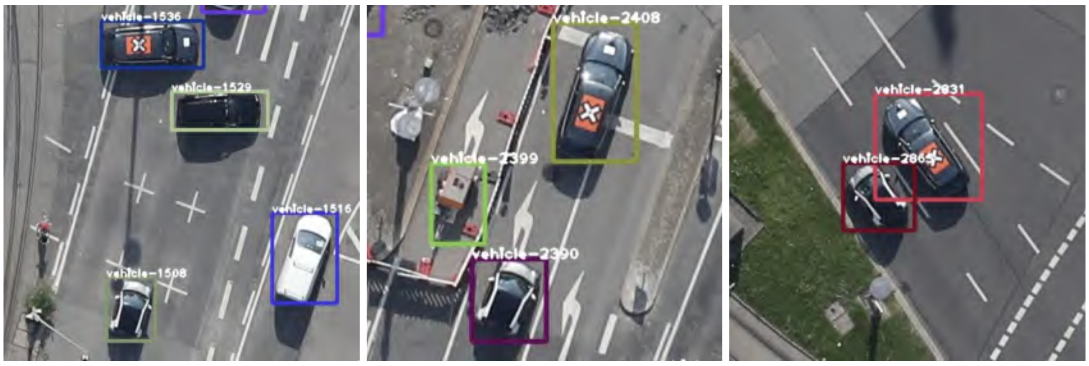


Figure 4.12: UAVMOT's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence.

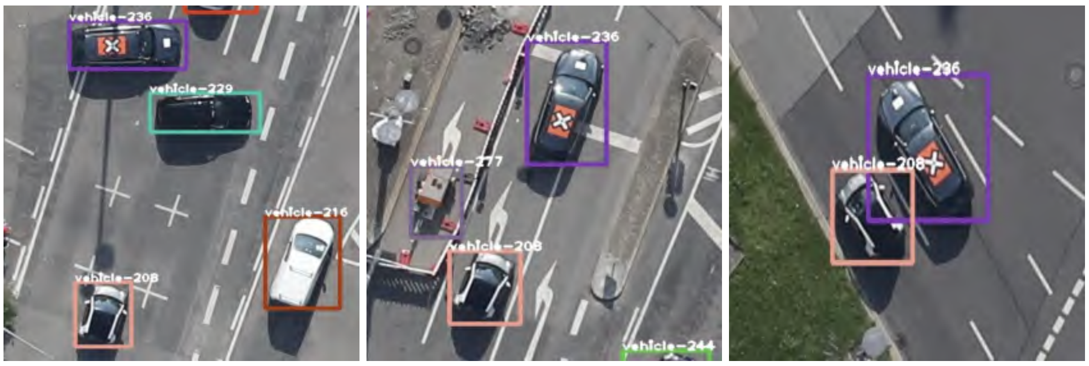


Figure 4.13: DeepSORT's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence.

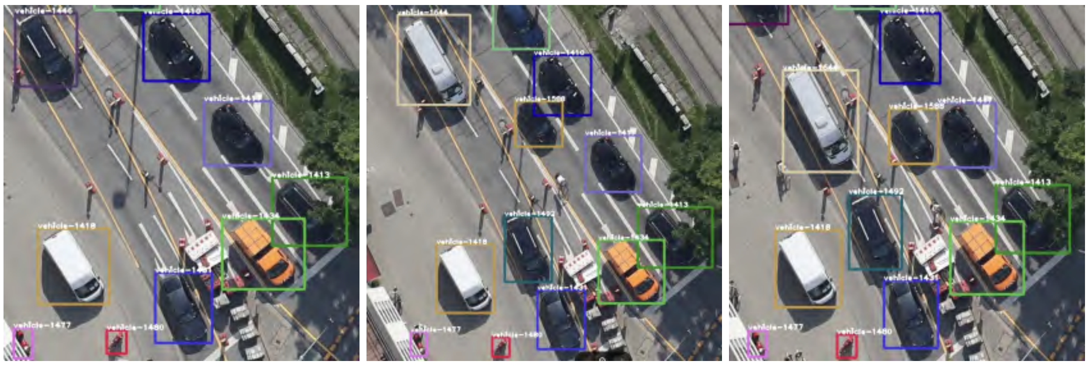


Figure 4.14: BoT-SORT's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence.

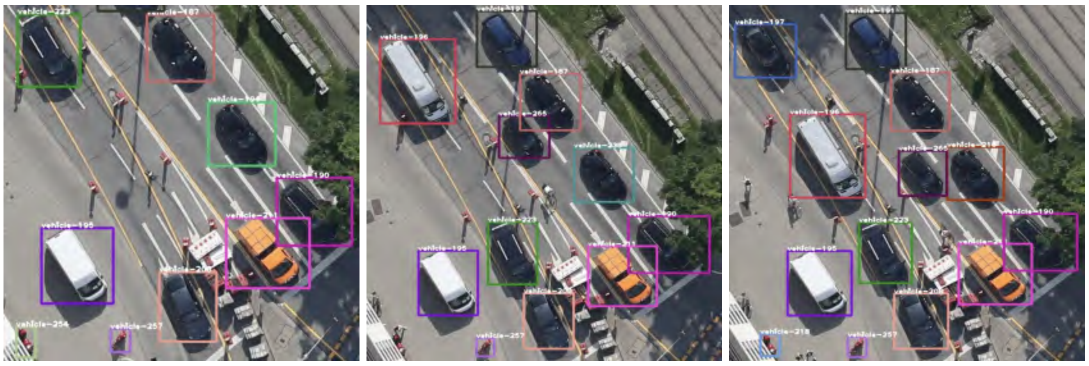


Figure 4.15: Byte-De-SORT's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence.

- **ByteTrack**: ByteTrack is one of the SOTA algorithms, and is often used as a basis for other SOTA algorithms as well. The highlight of the ByteTrack algorithm, which is taking into account the low score detections (confidence score typically between 0.15 and 0.5), can prove really useful for the DLR dataset with small objects. The quantitative performance of ByteTrack on the DLR dataset can be found in Table 4.3. **Average Inference Speed: 3.16 FPS**. The qualitative performance of ByteTrack is well represented by Figure 4.11. Because of large camera motion and low FPS data, an IoU based method like ByteTrack fails to be effective on the DLR dataset. The Kalman Filter is unable to predict the motion of the vehicles accurately, because of lack of camera motion compensation and low FPS data.

- **UAVMOT**: It is a SOTA method, that was specifically designed to work with aerial imagery and data captured from UAV (unmanned aerial vehicle). It has a Local Relational Filter, that tries to deal with camera motion in case the Kalman Filter fails to provide a significant number of matches. The quantitative performance of UAVMOT on the DLR dataset can be found in Table 4.4. **Average Inference Speed: 2.85 FPS**. The qualitative performance of UAVMOT is well represented by Figure 4.12. While, the method is known to be amongst SOTA for image sequences captured from a drone, it is probably not tailored for images captured from a helicopter that contain much smaller objects. Like ByteTrack, IoU based data association and Kalman Filter fail to work well. The Local Relational Filter also fails to work since the data is very low FPS and therefore the assumption, that the angle formed between the nearest and farthest vehicle in a local neighbourhood remains constant, does not hold good between two consecutive frames.

- **DeepSORT (without Kalman Filter)**: IoU based data association was not working and therefore the next choice was to check appearance based data association. Kalman Filter was not able to predict object motion because of large camera motion and low FPS data, and therefore, we removed it to verify the importance of appearance features alone in this dataset. The quantitative performance of DeepSORT (without Kalman Filter) on the DLR dataset can be found in Table 4.5. **Average Inference Speed: 2.92 FPS**. The qualitative performance of DeepSORT is well represented by Figure 4.13. Results conclusively indicate that removal of IoU based data association and inclusion of appearance features based data association is conducive to good tracking performance on this dataset. We can also conclude that if camera motion is not compensated, it is not useful to use Kalman Filter while performing tracking on this dataset, since we initially tried the original DeepSORT (with Kalman Filter) and it did not give good results and therefore, we removed the Kalman Filter and obtained fresh results to present here.

- **BoT-SORT**: BoT-SORT is based on ByteTrack, and is also one of the SOTA methods in tracking. On top of IoU based data association, BoT-SORT offers camera motion compensation, through image registration techniques like SIFT [62]. Therefore, the large camera motion which is an important characteristic of this dataset can now be dealt with appropriately. The quantitative performance of BoT-SORT on the DLR dataset can be found in Table 4.6. **Average Inference Speed: 0.01 FPS**. The qualitative performance of BoT-SORT is well represented by Figure 4.14. Because of camera motion compensation, both IoU based data association and Kalman Filter are now working to a great extent. However, as it can be seen in Figure 4.14, the effects of low FPS data have not been conquered. For example, a fast moving object, the vehicle-1446 (the black vehicle on top left) in Frame 1 is tracked as vehicle-1492 in Frame 15. This shows that even after camera motion compensation, IoU and Kalman Filter do not work well on fast moving objects in low FPS data. Camera motion compensation, using image registration techniques like SIFT, is very computationally expensive and therefore, BoT-SORT is much slower at inferencing than any other method presented here, making it unsuitable for real-time tracking.

- **Byte-De-SORT**: Our proposed method Byte-De-SORT is specifically tailored to work on this dataset, since it is based on data association using appearance features and not IoU. Byte-De-SORT does not use a Kalman Filter which has been shown to be not effective on this dataset. While camera motion compensation using SIFT has been shown to be effective on the dataset via BoT-SORT, it was computationally expensive and to keep the method relevant for real-time tracking, we have not included camera motion compensation in Byte-De-SORT. The quantitative performance of Byte-De-SORT on the DLR dataset can be found in Table 4.7. **Average Inference Speed: 2.82 FPS**. The qualitative performance of Byte-De-SORT is well represented by Figure 4.15. As can be seen in Figure 4.15, the fast moving object vehicle-223 (on top left of Frame 1), is now correctly tracked (unlike BoT-SORT). However, tracking of occluded objects suffers in this method since it depends on appearance features. For example, in Figure 4.15, the motor-cycle (vehicle-254, on bottom left of Frame 1) is not correctly tracked. Extremely similar looking vehicles suffer from ID-switches during tracking, because of dependence on appearance features. For example, in Figure 4.15, vehicle-194 (in middle-right) in Frame 1 is tracked as vehicle-238 in Frame 15 and is further tracked as vehicle-216 in Frame 25.

Sequence name	**HOTA**	**MOTA**	**IDF1**
2019-10-17-4k_130	0.41	0.23	0.34
2019-10-17-4k_134	0.34	0.14	0.22
2019-10-17-4k_139	0.37	0.26	0.27
2020-03-19-Kieswerke-A96_14	0.32	0.12	0.21
2020-06-19-A99-A8-Starnberg_24	0.43	0.23	0.32
2020-06-23-4k-Fahrzeugtracking	0.27	0.09	0.15
2021-07-21-ATM_43	0.15	0.05	0.07
COMBINED	0.22	0.08	0.11

Table 4.3: Quantitative results of ByteTrack on DLR dataset.

Sequence name	**HOTA**	**MOTA**	**IDF1**
2019-10-17-4k_130	0.44	0.23	0.36
2019-10-17-4k_134	0.35	0.13	0.22
2019-10-17-4k_139	0.37	0.23	0.25
2020-03-19-Kieswerke-A96_14	0.33	0.12	0.21
2020-06-19-A99-A8-Starnberg_24	0.43	0.23	0.32
2020-06-23-4k-Fahrzeugtracking	0.28	0.10	0.15
2021-07-21-ATM_43	0.15	0.06	0.06
COMBINED	0.23	0.09	0.10

Table 4.4: Quantitative results of UAVMOT on DLR dataset.

Sequence name	**HOTA**	**MOTA**	**IDF1**
2019-10-17-4k_130	0.38	0.35	0.42
2019-10-17-4k_134	0.33	0.40	0.32
2019-10-17-4k_139	0.39	0.35	0.40
2020-03-19-Kieswerke-A96_14	0.39	0.41	0.46
2020-06-19-A99-A8-Starnberg_24	0.37	0.26	0.45
2020-06-23-4k-Fahrzeugtracking	0.35	0.42	0.40
2021-07-21-ATM_43	0.60	0.73	0.73
COMBINED	0.54	0.64	0.64

Table 4.5: Quantitative results of DeepSORT (without Kalman Filter) on DLR dataset.

Sequence name	**HOTA**	**MOTA**	**IDF1**
2019-10-17-4k_130	0.43	0.27	0.40
2019-10-17-4k_134	0.37	0.12	0.24
2019-10-17-4k_139	0.45	0.22	0.37
2020-03-19-Kieswerke-A96_14	0.43	0.21	0.40
2020-06-19-A99-A8-Starnberg_24	0.42	0.23	0.32
2020-06-23-4k-Fahrzeugtracking	0.29	0.11	0.16
2021-07-21-ATM_43	0.64	0.73	0.71
COMBINED	0.59	0.59	0.61

Table 4.6: Quantitative results of BoT-SORT on DLR dataset.

Sequence name	**HOTA**	**MOTA**	**IDF1**
2019-10-17-4k_130	0.40	0.50	0.37
2019-10-17-4k_134	0.38	0.57	0.39
2019-10-17-4k_139	0.36	0.56	0.34
2020-03-19-Kieswerke-A96_14	0.35	0.49	0.39
2020-06-19-A99-A8-Starnberg_24	0.37	0.40	0.41
2020-06-23-4k-Fahrzeugtracking	0.39	0.56	0.43
2021-07-21-ATM_43	0.62	0.82	0.72
COMBINED	0.56	0.75	0.63

Table 4.7: Quantitative results of Byte-De-SORT on DLR dataset.

4.4 Conclusion

$$Camera\ Motion\ Compensation \approx Appearance\ Features >> Kalman\ Filter \approx IoU \quad (4.1)$$

BoT-SORT achieved the highest overall HOTA score of 0.59, closely followed by our proposed method Byte-De-SORT which achieved an overall HOTA score of 0.56. Byte-De-SORT also achieved an overall MOTA score of 0.75, which was the highest by a large margin. DeepSORT (without Kalman Filter) achieved the highest overall IDF1 score of 0.64, followed very closely by Byte-De-SORT which achieved an overall IDF1 score of 0.63. Byte-De-SORT clocked an average inference speed of 2.82 FPS, making it much more suitable for real-time tracking than BoT-SORT which showed an average inference speed of only 0.01 FPS. Byte-De-SORT is also able to effectively track fast-moving vehicles, which proved difficult for most trackers using Kalman Filter on this low FPS DLR dataset. For the DLR dataset, we can safely say that the order of importance of different features of a tracking algorithm can be given by the Equation 4.1. Therefore, as per observations, **Byte-De-SORT stands out as the best solution to be used for real-time tracking on the DLR dataset**.

5 Working on the A9 dataset

5.1 Dataset overview

- This is the second version of the A9 dataset [6]. The first version was called **r00** and this version is called **r01**. As of writing this thesis, **r01** has not been completely published. Only 3 sequences of **r01**, namely r01_s01, r01_s02 and r01_s03 have been published so far, which focus on the highway. The sequences from the intersections will be published in the near future. The dataset, in total, consists of 9 sequences ranging from **r01_s01** to **r01_s09**.
- Each of these sequences are captured simultaneously from multiple cameras. For example, **r01_s01** has been captured by 4 different cameras. Although the inputs of 4 different cameras can be fused using timestamps of capture, for our purposes, we treat them as 4 different sequences, each with its own set of unique objects. In the future there are plans to fuse the data such that there will be unique objects per sequence and not per camera view.
- The cameras that have been used to construct the dataset are hosted on traffic monitoring systems on the A9 traffic highway and intersections in Garching, a little north of Munich, Germany. There is **zero camera motion** (unless the weather is very windy and it ends up moving the camera) in this dataset.
- All images are of a **medium resolution: 1920 X 1200**. All sequences are of **medium-high frequency: 10 FPS**.
- The annotations were originally in **OpenLABEL [63] JSON format**, and had to be brought into the **MS-COCO [8] YOLOv7 [7] format**, for the detector and trackers.
- We are using the tracking evaluation metrics from the MOT benchmarks, namely HOTA, MOTA and IDF1. In order to do so, the **Ground Truth annotations** had to be converted to the **MOT benchmark compliant format** as well.
- There are **11 classes of objects** in this dataset, as can be seen in Table 5.1. Please always refer to this Table for a mapping from object class name to object class id, since the class names are not used in other Tables and only class id is used. This is because of the large number of columns in these Tables which make it difficult to fit large column names.
- We also provide a detailed overview of the dataset across Tables 5.2 - 5.10. For example, Table 5.2 shows that the sequence r01_s01_s040_north_50mm contains a total

of 86 unique objects of the CAR (object class id 0 as per Table5.1) type, across all frames. The Table 5.2 also informs us that the sequence r01_s01_s040_north_50mm has a total of 354 image frames and that it has a total of 27428 labelled bounding boxes (annotations) across all frames including all object types and instances.

CAR	TRUCK	TRAILER	VAN	MOTORCYCLE	BUS	PEDESTRIAN	BICYCLE	EMERGENCY_VEHICLE	OTHER	LICENSE_PLATE_LOCATION
0	1	2	3	4	5	6	7	8	9	10

Table 5.1: Classes of objects in the A9 dataset. Mapping of object class name to object class id.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s01_s040_north_50mm	86	28	135	19	22	0	0	0	0	0	1	27428	354
r01_s01_s040_north_16mm	56	25	81	19	15	0	0	0	0	0	0	14018	354
r01_s01_s050_south_16mm	65	26	91	18	20	0	0	0	0	0	0	15335	375
r01_s01_s050_south_50mm	74	27	113	14	25	0	0	0	0	0	0	28723	375

Table 5.2: Overview of the **r01_s01** sequence of the A9 dataset.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s02_s040_north_50mm	100	16	0	10	18	0	0	0	0	0	0	11338	375
r01_s02_s040_north_16mm	80	16	0	10	17	0	0	0	0	0	0	7055	375
r01_s02_s050_south_16mm	60	17	0	13	19	0	0	0	0	0	0	5790	358
r01_s02_s050_south_50mm	79	15	0	10	18	0	0	0	0	0	0	10312	358

Table 5.3: Overview of the **r01_s02** sequence of the A9 dataset.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s03_s040_north_50mm	164	55	245	35	45	0	0	0	0	0	3	62682	750
r01_s03_s040_north_16mm	98	39	154	42	25	0	0	0	0	0	4	26221	750
r01_s03_s050_south_16mm	105	34	162	26	32	0	0	6	0	0	3	35094	750
r01_s03_s050_south_50mm	146	41	230	24	37	0	0	1	0	0	3	61640	750

Table 5.4: Overview of the **r01_s03** sequence of the A9 dataset.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s04_s110_east_50mm	21	1	0	0	4	0	0	0	3	0	0	2963	300
r01_s04_s110_east_16mm	34	1	0	0	1	0	0	2	1	0	0	5268	300
r01_s04_s110_north_50mm	12	1	0	2	5	0	0	0	0	0	0	2215	300
r01_s04_s110_south2_8mm	19	2	0	1	3	0	0	3	0	0	0	4698	300
r01_s04_m090_west_50mm	44	3	0	1	5	0	0	4	0	0	0	6611	300
r01_s04_s110_south1_8mm	18	3	0	1	4	0	0	3	0	0	0	4152	300
r01_s04_s110_north_16mm	7	0	0	0	2	0	0	2	0	0	0	2094	300

Table 5.5: Overview of the **r01_s04** sequence of the A9 dataset.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s05_s110_east_50mm	32	2	0	2	3	0	0	0	0	0	0	4336	300
r01_s05_s110_east_16mm	53	3	0	2	2	0	0	6	0	0	0	7169	300
r01_s05_s110_north_50mm	13	1	0	2	1	0	0	0	0	0	0	2883	300
r01_s05_s110_south2_8mm	28	1	0	1	3	0	0	2	0	0	0	4449	300
r01_s05_m090_west_50mm	55	4	0	3	2	0	0	4	0	0	0	8471	300
r01_s05_s110_south1_8mm	27	3	0	2	1	0	0	2	1	0	0	4713	300
r01_s05_s110_north_16mm	7	0	0	0	1	0	0	0	0	0	0	1043	300

Table 5.6: Overview of the **r01_s05** sequence of the A9 dataset.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s06_s040_north_50mm	127	12	0	12	24	0	1	0	0	0	0	16399	300
r01_s06_s040_north_16mm	84	9	0	7	17	0	0	0	0	0	0	8609	300
r01_s06_s050_south_16mm	83	9	0	7	16	0	0	0	0	0	0	3439	126
r01_s06_s050_south_50mm	138	14	0	8	27	0	0	0	0	0	0	18493	300

Table 5.7: Overview of the **r01_s06** sequence of the A9 dataset.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s07_s040_north_50mm	119	14	0	7	12	0	0	6	0	3	0	16625	300
r01_s07_s040_north_16mm	96	5	0	4	11	0	1	2	0	5	0	10481	300
r01_s07_s050_south_16mm	71	4	0	4	7	0	0	0	0	0	0	2584	131
r01_s07_s050_south_50mm	119	17	0	10	22	0	0	12	0	5	0	19726	300

Table 5.8: Overview of the **r01_s07** sequence of the A9 dataset.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s08_s110_east_16mm	108	8	0	6	12	2	0	1	2	0	0	26616	1200
r01_s08_s110_north_50mm	61	9	0	7	14	3	1	15	5	0	0	23084	1200
r01_s08_m090_north_16mm	70	2	0	2	7	0	2	1	2	0	0	14770	1201
r01_s08_s110_south2_8mm	86	9	0	11	14	2	4	15	2	0	0	22527	1200
r01_s08_m090_west_50mm	182	17	0	16	19	5	2	7	6	0	0	33826	1200
r01_s08_m090_east_50mm	95	8	0	7	21	1	0	0	0	0	0	8516	1200
r01_s08_s110_south1_8mm	74	13	0	9	9	2	2	10	3	0	0	15190	1200
r01_s08_s110_north_16mm	48	7	0	5	6	2	0	4	1	0	0	9983	1200
r01_s08_m090_west_16mm	151	6	0	6	12	2	0	1	0	0	0	36090	1201

Table 5.9: Overview of the **r01_s08** sequence of the A9 dataset.

Sequence name	#0	#1	#10	#2	#3	#4	#5	#6	#7	#8	#9	#annotations	#frames
r01_s09_s110_east_16mm	8	1	0	0	1	0	1	0	0	0	0	1152	618
r01_s09_s110_north_50mm	6	0	0	0	1	0	0	0	0	0	0	1211	619
r01_s09_m090_north_16mm	6	0	0	0	1	0	0	0	0	0	0	1286	606
r01_s09_s110_south2_8mm	10	4	0	2	2	0	2	0	0	0	0	4947	620
r01_s09_m090_west_50mm	11	1	0	0	1	0	1	0	0	0	0	1401	606
r01_s09_m090_east_50mm	9	1	0	1	0	0	2	0	0	0	0	1185	602
r01_s09_s110_south1_8mm	7	1	0	1	1	0	1	0	0	0	0	1815	620
r01_s09_s110_north_16mm	2	0	0	0	0	0	0	0	0	0	0	723	619
r01_s09_m090_west_16mm	10	1	0	2	1	0	1	0	0	0	0	1301	605

Table 5.10: Overview of the **r01_s09** sequence of the A9 dataset.

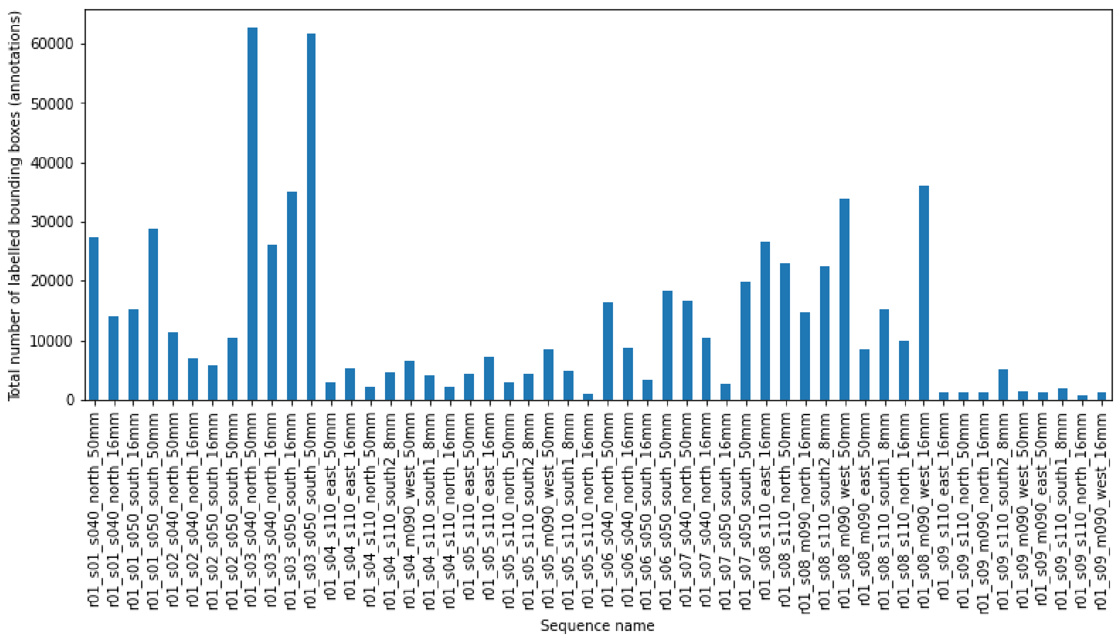


Figure 5.1: A distribution of the total number of labelled bounding boxes (annotations) across the A9 dataset.

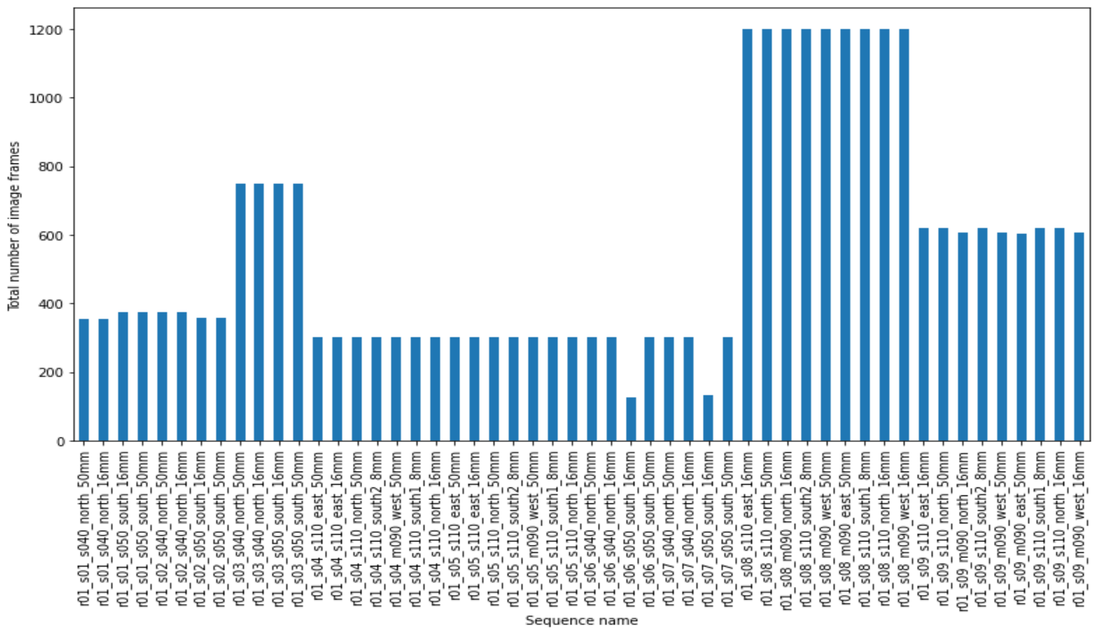


Figure 5.2: A distribution of the total number of image frames across the A9 dataset.

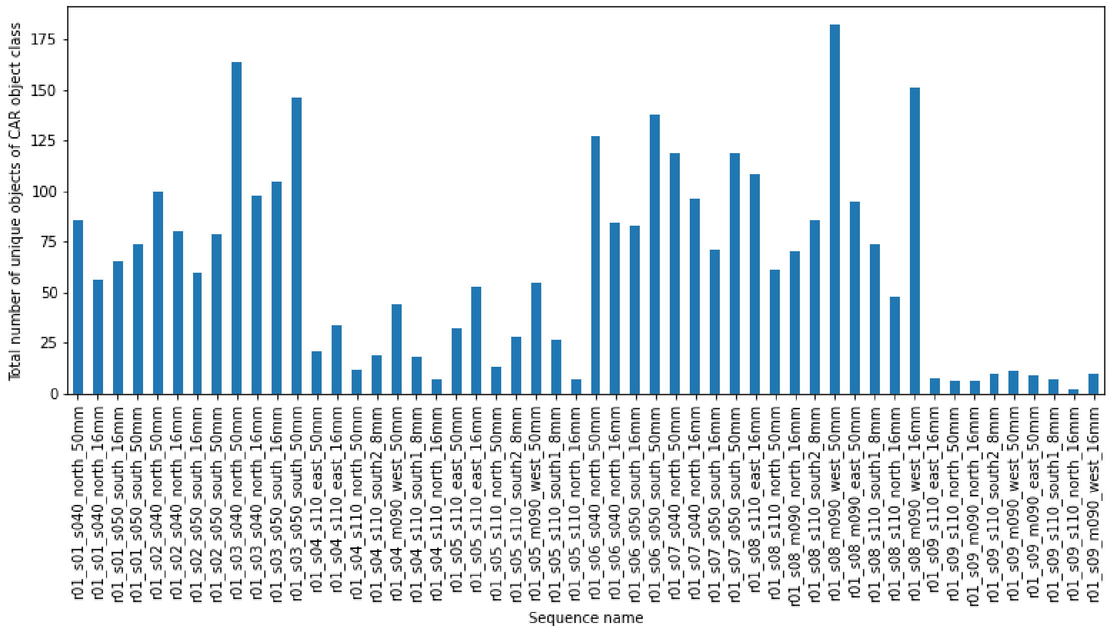


Figure 5.3: A distribution of the total number of unique objects of the CAR object class across the A9 dataset.

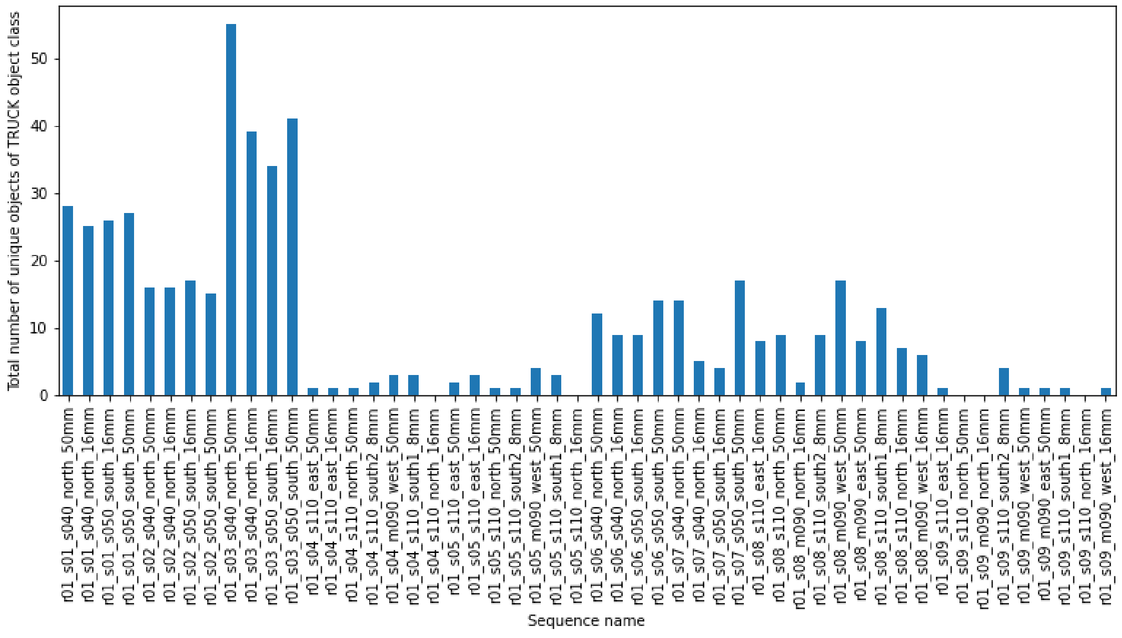


Figure 5.4: A distribution of the total number of unique objects of the TRUCK object class across the A9 dataset.

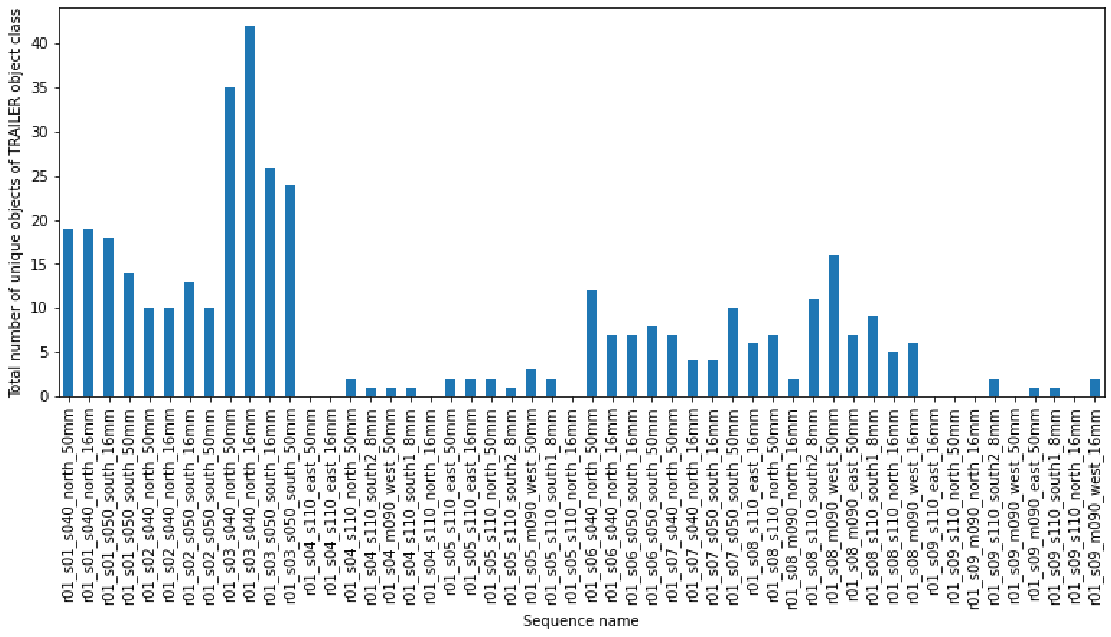


Figure 5.5: A distribution of the total number of unique objects of the TRAILER object class across the A9 dataset.

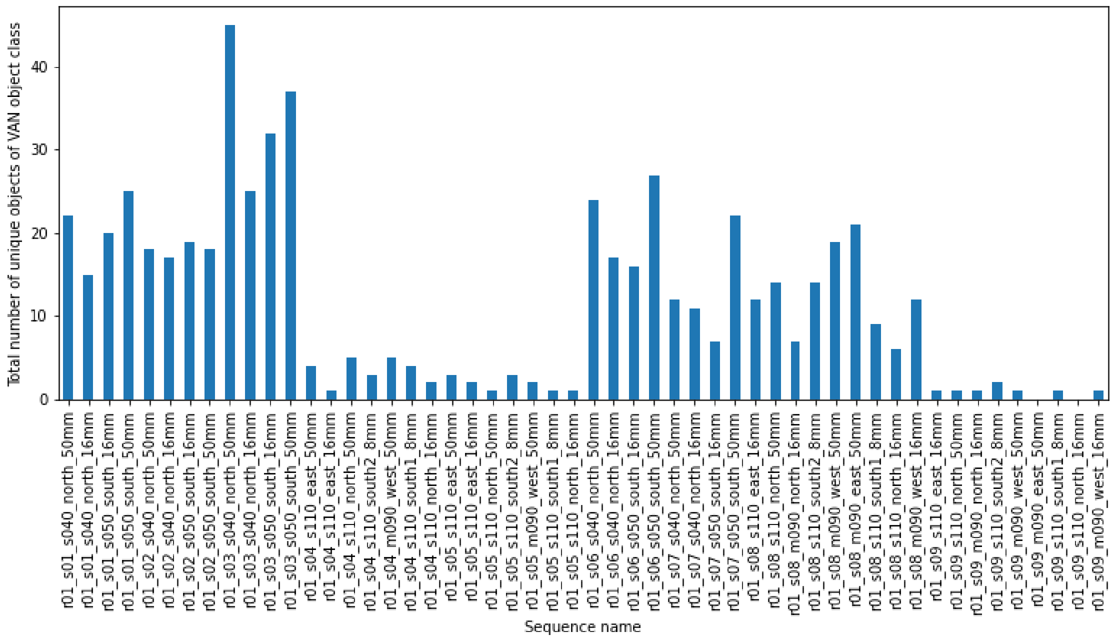


Figure 5.6: A distribution of the total number of unique objects of the VAN object class across the A9 dataset.

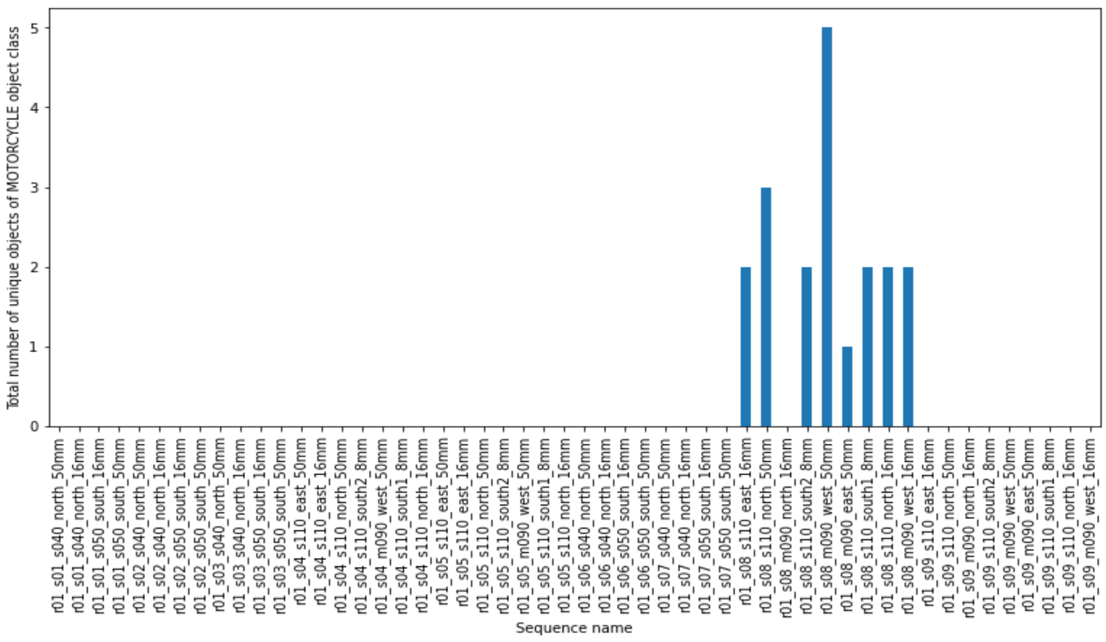


Figure 5.7: A distribution of the total number of unique objects of the MOTORCYCLE object class across the A9 dataset.

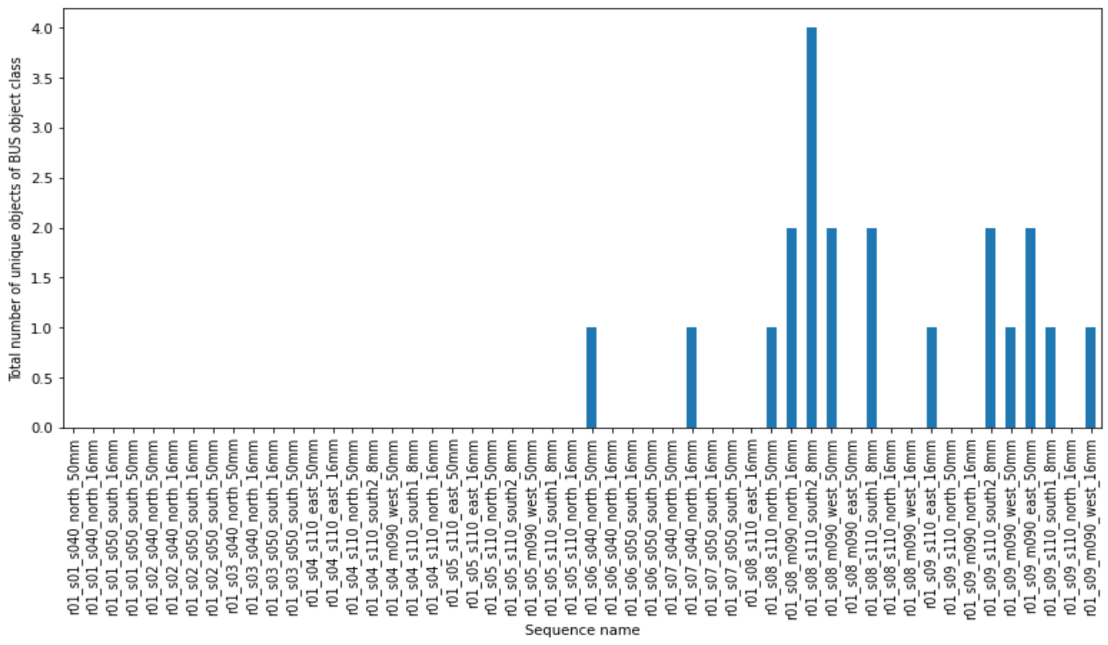


Figure 5.8: A distribution of the total number of unique objects of the BUS object class across the A9 dataset.

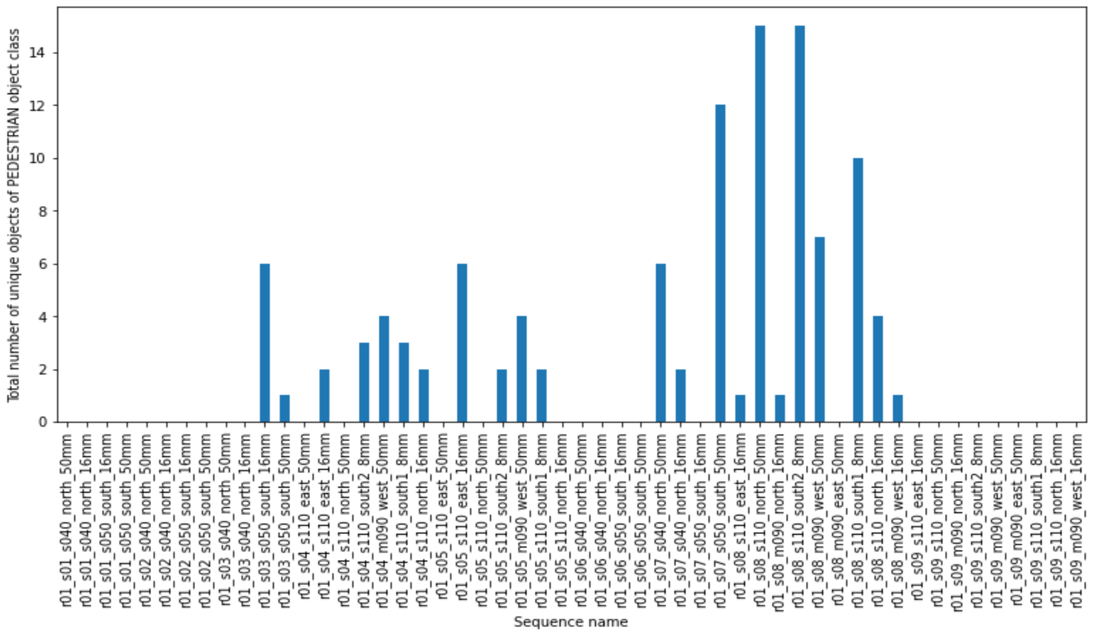


Figure 5.9: A distribution of the total number of unique objects of the PEDESTRIAN object class across the A9 dataset.

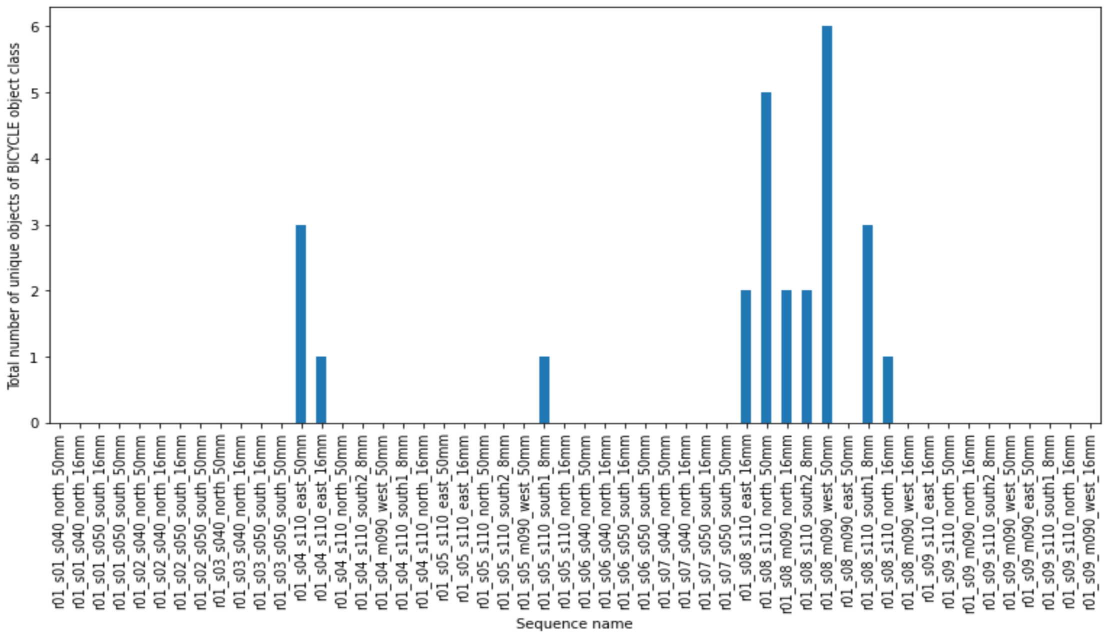


Figure 5.10: A distribution of the total number of unique objects of the BICYCLE object class across the A9 dataset.

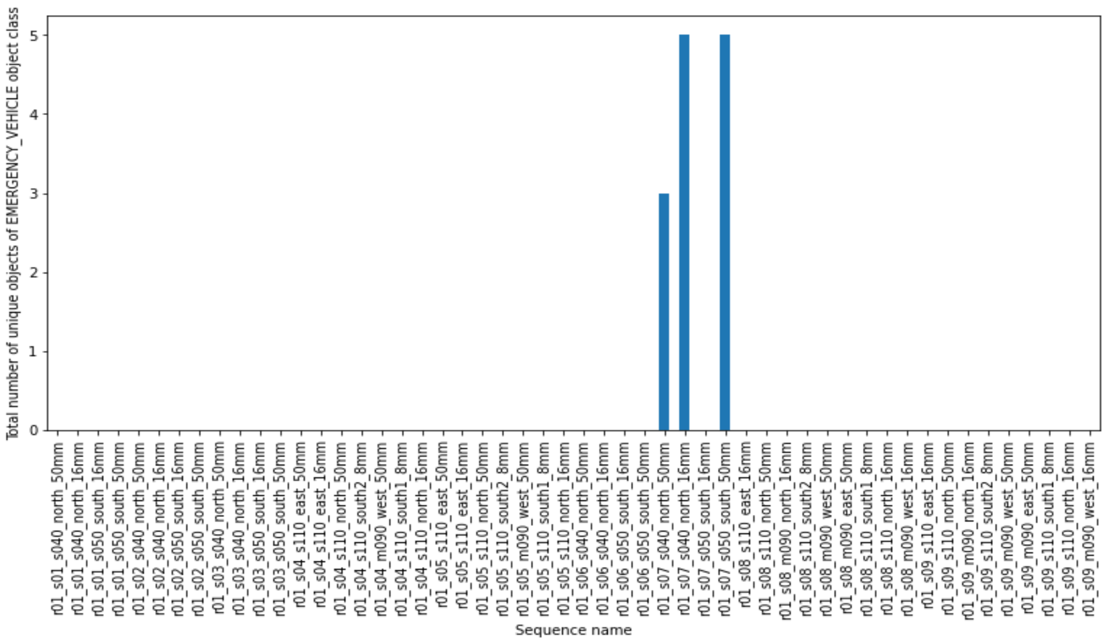


Figure 5.11: A distribution of the total number of unique objects of the EMERGENCY_VEHICLE object class across the A9 dataset.

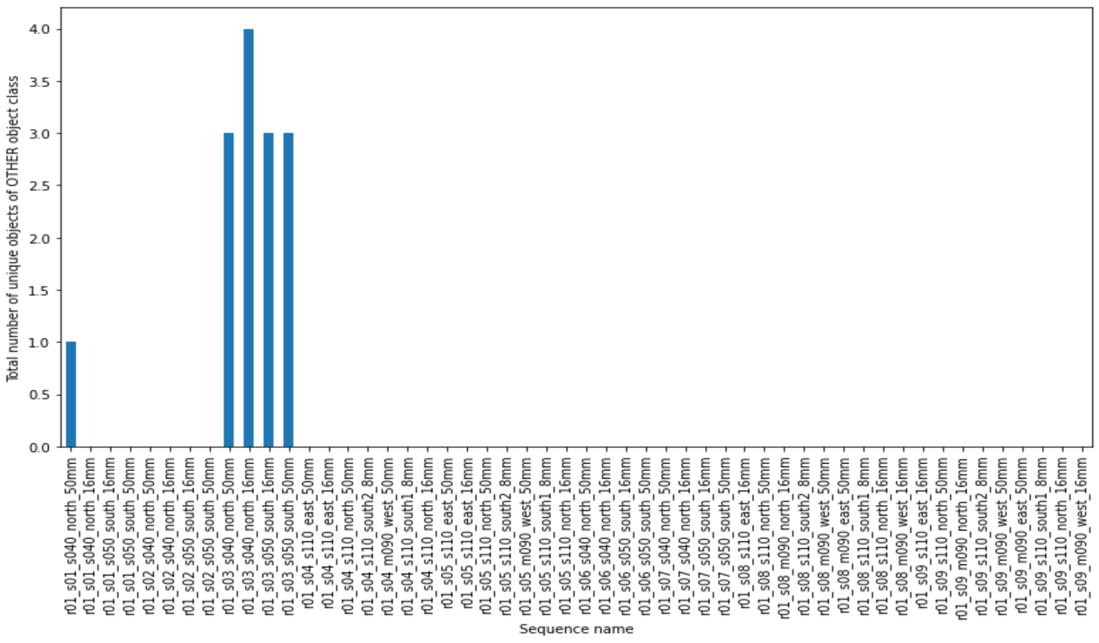


Figure 5.12: A distribution of the total number of unique objects of the OTHER object class across the A9 dataset.

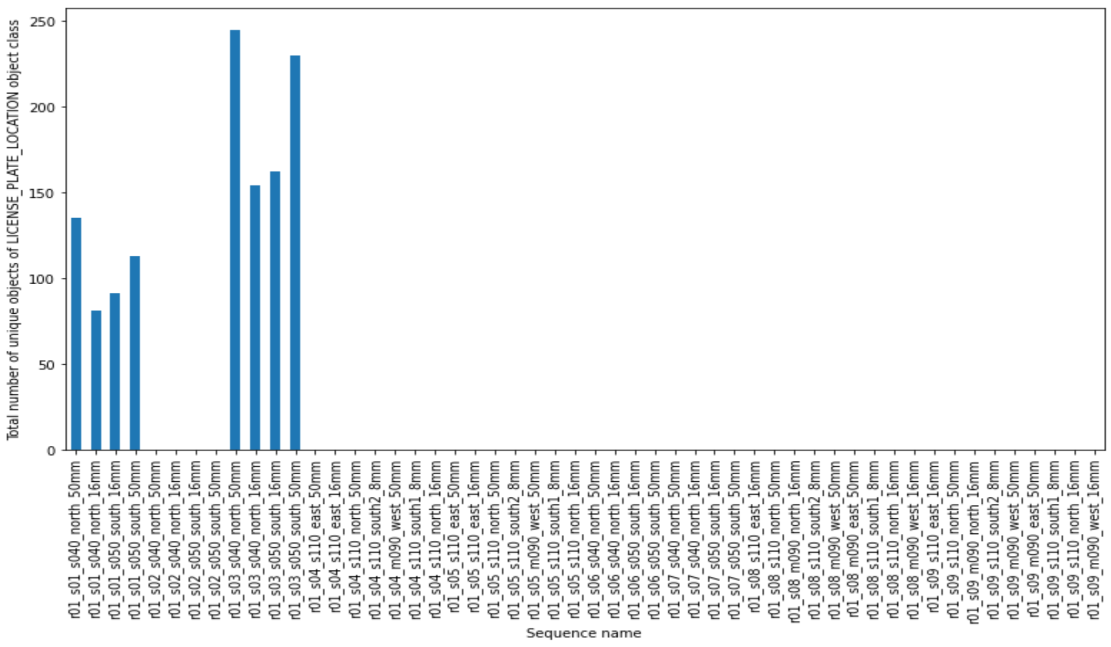


Figure 5.13: A distribution of the total number of unique objects of the LICENSE_PLATE_LOCATION object class across the A9 dataset.

5.2 Detection experiments

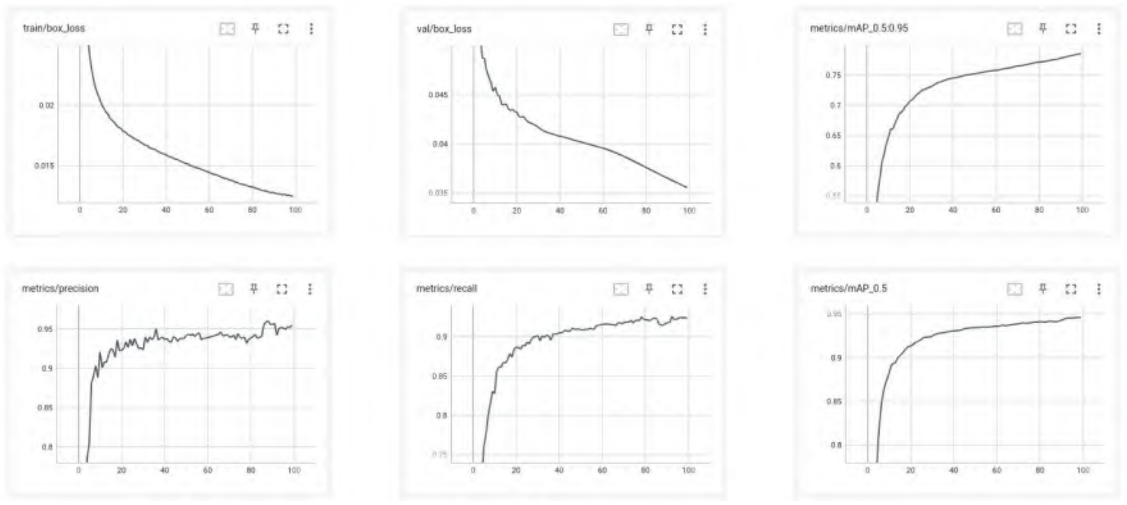


Figure 5.14: YOLOv7 training results, with 1216 X 1216 image-size parameter value.

Exp.	GPU	#GPUs	#workers	Time	#epochs	image-size	batch-size	pre-trained	#train images	#val images	#test images
1	NVIDIA TITAN RTX (24GB)	2 (90% used)	8	40 hours	100	1216 X 1216	16	No	25648	2280	570

Table 5.11: Overview of YOLOv7 detector's training experiment on the A9 dataset.

- We trained a **YOLOv7 model from scratch** on the entire A9 dataset instead of using pretrained MS-COCO weights, because out of the 11 classes of the A9 dataset, only 6 are found in the MS-COCO dataset.
- We trained the model using an **image-size of 1216 X 1216** which is slightly higher than the lower dimension of the images in the A9 dataset, namely, 1920 X 1200.
- Although higher resolutions (larger image-size) have been observed to provide better results, they are also significantly more memory intensive to train (as we saw for the Detection experiments on the DLR dataset in Table 4.2) and given the large number of images in the A9 dataset, we were both limited by time and hardware. Besides, the inference time of higher resolution models is also larger. For example, the 640 X 640 YOLOv7 detection model of the DLR dataset took 1.966 seconds to perform inference on a test image, whereas the 3200 X 3200 YOLOv7 detection model took 2.017 seconds to perform inference on the same test image.
- An overview of the training experiment can be found in Table 5.11. The quantitative results of the training can be found in Figure 5.14. The quantitative results on the validation set can be found in Table 5.12. The quantitative results on the test set can be found in Table 5.13. The qualitative results obtained by inferencing the trained model on a test image, can be found in Figure 5.15.

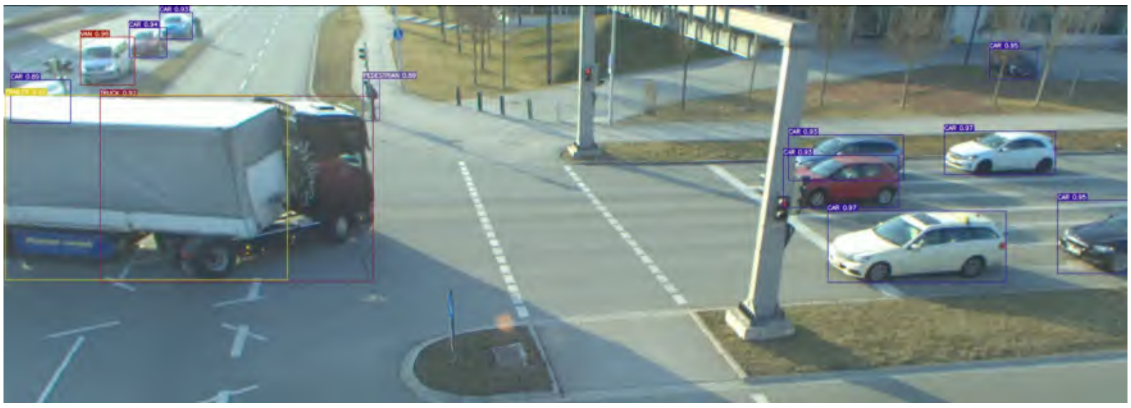


Figure 5.15: YOLOv7 inference result, with 1280 X 1280 image-size parameter value, on a test image of the A9 dataset.

Class	#Images	#Labels	Precision	Recall	mAP@.5	mAP@.5:.95
all	**2280**	**54275**	**0.955**	**0.923**	**0.946**	**0.785**
CAR	2280	30959	0.967	0.984	0.994	0.872
TRUCK	2280	5571	0.974	0.992	0.997	0.869
TRAILER	2280	2968	0.965	0.979	0.987	0.891
VAN	2280	4999	0.966	0.989	0.994	0.891
MOTORCYCLE	2280	192	0.974	0.984	0.986	0.832
BUS	2280	165	0.975	0.994	0.995	0.866
PEDESTRIAN	2280	926	0.921	0.954	0.973	0.684
BICYCLE	2280	192	0.942	0.943	0.981	0.74
EMERGENCY_VEHICLE	2280	198	1	0.954	0.973	0.9
OTHER	2280	139	0.94	1	0.993	0.884
LICENSE_PLATE_LOCATION	2280	7966	0.88	0.382	0.535	0.21

Table 5.12: YOLOv7 quantitative results on the validation set, with 1216 X 1216 image-size parameter value.

Class	#Images	#Labels	Precision	Recall	mAP@.5	mAP@.5:.95
all	**570**	**13536**	**0.949**	**0.931**	**0.949**	**0.78**
CAR	570	7763	0.967	0.982	0.993	0.872
TRUCK	570	1437	0.969	0.992	0.996	0.868
TRAILER	570	778	0.973	0.985	0.993	0.894
VAN	570	1194	0.961	0.99	0.995	0.9
MOTORCYCLE	570	43	0.981	1	0.996	0.857
BUS	570	49	0.96	1	0.994	0.877
PEDESTRIAN	570	270	0.908	0.922	0.953	0.64
BICYCLE	570	34	0.914	1	0.996	0.742
EMERGENCY_VEHICLE	570	71	0.986	0.915	0.968	0.861
OTHER	570	31	0.974	1	0.996	0.854
LICENSE_PLATE_LOCATION	570	1866	0.85	0.452	0.565	0.221

Table 5.13: YOLOv7 quantitative results on the test set, with 1280 X 1280 image-size parameter value.

5.3 Tracking experiments

In this section, we would like to present the tracking results of several algorithms, namely, SORT [9], ByteTrack [11], DeepSORT [12], and our proposed Byte-De-SORT on all the sequences of the A9 dataset. For all tracking experiments, we used NVIDIA TITAN RTX (24 GB) GPU and we used the same YOLOv7 detector, which was trained from scratch on the A9 dataset. We use an image-size of 1280 X 1280 for inferencing, since that was the optimal choice between speed and accuracy for this dataset. We do not display track history in the qualitative results, because that is already indicated by the track-id and bounding box colour, and we focus on specific parts of the frame and not the entire frame, generally over a long period of time (non-consecutive frames), to ascertain long-term tracking performance.

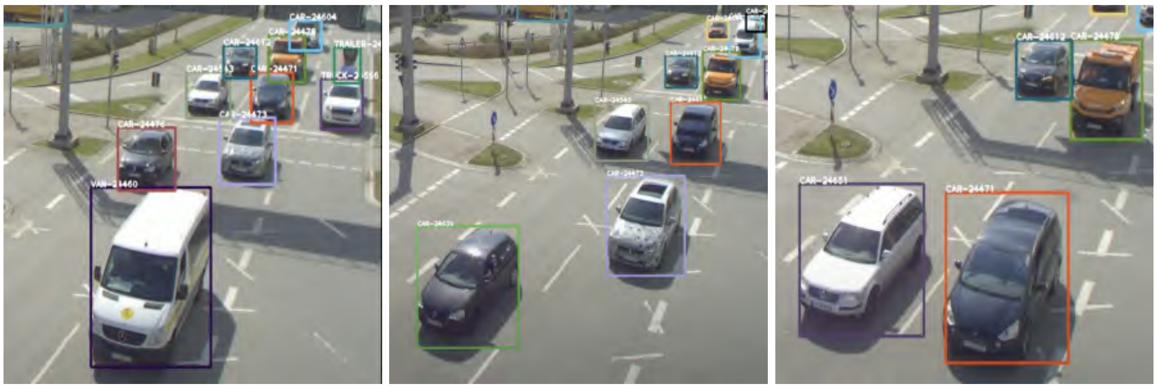


Figure 5.16: SORT's qualitative performance on the A9 dataset. These images were taken from the **r01_s08_s110_south2_8mm** sequence.

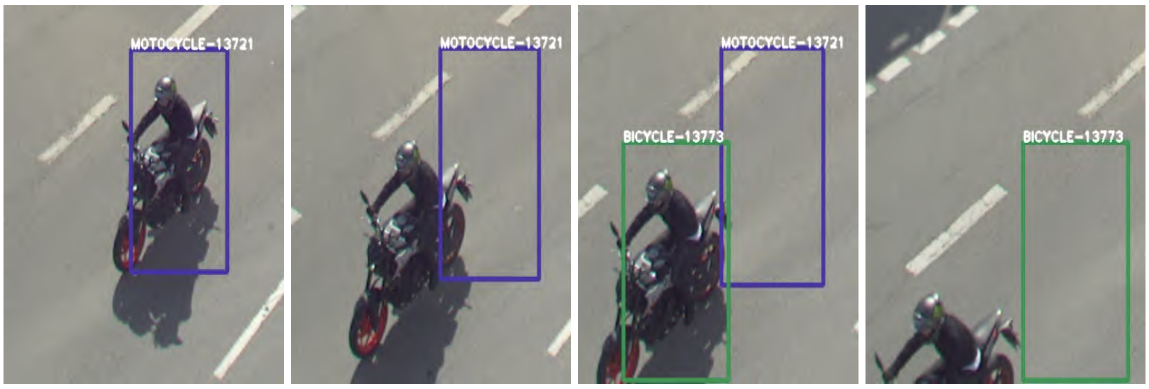


Figure 5.17: ByteTrack's qualitative performance on the A9 dataset. These images were taken from the **r01_s08_s110_south2_8mm** sequence.

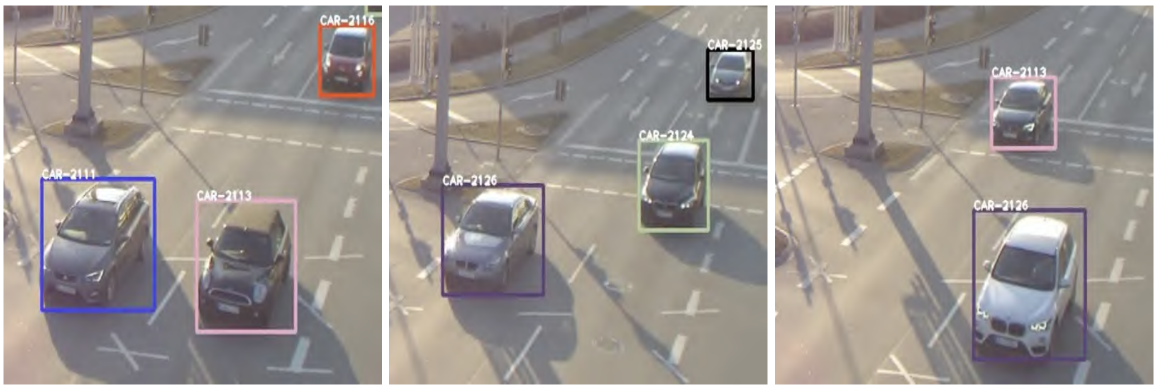


Figure 5.18: DeepSORT's qualitative performance on the A9 dataset. These images were taken from the **r01_s05_s110_south2_8mm** sequence.

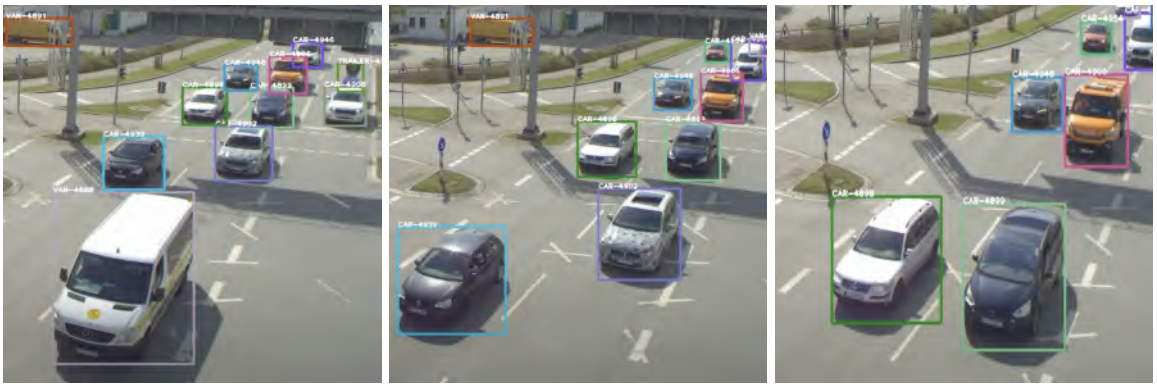


Figure 5.19: Byte-De-SORT's qualitative performance on the A9 dataset. These images were taken from the **r01_s08_s110_south2_8mm** sequence.

- **SORT**: As of writing this thesis, SORT is currently planned to be used in the 3D perception pipeline in the near future of the Providentia [5] project. Therefore, it made sense to judge its performance with the detector we have trained. SORT is a straightforward tracker without any algorithmic complexities and its performance can shed light on the important characteristics needed by a tracker to do tracking on the A9 dataset. SORT achieved **HOTA: 0.62**, **MOTA: 0.69**, **IDF1: 0.74**, and **Average Inference Speed: 19.31 FPS** overall on the A9 dataset. The qualitative performance of SORT can be found in Figure 5.16. A9 dataset, being a medium-high FPS dataset with no camera motion, is perfect for an IoU based algorithm like SORT that also utilises the Kalman Filter. Because of high FPS, IoU thresholds between frames are easily met and because of no camera motion, Kalman Filter's predictions work well. Consequently, SORT gives us the best overall results on the A9 dataset amongst all the algorithms presented here. However, since SORT is purely IoU based, it is unable to work well when an object moves too fast or accelerates as is the case with the vehicles in the left lane in Figure 5.16. It is difficult for a fast moving object to have a large IoU with itself from the previous frame. Moreover, Kalman Filter works on the assumption of constant velocity. Therefore, if an object rapidly accelerates, it fails to be tracked correctly by SORT.

- **ByteTrack**: ByteTrack is one of the SOTA methods used for tracking on the MOT benchmark datasets. ByteTrack has the special characteristic of matching low-score detections with remaining tracks after matching high-score detections. This has proven to work well on pedestrian heavy MOT datasets and we can verify whether it works well on a vehicle heavy dataset like the A9 dataset. ByteTrack achieved **HOTA: 0.58**, **MOTA: 0.63**, **IDF1: 0.71**, and **Average Inference Speed: 18.88 FPS** overall on the A9 dataset. The qualitative performance of ByteTrack can be found in Figure 5.17. Despite being a SOTA method and being IoU based, ByteTrack falls short of being the best tracking algorithm for the A9 dataset, and loses the top spot to SORT by a small margin. The consideration of low-score detections which is the highlight of ByteTrack, that works for MOT benchmarks and pedestrian datasets, leads to False Positives on the A9 dataset (as can be seen in Figure 5.17). The low-score detections were primarily built into ByteTrack to deal with occluded pedestrians in crowded scenarios in the MOT benchmarks. However, A9 dataset has several non-crowded and non-occluded scenarios and is primarily composed of large vehicles and very few pedestrians. Therefore, instead of increasing performance, low-score detections end up decreasing it. Because of considering low-score detections, ByteTrack is also slower at inferencing than SORT.

- **DeepSORT**: DeepSORT is a great choice when we wish to verify the importance of appearance based features for data-association instead of the conventional IoU based data association. We have not removed the Kalman Filter from DeepSORT like we did for the DLR dataset. This is because A9 dataset is relatively high FPS and has no camera motion and therefore Kalman Filter works well on this dataset. DeepSORT achieved **HOTA: 0.50**, **MOTA: 0.58**, **IDF1: 0.56**, and **Average Inference Speed: 15.22 FPS** overall on the A9 dataset. The qualitative performance of DeepSORT can be found in Figure 5.18. DeepSORT is slower than IoU based methods at inferencing because a feature embedding needs to be calculated for every detection using a ResNet-50 backbone. DeepSORT does not utilise crucial location based information, e.g. where the object was in the previous frame and where is it in the current frame. This information is highly beneficial in a high FPS dataset, since in this case the object should generally be quite near to itself from the previous frame. Since only appearance based features are given importance, sometimes pedestrians are mistracked as bicycles and vice versa, since they look similar. Sometimes ID-switches also happen between similar looking vehicles (as can be seen for both CAR-2126 and CAR-2113 in Figure 5.18). Because of these reasons, DeepSORT performs worse when compared to ByteTrack and SORT.

- **Byte-De-SORT**: While Byte-De-SORT was primarily developed for the DLR dataset, it also provides an opportunity to test a new method, which is completely devoid of IoU and Kalman Filter and depends solely on appearance features, on the A9 dataset. This is the only method amongst our chosen methods that is devoid of a Kalman Filter and therefore, the results obtained with this method will clearly reflect the importance of Kalman Filter for the A9 dataset. Byte-De-SORT achieved **HOTA: 0.48**, **MOTA: 0.68**, **IDF1: 0.51**, and **Average Inference Speed: 14.34 FPS** overall on the A9 dataset. The qualitative performance of Byte-De-SORT can be found in Figure 5.19. Byte-De-SORT is the worst performer amongst all the algorithms that we tried for tracking on the A9 dataset. However, there is a logical reasoning behind this. Byte-De-SORT was primarily designed to work well on the DLR dataset, which is a low FPS dataset with large camera motion, and therefore quite the opposite of the A9 dataset. Byte-De-SORT is a combination of ByteTrack and DeepSORT and performs worse than either of them on the A9 dataset. This is because unlike DeepSORT, Byte-De-SORT does not use a Kalman Filter which is useful in the A9 dataset for motion prediction because of no camera motion. Byte-De-SORT performs worse than ByteTrack because unlike ByteTrack, it does not use IoU for data association which is useful in a high FPS dataset. Despite its flaws, Byte-De-SORT shines where the best algorithm for A9 dataset, namely SORT, does not. As can be seen in Figure 5.19, unlike SORT, Byte-De-SORT can consistently track fast moving vehicles, because of the absence of IoU based data association and Kalman Filter.

The A9 dataset has several sequences (9 in total, but because of multiple camera views, 52 overall in our case), with multiple classes of objects (11 classes) in very different scenarios (highways, intersections, day and night). Thus, it would make sense to have a detailed quantitative overview of the tracking results on all sequences. This will enable researchers in future to draw valuable insights as to which scenarios, camera angles, and object classes are conducive to tracking.

Sequence	**HOTA_BDS**	**HOTA_BT**	**HOTA_DS**	**HOTA_ST**	**IDF1_BDS**	**IDF1_BT**	**IDF1_DS**	**IDF1_ST**	**MOTA_BDS**	**MOTA_BT**	**MOTA_DS**	**MOTA_ST**
r01_s01_s040_north_16mm	0.37	0.46	0.37	0.49	0.38	0.58	0.41	0.60	0.47	0.41	0.35	0.46
r01_s01_s040_north_50mm	0.39	0.51	0.41	0.56	0.42	0.63	0.47	0.67	0.53	0.48	0.40	0.54
r01_s01_s050_south_16mm	0.35	0.51	0.40	0.56	0.35	0.65	0.45	0.67	0.52	0.46	0.40	0.54
r01_s01_s050_south_50mm	0.39	0.51	0.42	0.55	0.41	0.64	0.49	0.66	0.52	0.46	0.37	0.52

Table 5.14: Quantitative results of all tracking algorithms on the **r01_s01** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	**HOTA_BDS**	**HOTA_BT**	**HOTA_DS**	**HOTA_ST**	**IDF1_BDS**	**IDF1_BT**	**IDF1_DS**	**IDF1_ST**	**MOTA_BDS**	**MOTA_BT**	**MOTA_DS**	**MOTA_ST**
r01_s02_s040_north_16mm	0.39	0.54	0.38	0.55	0.44	0.79	0.48	0.77	0.73	0.65	0.57	0.69
r01_s02_s040_north_50mm	0.43	0.56	0.44	0.63	0.50	0.78	0.57	0.85	0.81	0.74	0.69	0.87
r01_s02_s050_south_16mm	0.36	0.54	0.37	0.56	0.41	0.77	0.46	0.77	0.72	0.68	0.58	0.74
r01_s02_s050_south_50mm	0.40	0.57	0.48	0.62	0.50	0.82	0.65	0.87	0.75	0.73	0.68	0.84

Table 5.15: Quantitative results of all tracking algorithms on the **r01_s02** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	HOTA_BDS	HOTA_BT	HOTA_DS	HOTA_ST	IDF1_BDS	IDF1_BT	IDF1_DS	IDF1_ST	MOTA_BDS	MOTA_BT	MOTA_DS	MOTA_ST
r01_s03_s040_north_16mm	0.38	0.50	0.35	0.51	0.39	0.62	0.37	0.62	0.54	0.46	0.38	0.50
r01_s03_s040_north_50mm	0.37	0.49	0.37	0.53	0.36	0.59	0.39	0.61	0.56	0.51	0.41	0.55
r01_s03_s050_south_16mm	0.43	0.54	0.41	0.58	0.44	0.66	0.45	0.68	0.57	0.51	0.45	0.56
r01_s03_s050_south_50mm	0.40	0.50	0.39	0.55	0.41	0.60	0.40	0.62	0.53	0.48	0.38	0.53

Table 5.16: Quantitative results of all tracking algorithms on the **r01_s03** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	HOTA_BDS	HOTA_BT	HOTA_DS	HOTA_ST	IDF1_BDS	IDF1_BT	IDF1_DS	IDF1_ST	MOTA_BDS	MOTA_BT	MOTA_DS	MOTA_ST
r01_s04_m090_west_50mm	0.56	0.65	0.61	0.71	0.64	0.84	0.76	0.88	0.80	0.76	0.71	0.84
r01_s04_s110_east_16mm	0.65	0.71	0.68	0.73	0.74	0.86	0.82	0.89	0.81	0.78	0.76	0.82
r01_s04_s110_east_50mm	0.55	0.74	0.70	0.81	0.52	0.85	0.76	0.90	0.89	0.81	0.84	0.91
r01_s04_s110_north_16mm	0.50	0.60	0.53	0.55	0.56	0.67	0.62	0.57	0.29	0.39	0.33	0.38
r01_s04_s110_north_50mm	0.66	0.76	0.73	0.79	0.64	0.78	0.74	0.80	0.64	0.62	0.61	0.66
r01_s04_s110_south1_8mm	0.68	0.73	0.70	0.76	0.73	0.86	0.80	0.89	0.79	0.74	0.75	0.80
r01_s04_s110_south2_8mm	0.69	0.79	0.79	0.82	0.74	0.94	0.93	0.95	0.96	0.93	0.93	0.97

Table 5.17: Quantitative results of all tracking algorithms on the **r01_s04** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	HOTA_BDS	HOTA_BT	HOTA_DS	HOTA_ST	IDF1_BDS	IDF1_BT	IDF1_DS	IDF1_ST	MOTA_BDS	MOTA_BT	MOTA_DS	MOTA_ST
r01_s05_m090_west_50mm	0.53	0.65	0.56	0.69	0.58	0.85	0.71	0.87	0.82	0.80	0.71	0.87
r01_s05_s110_east_16mm	0.60	0.65	0.61	0.71	0.66	0.83	0.74	0.87	0.80	0.71	0.68	0.81
r01_s05_s110_east_50mm	0.55	0.73	0.61	0.79	0.53	0.87	0.65	0.92	0.88	0.80	0.79	0.92
r01_s05_s110_north_16mm	0.31	0.38	0.28	0.41	0.30	0.37	0.27	0.40	-0.11	-0.05	-0.14	0.00
r01_s05_s110_north_50mm	0.57	0.70	0.72	0.71	0.56	0.78	0.84	0.77	0.77	0.75	0.74	0.78
r01_s05_s110_south1_8mm	0.60	0.71	0.70	0.75	0.67	0.90	0.88	0.92	0.82	0.79	0.79	0.86
r01_s05_s110_south2_8mm	0.67	0.78	0.74	0.79	0.72	0.91	0.86	0.91	0.93	0.90	0.87	0.93

Table 5.18: Quantitative results of all tracking algorithms on the **r01_s05** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	HOTA_BDS	HOTA_BT	HOTA_DS	HOTA_ST	IDF1_BDS	IDF1_BT	IDF1_DS	IDF1_ST	MOTA_BDS	MOTA_BT	MOTA_DS	MOTA_ST
r01_s06_s040_north_16mm	0.55	0.65	0.50	0.67	0.60	0.84	0.56	0.82	0.84	0.77	0.73	0.80
r01_s06_s040_north_50mm	0.55	0.64	0.56	0.70	0.61	0.80	0.66	0.85	0.84	0.77	0.74	0.87
r01_s06_s050_south_16mm	0.44	0.52	0.36	0.51	0.48	0.70	0.41	0.59	0.67	0.56	0.32	0.52
r01_s06_s050_south_50mm	0.53	0.65	0.54	0.71	0.58	0.80	0.63	0.86	0.82	0.76	0.73	0.86

Table 5.19: Quantitative results of all tracking algorithms on the **r01_s06** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	HOTA_BDS	HOTA_BT	HOTA_DS	HOTA_ST	IDF1_BDS	IDF1_BT	IDF1_DS	IDF1_ST	MOTA_BDS	MOTA_BT	MOTA_DS	MOTA_ST
r01_s07_s040_north_16mm	0.53	0.62	0.52	0.66	0.58	0.78	0.60	0.80	0.88	0.78	0.79	0.89
r01_s07_s040_north_50mm	0.54	0.67	0.58	0.72	0.60	0.83	0.66	0.87	0.84	0.81	0.77	0.88
r01_s07_s050_south_16mm	0.46	0.54	0.31	0.46	0.50	0.78	0.35	0.60	0.70	0.60	0.41	0.46
r01_s07_s050_south_50mm	0.50	0.60	0.53	0.67	0.59	0.77	0.64	0.83	0.78	0.74	0.70	0.81

Table 5.20: Quantitative results of all tracking algorithms on the **r01_s07** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	HOTA_BDS	HOTA_BT	HOTA_DS	HOTA_ST	IDF1_BDS	IDF1_BT	IDF1_DS	IDF1_ST	MOTA_BDS	MOTA_BT	MOTA_DS	MOTA_ST
r01_s08_m090_east_50mm	0.48	0.59	0.41	0.58	0.47	0.73	0.39	0.67	0.62	0.53	0.47	0.59
r01_s08_m090_north_16mm	0.48	0.49	0.51	0.55	0.56	0.58	0.59	0.65	0.76	0.73	0.71	0.77
r01_s08_m090_west_16mm	0.53	0.58	0.56	0.60	0.56	0.64	0.60	0.65	0.66	0.63	0.61	0.66
r01_s08_m090_west_50mm	0.45	0.55	0.46	0.60	0.50	0.67	0.52	0.73	0.76	0.72	0.68	0.78
r01_s08_s110_east_16mm	0.55	0.62	0.60	0.65	0.55	0.72	0.66	0.74	0.78	0.70	0.70	0.77
r01_s08_s110_north_16mm	0.55	0.66	0.63	0.69	0.59	0.75	0.71	0.78	0.62	0.58	0.58	0.64
r01_s08_s110_north_50mm	0.54	0.68	0.64	0.73	0.58	0.81	0.71	0.85	0.88	0.82	0.80	0.90
r01_s08_s110_south1_8mm	0.54	0.65	0.54	0.70	0.61	0.79	0.61	0.83	0.74	0.65	0.65	0.75
r01_s08_s110_south2_8mm	0.53	0.67	0.64	0.71	0.56	0.81	0.74	0.83	0.91	0.85	0.84	0.91

Table 5.21: Quantitative results of all tracking algorithms on the **r01_s08** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	HOTA_BDS	HOTA_BT	HOTA_DS	HOTA_ST	IDF1_BDS	IDF1_BT	IDF1_DS	IDF1_ST	MOTA_BDS	MOTA_BT	MOTA_DS	MOTA_ST
r01_s09_m090_east_50mm	0.40	0.50	0.45	0.49	0.54	0.74	0.61	0.72	0.57	0.48	0.46	0.55
r01_s09_m090_north_16mm	0.42	0.65	0.39	0.47	0.55	0.94	0.52	0.66	0.78	0.89	0.65	0.75
r01_s09_m090_west_16mm	0.48	0.54	0.43	0.52	0.51	0.67	0.42	0.63	0.41	0.37	0.33	0.34
r01_s09_m090_west_50mm	0.52	0.64	0.49	0.65	0.66	0.88	0.58	0.88	0.85	0.85	0.84	0.86
r01_s09_s110_east_16mm	0.51	0.67	0.58	0.71	0.59	0.88	0.67	0.89	0.87	0.77	0.78	0.85
r01_s09_s110_north_16mm	0.86	0.86	0.86	0.86	0.98	0.98	0.98	0.99	0.97	0.97	0.97	0.97
r01_s09_s110_north_50mm	0.76	0.81	0.79	0.81	0.86	0.95	0.94	0.95	0.92	0.91	0.89	0.92
r01_s09_s110_south1_8mm	0.46	0.63	0.62	0.65	0.49	0.81	0.80	0.82	0.67	0.62	0.62	0.65
r01_s09_s110_south2_8mm	0.67	0.80	0.78	0.81	0.77	0.95	0.92	0.96	0.94	0.92	0.92	0.94

Table 5.22: Quantitative results of all tracking algorithms on the **r01_s09** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT.

Sequence	HOTA_BDS	HOTA_BT	HOTA_DS	HOTA_ST	IDF1_BDS	IDF1_BT	IDF1_DS	IDF1_ST	MOTA_BDS	MOTA_BT	MOTA_DS	MOTA_ST
COMBINED	0.48	0.58	0.50	0.62	0.51	0.71	0.56	0.74	0.68	0.63	0.58	0.69

Table 5.23: Overall quantitative results of all tracking algorithms on all sequences of the A9 dataset.

5.4 Conclusion

$$\textit{Kalman Filter} \approx \textit{IoU} > \textit{Appearance Features} > \textit{Camera Motion Compensation} \qquad (5.1)$$

Table 5.23 contains the overall quantitative tracking results of all algorithms on all sequences of the A9 dataset. SORT gives the best quantitative results while our proposed algorithm Byte-De-SORT performs relatively the worst on the A9 dataset. However, overall, all 4 algorithms managed to give decent results. IoU based algorithms like SORT and ByteTrack produced relatively better HOTA scores of 0.62 and 0.58 respectively in comparison to appearance features based algorithms like DeepSORT and Byte-De-SORT that managed to produce HOTA scores of 0.50 and 0.48 respectively. SORT achieved the highest MOTA score of 0.69, closely followed by Byte-De-SORT which achieved a MOTA score of 0.68. SORT and ByteTrack achieved relatively higher IDF1 scores of 0.74 and 0.71 respectively in comparison to DeepSORT and Byte-De-SORT that achieved IDF1 scores of 0.56 and 0.51 respectively. SORT clocked the fastest inference speed at 19.31 FPS and Byte-De-SORT clocked the slowest inference speed at 14.34 FPS, making all 4 algorithms relatively suitable for real-time tracking on the A9 dataset. It is quite clear that unlike the DLR dataset, IoU and Kalman Filter work quite well for the A9 dataset since it is a high FPS dataset with no camera motion. Therefore, for the A9 dataset, we can safely say that the order of importance of different features of a tracking algorithm can be given by the Equation 5.1. Byte-De-SORT, however, proved to be quite effective in tracking fast moving vehicles that were specially accelerating quickly. This is because IoU based data association is difficult for fast moving objects between frames and Kalman Filter is based on the constant velocity assumption and therefore works poorly with rapid acceleration. Since Byte-De-SORT lacks IoU based data association and Kalman Filter, it could track fast moving vehicles well. Therefore, as per observations, **SORT stands out as the overall best solution for real-time tracking while Byte-De-SORT is a good candidate for tracking fast moving objects on the A9 dataset**.

6 Conclusion

- All 4 tracking algorithms that were tested on the A9 dataset provided decent results, which is unlike the case of the DLR dataset where IoU based algorithms like ByteTrack performed quite badly. Therefore, it can be said that **the DLR dataset is more challenging for tracking than the A9 dataset**, particularly when MOT benchmark based tracking algorithms are applied.
- **We did not try BoT-SORT and camera motion compensation on the A9 dataset**, since the cameras are static and therefore there was no special advantage to be gained. Similarly, **UAVMOT did not make sense for the A9 dataset** since it was designed for datasets captured from drones and also had a Local Relational Filter to compensate for the drone's motion.
- Similarly, **we did not try the SORT algorithm on the DLR dataset**, because from the pattern observed it was clear that it would perform even worse than ByteTrack on that dataset and ByteTrack had already performed quite poorly on the DLR dataset.
- We understood that **IoU based data association and Kalman Filter work best for slow moving objects in a high FPS dataset containing no camera motion**. For other cases, appearance features based data association is the way to go. Effective camera motion compensation, like in BoT-SORT, is quite computationally expensive and not suitable for real-time tracking.
- **SOTA algorithms like ByteTrack are quite tailored for popular pedestrian datasets like the MOT benchmarks**. However, these algorithms are not the best when it comes to tracking vehicles.
- **Byte-De-SORT and DeepSORT gave decent performance in both datasets**, unlike other algorithms which made sense for just one of the datasets. However, **we used DeepSORT without Kalman Filter for the DLR dataset** and therefore technically it was not DeepSORT anymore.
- Thus, **our proposed method Byte-De-SORT**, which is a combination of ByteTrack and DeepSORT, with IoU based data association and Kalman Filter completely removed, **alone emerged as a robust real-time tracking algorithm that worked well for two very contrasting datasets, the DLR dataset with low FPS image sequences containing large camera motion and the A9 dataset with high FPS image sequences containing zero camera motion**.

7 Future Work

- With this thesis, we established previously non-existent baselines for object tracking on not one but two contrasting datasets. Analysing and further building up on those baselines will be expected of the successive works that are carried out in future. For example, A9 dataset has a large data diversity including image sequences from both day and night, highways and intersections, containing 11 classes of objects. The quantitative results that were produced as part of this work, can be analysed to build more robust algorithms that work in a variety of scenes.
- We tested tracking algorithms on the A9 and DLR datasets which were trained on the VisDrone dataset. We can expect a potential improvement in performance when the algorithms are trained on these datasets themselves.
- While using appearance features, the bounding boxes can be scaled to include some local surrounding information. This can hopefully compensate for the lack of location based information that is utilised by IoU based systems and is missed by the appearance features based systems.
- The DLR dataset is yet to be published, and therefore we have to limit the information supplied by this thesis about the dataset. That being said, we filtered out several sequences from the DLR dataset based on GSD and FPS. We also converted the multi-class dataset into a single-class one, containing only a *vehicle* class. We converted the oriented bounding boxes to rectangular bounding boxes. In future, when the dataset is published, we can come up with more robust algorithms that can work in a generic fashion on this dataset.
- We trained detection models to predict only the bounding boxes. However, to produce a digital twin as in the Providentia project, instance segmentation is needed and can be worked upon in future. Because of time and resource limitation, we never trained the detectors more than 100 epochs, and also upto a certain image resolution only. Training for greater number of epochs on higher resolution may lead to improvements in performance. Also, at the time of writing this thesis, YOLOv8 was in active development and it promises performance improvements over the YOLOv7 based models that we have used.
- Our proposed method Byte-De-SORT can be further improved by including a camera motion compensation component in it which is effective like BoT-SORT's SIFT module but should not be computationally expensive like it. Sparse Optical Flow methods could be a good starting point in this direction.

List of Figures

2.1 Example of Object Classification (image source [16]) 5
2.2 AlexNet architecture (image source [17]) 6
2.3 VGGNet architecture (image source [19]) 7
2.4 Inception module architecture (image source [20]) 7
2.5 Residual block architecture (image source [21]) 8
2.6 Example of Object Detection (image source [22]) 10
2.7 R-CNN (image source [27]) 11
2.8 YOLO (image source [28]) 11
2.9 YOLOv7 performance (image source [7]) 13
2.10 Different domains of MOT. (From left to right: MOT-16 (source [31]), KIT AIS (source [32]), A9 (source [6])) 14
2.11 IoU: a comparison of different bounding boxes. Red *bbox* is GT and Blue *bbox* is *Prediction* (image source [33]) 15
2.12 IoU: an illustration of how it is calculated. Red *bbox* is GT and Blue *bbox* is *Prediction* (image source [33]) 16
2.13 Example of re-ID in the domain of person re-identification (image source [35]) 17
2.14 Common errors in MOT. (a) An ID switch occurs when the mapping switches from the previously assigned red track to the blue one. (b) A track fragmentation is counted in frame 3 because the target is tracked in frames 1-2, then interrupts, and then reacquires its 'tracked' status at a later point. A new (blue) track hypothesis also causes an ID switch at this point. (c) Note that no fragmentations are counted in frames 3 and 6 because tracking of those targets is not resumed at a later point. (image and caption source [2]) 19
2.15 IDF1 calculation example (image source [40]) 20
2.16 A simple tracking example highlighting one of the main differences between evaluation metrics. Three different trackers are shown in order of increasing detection accuracy and decreasing association accuracy. MOTA and IDF1 overemphasize the effect of accurate detection and association respectively. HOTA balances both of these by being an explicit combination of a detection score DetA and an association score AssA. (image and caption source [40]) 21
2.17 Diagrammatic representation of how HOTA can be decomposed into separate sub-metrics which are able to differentiate between different types of tracking errors. (image and caption source [40]) 21

3.1 Pipeline of TransTrack (image and caption source [45]) 26
3.2 An overview of the KIT AIS Vehicle Dataset (image source [43]) 27
3.3 Overview of the AerialMPTNet architecture including an SNN, an LSTM, and a GCNN module. The inputs are two consecutive images in a sequence, cropped and centered to a target object, and the output is the object coordinates in the second snippet which is then mapped to the image coordinates. (image and caption source [48]) 28
3.4 Overview of MPNTrack (image source [51]) 30
3.5 Overview of SiamMOT (image source [53]) 32
3.6 Overview of TrackFormer (image source [56]) 34
3.7 Overview of UAVMOT (image source [59]) 38
3.8 Schematic diagram of relative relation vector (image and caption source [59]) 39
3.9 Overview of BoT-SORT (image source [13]) 40

4.1 Frame 1 and Frame 20 of the **2021-07-21-ATM_43** DLR dataset sequence. 44
4.2 Ground truth visualisation of bounding boxes, on an image of the **2021-07-21-ATM_43** sequence. 45
4.3 YOLOv7 inference result, with 640 X 640 image-size parameter value, on an image of the **2021-07-21-ATM_43** sequence. 46
4.4 YOLOv7 training results, with 640 X 640 image-size parameter value. . . 46
4.5 YOLOv7 inference result, with 1280 X 1280 image-size parameter value, on an image of the **2021-07-21-ATM_43** sequence. 47
4.6 YOLOv7 training results, with 1280 X 1280 image-size parameter value. . 47
4.7 YOLOv7 inference result, with 2560 X 2560 image-size parameter value, on an image of the **2021-07-21-ATM_43** sequence. 48
4.8 YOLOv7 training results, with 2560 X 2560 image-size parameter value. . 48
4.9 YOLOv7 inference result, with 3200 X 3200 image-size parameter value, on an image of the **2021-07-21-ATM_43** sequence. 49
4.10 YOLOv7 training results, with 3200 X 3200 image-size parameter value. . 49
4.11 ByteTrack's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence. . . 50
4.12 UAVMOT's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence. . . 50
4.13 DeepSORT's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence. 51
4.14 BoT-SORT's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence. . . 51
4.15 Byte-De-SORT's qualitative performance on the DLR dataset. Here we can see Frame 1, Frame 15 and Frame 25 of the **2021-07-21-ATM_43** sequence. 51

5.1 A distribution of the total number of labelled bounding boxes (annotations) across the A9 dataset. 60

5.2 A distribution of the total number of image frames across the A9 dataset. 60
5.3 A distribution of the total number of unique objects of the CAR object class across the A9 dataset. 61
5.4 A distribution of the total number of unique objects of the TRUCK object class across the A9 dataset. 61
5.5 A distribution of the total number of unique objects of the TRAILER object class across the A9 dataset. 62
5.6 A distribution of the total number of unique objects of the VAN object class across the A9 dataset. 62
5.7 A distribution of the total number of unique objects of the MOTORCYCLE object class across the A9 dataset. 63
5.8 A distribution of the total number of unique objects of the BUS object class across the A9 dataset. 63
5.9 A distribution of the total number of unique objects of the PEDESTRIAN object class across the A9 dataset. 64
5.10 A distribution of the total number of unique objects of the BICYCLE object class across the A9 dataset. 64
5.11 A distribution of the total number of unique objects of the EMERGENCY_VEHICLE object class across the A9 dataset. 65
5.12 A distribution of the total number of unique objects of the OTHER object class across the A9 dataset. 65
5.13 A distribution of the total number of unique objects of the LICENSE_PLATE_LOCATION object class across the A9 dataset. 66
5.14 YOLOv7 training results, with 1216 X 1216 image-size parameter value. . 67
5.15 YOLOv7 inference result, with 1280 X 1280 image-size parameter value, on a test image of the A9 dataset. 68
5.16 SORT's qualitative performance on the A9 dataset. These images were taken from the **r01_s08_s110_south2_8mm** sequence. 69
5.17 ByteTrack's qualitative performance on the A9 dataset. These images were taken from the **r01_s08_s110_south2_8mm** sequence. 69
5.18 DeepSORT's qualitative performance on the A9 dataset. These images were taken from the **r01_s05_s110_south2_8mm** sequence. 70
5.19 Byte-De-SORT's qualitative performance on the A9 dataset. These images were taken from the **r01_s08_s110_south2_8mm** sequence. 70

List of Tables

3.1 An overview of MPNTrack's results on MOT benchmarks. 30
3.2 An overview of TrackFormer's results on MOT17 test set. 34
3.3 An overview of ByteTrack's results on MOT benchmarks. 36
3.4 An overview of UAVMOT's results. 39
3.5 An overview of BoT-SORT's results on MOT benchmarks. 41

4.1 Overview of the DLR dataset. 44
4.2 Overview of YOLOv7 detector's training experiments on the DLR dataset. 45
4.3 Quantitative results of ByteTrack on DLR dataset. 54
4.4 Quantitative results of UAVMOT on DLR dataset. 54
4.5 Quantitative results of DeepSORT (without Kalman Filter) on DLR dataset. 54
4.6 Quantitative results of BoT-SORT on DLR dataset. 55
4.7 Quantitative results of Byte-De-SORT on DLR dataset. 55

5.1 Classes of objects in the A9 dataset. Mapping of object class name to object class id. 58
5.2 Overview of the **r01_s01** sequence of the A9 dataset. 58
5.3 Overview of the **r01_s02** sequence of the A9 dataset. 58
5.4 Overview of the **r01_s03** sequence of the A9 dataset. 58
5.5 Overview of the **r01_s04** sequence of the A9 dataset. 58
5.6 Overview of the **r01_s05** sequence of the A9 dataset. 59
5.7 Overview of the **r01_s06** sequence of the A9 dataset. 59
5.8 Overview of the **r01_s07** sequence of the A9 dataset. 59
5.9 Overview of the **r01_s08** sequence of the A9 dataset. 59
5.10 Overview of the **r01_s09** sequence of the A9 dataset. 59
5.11 Overview of YOLOv7 detector's training experiment on the A9 dataset. . 67
5.12 YOLOv7 quantitative results on the validation set, with 1216 X 1216 image-size parameter value. 68
5.13 YOLOv7 quantitative results on the test set, with 1280 X 1280 image-size parameter value. 68
5.14 Quantitative results of all tracking algorithms on the **r01_s01** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 72
5.15 Quantitative results of all tracking algorithms on the **r01_s02** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 72

5.16 Quantitative results of all tracking algorithms on the **r01_s03** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 73
5.17 Quantitative results of all tracking algorithms on the **r01_s04** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 73
5.18 Quantitative results of all tracking algorithms on the **r01_s05** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 73
5.19 Quantitative results of all tracking algorithms on the **r01_s06** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 73
5.20 Quantitative results of all tracking algorithms on the **r01_s07** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 73
5.21 Quantitative results of all tracking algorithms on the **r01_s08** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 73
5.22 Quantitative results of all tracking algorithms on the **r01_s09** sequence of the A9 dataset. BDS: Byte-De-SORT, BT: ByteTrack, DS: DeepSORT, ST: SORT. 74
5.23 Overall quantitative results of all tracking algorithms on all sequences of the A9 dataset. 74

Bibliography

[1] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler. *MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking*. 2015. arXiv: 1504.01942 [cs.CV].

[2] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler. *MOT16: A Benchmark for Multi-Object Tracking*. 2016. DOI: 10.48550/ARXIV.1603.00831. URL: https://arxiv.org/abs/1603.00831.

[3] *MOT17*. https://motchallenge.net/data/MOT17/. Accessed: 2023-03-09.

[4] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé. *MOT20: A benchmark for multi object tracking in crowded scenes*. 2020. arXiv: 2003.09003 [cs.CV].

[5] A. Krämmer, C. Schöller, D. Gulati, V. Lakshminarasimhan, F. Kurz, D. Rosenbaum, C. Lenz, and A. Knoll. *Providentia – A Large-Scale Sensor System for the Assistance of Autonomous Vehicles and Its Evaluation*. 2021. arXiv: 1906.06789 [cs.RO].

[6] C. Creß, W. Zimmer, L. Strand, V. Lakshminarasimhan, M. Fortkord, S. Dai, and A. Knoll. *A9-Dataset: Multi-Sensor Infrastructure-Based Dataset for Mobility Research*. 2022. DOI: 10.48550/ARXIV.2204.06527. URL: https://arxiv.org/abs/2204.06527.

[7] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. DOI: 10.48550/ARXIV.2207.02696. URL: https://arxiv.org/abs/2207.02696.

[8] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312.

[9] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. "Simple online and realtime tracking". In: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016. DOI: 10.1109/icip.2016.7533003. URL: https://doi.org/10.1109%2Ficip.2016.7533003.

[10] *DLR Homepage*. https://www.dlr.de/EN/Home/home_node.html. Accessed: 2023-02-26.

[11] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang. *ByteTrack: Multi-Object Tracking by Associating Every Detection Box*. 2021. DOI: 10.48550/ARXIV.2110.06864. URL: https://arxiv.org/abs/2110.06864.

[12] N. Wojke, A. Bewley, and D. Paulus. *Simple Online and Realtime Tracking with a Deep Association Metric*. 2017. DOI: 10.48550/ARXIV.1703.07402. URL: https://arxiv.org/abs/1703.07402.

[13] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky. *BoT-SORT: Robust Associations Multi-Pedestrian Tracking*. 2022. arXiv: 2206.14651 [cs.CV].

[14] G. Bradski. "The openCV library." In: *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 25.11 (2000), pp. 120–123.

[15] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling. *Detection and Tracking Meet Drones Challenge*. 2020. DOI: 10.48550/ARXIV.2001.06303. URL: https://arxiv.org/abs/2001.06303.

[16] *Faster R-CNN Object Detection with PyTorch*. https://learnopencv.com/faster-r-cnn-object-detection-with-pytorch/. Accessed: 2023-02-22.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. Burges, L. Bottou, and K. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[19] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going Deeper with Convolutions*. 2014. DOI: 10.48550/ARXIV.1409.4842. URL: https://arxiv.org/abs/1409.4842.

[21] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[22] A. Ouaknine. *Review of Deep Learning Algorithms for Object Detection*. https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852. Accessed: 2023-02-22.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. DOI: 10.48550/ARXIV.1311.2524. URL: https://arxiv.org/abs/1311.2524.

[24] R. Girshick. *Fast R-CNN*. 2015. DOI: 10.48550/ARXIV.1504.08083. URL: https://arxiv.org/abs/1504.08083.

[25] S. Ren, K. He, R. Girshick, and J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. DOI: 10.48550/ARXIV.1506.01497. URL: https://arxiv.org/abs/1506.01497.

[26] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* (2013). DOI: 10.1007/s11263-013-0620-5. URL: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[27] *Deep Learning for Object Detection: A Comprehensive Review*. https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9. Accessed: 2023-02-23.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640.

[29] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. http://www.pascalnetwork.org/challenges/VOC/voc2007/workshop/index.html.

[30] *Mean Average Precision (mAP) in Object Detection*. https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/. Accessed: 2023-02-25.

[31] Y. Xiang, A. Alahi, and S. Savarese. "Learning to Track: Online Multi-object Tracking by Decision Making". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4705–4713. DOI: 10.1109/ICCV.2015.534.

[32] *KIT AIS Data Set*. https://www.ipf.kit.edu/downloads_data_set_AIS_vehicle_tracking.php. Accessed: 2023-02-26.

[33] *Intersection over Union (IoU) in Object Detection & Segmentation*. https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/. Accessed: 2023-03-04.

[34] H. Masnadi-Shirazi, A. Masnadi-Shirazi, and M.-A. Dastgheib. *A Step by Step Mathematical Derivation and Tutorial on Kalman Filters*. 2019. DOI: 10.48550/ARXIV.1910.03558. URL: https://arxiv.org/abs/1910.03558.

[35] L. Zheng, Y. Yang, and A. G. Hauptmann. *Person Re-identification: Past, Present and Future*. 2016. DOI: 10.48550/ARXIV.1610.02984. URL: https://arxiv.org/abs/1610.02984.

[36] A. Hermans, L. Beyer, and B. Leibe. *In Defense of the Triplet Loss for Person Re-Identification*. 2017. DOI: 10.48550/ARXIV.1703.07737. URL: https://arxiv.org/abs/1703.07737.

[37] *The Hungarian method for the assignment problem*. https://web.eecs.umich.edu/~pettie/matching/Kuhn-hungarian-assignment.pdf. Accessed: 2023-03-04.

[38] K. Bernardin and R. Stiefelhagen. "Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics". In: *EURASIP Journal on Image and Video Processing* 2008 (2008), pp. 1–10.

[39] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi. *Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking*. 2016. DOI: `10.48550/ARXIV.1609.01775`. URL: `https://arxiv.org/abs/1609.01775`.

[40] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé, and B. Leibe. "HOTA: A Higher Order Metric for Evaluating Multi-Object Tracking". In: *International Journal of Computer Vision* (2020), pp. 1–31.

[41] *Evaluation Metrics for Multiple Object Tracking*. `https://arshren.medium.com/evaluation-metrics-for-multiple-object-tracking-7b26ef23ef5f`. Accessed: 2023-03-05.

[42] T. Beheim. "Multi-Vehicle Detection and Tracking in Aerial Imagery Sequences using Deep Learning Algorithms". 2021. URL: `https://elib.dlr.de/146903/`.

[43] M. Kraus. "Multi-Object Tracking in Aerial and Satellite Imagery". 2020. URL: `https://elib.dlr.de/138070/`.

[44] *TUM I6 Chair Homepage*. `https://www.ce.cit.tum.de/en/air/home/`. Accessed: 2023-03-07.

[45] P. Sun, J. Cao, Y. Jiang, R. Zhang, E. Xie, Z. Yuan, C. Wang, and P. Luo. *TransTrack: Multiple Object Tracking with Transformer*. 2020. DOI: `10.48550/ARXIV.2012.15460`. URL: `https://arxiv.org/abs/2012.15460`.

[46] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need*. 2017. DOI: `10.48550/ARXIV.1706.03762`. URL: `https://arxiv.org/abs/1706.03762`.

[48] M. Kraus, S. M. Azimi, E. Ercelik, R. Bahmanyar, P. Reinartz, and A. Knoll. *AerialMPTNet: Multi-Pedestrian Tracking in Aerial Imagery Using Temporal and Graphical Features*. 2020. DOI: `10.48550/ARXIV.2006.15457`. URL: `https://arxiv.org/abs/2006.15457`.

[49] R. Bahmanyar, S. Azimi, and P. Reinartz. "Multiple vehicle and people tracking in aerial imagery using stack of micro single-object-tracking CNNs". In: ISPRS. 2019.

[50] T. N. Kipf and M. Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2016. DOI: `10.48550/ARXIV.1609.02907`. URL: `https://arxiv.org/abs/1609.02907`.

[51] G. Braso and L. Leal-Taixe. "Learning a Neural Solver for Multiple Object Tracking". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[52] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. *Graph Neural Networks: A Review of Methods and Applications*. 2018. DOI: `10.48550/ARXIV.1812.08434`. URL: `https://arxiv.org/abs/1812.08434`.

[53] B. Shuai, A. Berneshawi, X. Li, D. Modolo, and J. Tighe. "SiamMOT: Siamese Multi-Object Tracking". In: (2021). DOI: `10.48550/ARXIV.2105.11595`. URL: `https://arxiv.org/abs/2105.11595`.

[54] A. Dave, T. Khurana, P. Tokmakov, C. Schmid, and D. Ramanan. *TAO: A Large-Scale Benchmark for Tracking Any Object*. 2020. DOI: `10.48550/ARXIV.2005.10356`. URL: `https://arxiv.org/abs/2005.10356`.

[55] D. Hall and P. Perona. *Fine-Grained Classification of Pedestrians in Video: Benchmark and State of the Art*. 2016. DOI: `10.48550/ARXIV.1605.06177`. URL: `https://arxiv.org/abs/1605.06177`.

[56] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer. *TrackFormer: Multi-Object Tracking with Transformers*. 2021. DOI: `10.48550/ARXIV.2101.02702`. URL: `https://arxiv.org/abs/2101.02702`.

[57] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe. "MOTS: Multi-Object Tracking and Segmentation". In: (2019). DOI: `10.48550/ARXIV.1902.03604`. URL: `https://arxiv.org/abs/1902.03604`.

[58] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun. *YOLOX: Exceeding YOLO Series in 2021*. 2021. DOI: `10.48550/ARXIV.2107.08430`. URL: `https://arxiv.org/abs/2107.08430`.

[59] S. Liu, X. Li, H. Lu, and Y. He. "Multi-Object Tracking Meets Moving UAV". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 8876–8885.

[60] D. Du, Y. Qi, H. Yu, Y. Yang, K. Duan, G. Li, W. Zhang, Q. Huang, and Q. Tian. *The Unmanned Aerial Vehicle Benchmark: Object Detection and Tracking*. 2018. DOI: `10.48550/ARXIV.1804.00518`. URL: `https://arxiv.org/abs/1804.00518`.

[61] T. Taketomi, H. Uchiyama, and S. Ikeda. "Visual SLAM algorithms: a survey from 2010 to 2016". In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (2017), p. 16. DOI: `10.1186/s41074-017-0027-2`. URL: `https://doi.org/10.1186/s41074-017-0027-2`.

[62] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60 (2004), pp. 91–110.

[63] *ASAM OpenLABEL*. `https://www.asam.net/standards/detail/openlabel/`. Accessed: 2023-04-03.