Wrocław University of Science and Technology
Faculty of Information and Communication Technology

Field of study: **Applied Computer Science**
Speciality: **Designing IT Systems**

MASTER THESIS

Trajectory prediction of cyclists in urban setting by using machine learning

Kamil Moszczyc

Supervisor
Radosław Michalski, PhD, DSc

machine learning, object detection, object tracking, cyclist, trajectory prediction

WROCŁAW 2023

Streszczenie

W wyniku wzrostu popularności jazdy na rowerze w ostatnich latach, problem bezpieczeństwa rowerzystów i innych użytkowników ruchu drogowego, w szczególności tych mniej chronionych, staje się palący. Rowerzyści – w przeciwieństwie do samochodów – są zwinni i stosunkowo niewielcy. Dodatkowo, rowerzyści poruszają się szybciej od pieszych, a w razie wypadku są równie wrażliwi na obrażenia jak oni. Pomoc kierowcom samochodów w wykrywaniu i przewidywaniu trajektorii rowerzystów, którzy często wyłaniają się niespodziewanie z różnych kierunków w ruchliwych obszarach miejskich, może pomóc zmniejszyć liczbę wypadków i ofiar śmiertelnych z udziałem rowerzystów. Dlatego też niniejsza praca ma na celu zbadanie rozwiązań uczenia maszynowego w celu znalezienia najdokładniejszego sposobu przewidywania trajektorii rowerzystów w warunkach miejskich. Wdrożone rozwiązanie do przewidywania trajektorii składa się z czterech głównych elementów: wykrywania obiektów (YOLOv7), śledzenia obiektów (Sort), estymatora ruchu kamery wykorzystującego punkt zanikający i przepływ optyczny oraz przewidywania trajektorii z wykorzystaniem interpolacji wielomianowej i liniowej skutkujące przewidywaniem przyszłych ramek rowerzystów w następnych 1, 3 i 10 klatkach (0,1s, 0,3s i 1s do przodu). Po wyuczeniu YOLO, testach i optymalizacji hiperparametrów uzyskano następujące wyniki: AP_{50}^{test}=0.376, $AP_{50:95}^{test}$=0.182, $Precision_{50}^{test}$=0.56, $Recall_{50}^{test}$=0.462, miara $F1_{50}^{test}$=0.507, MAE kąta=22.14 stopni oraz RMSE kąta=35.5 stopni, uśrednione dla wszystkich kroków. Użycie estymatora ruchu kamery w zadaniu estymacji kąta orientacji rowerzysty zmniejszyło MAE kąta o 2%.

Abstract

As recently cycling has experienced a significant growth in popularity, the safety of cyclists and other road users has to be addressed even more than before. Cyclists – unlike cars – are agile, relatively small. Cyclists move faster than pedestrians, and are just as vulnerable to injury as pedestrians in the event of an accident. Assisting car drivers in detecting and predicting the trajectories of cyclists, who often emerge unexpectedly from various directions in busy urban areas, could help reduce the number of accidents and fatalities involving cyclists. Therefore this thesis aimed for examining machine learning solutions to find the most accurate way to predict the trajectory of cyclists in urban conditions. A solution for Trajectory Prediction was implemented consisting

of four main elements: Object Detection (YOLOv7), Object Tracking (Sort), Camera Motion Estimator using vanishing point and optical flow and Trajectory Prediction using polynomial and linear interpolation resulting in predicting future bounding boxes of cyclists in the next 1, 3 and 10 frames (0.1s, 0.3s and 1s ahead). After the training, testing and hyperparameter optimization following results were obtained: AP_{50}^{test}=0.376, $AP_{50:95}^{test}$=0.182, $Precision_{50}^{test}$=0.56, $Recall_{50}^{test}$=0.462, F1 $score_{50}^{test}$=0.507, Angle MAE=22.14 degrees and Angle RMSE=35.5 degrees, averaged over all steps. Incorporating Camera Motion Estimator in the task of estimating the cyclist's orientation angle reduced the MAE of the angle by 2%.

Table of contents

1. Introduction

In recent years, urban areas have witnessed a significant increase in the popularity of cycling as a mode of transportation. The rise of the number of cyclists not only promotes sustainable mobility but also poses new challenges for ensuring safety for all road users. One critical aspect of enhancing safety is accurately predicting the trajectories of cyclists, which unlike pedestrians are fast, they can change direction quickly and they are smaller than cars, making it harder to detect them and predict their movement.

The difficulty in detecting cyclists within urban environments arises from a combination of factors. Firstly, the presence of various other objects, such as pedestrians, motor vehicles, and roadside infrastructure, adds to the complexity of the detection process. Additionally, the varying illumination, shadows and the occlusion caused by vehicles, buildings, or other obstacles further hampers the visibility of cyclists, making their detection even more difficult.

Furthermore cyclists, unlike cars or pedestrians are not singular objects. Cyclist object contains a person seated in a specific position and a bike below the person. Therefore there is a risk of a model detecting pedestrians and parked bikes as cyclists. What if in the training data a cyclist is partly occluded by pedestrians and the bike is not fully visible? Should the training data contain only fully visible cyclists without occlusions? Wouldn't it then lower the detection rate of occluded or truncated cyclists in the test data (as the model would detect only a fully visible cyclist). Or would it decrease the overall mAP because the model could treat the objects that occlude cyclists (like a pedestrian in front of the cyclists) as part of the cyclist object itself.

Therefore, this master thesis aims to investigate the trajectory prediction of cyclists in an urban setting by utilizing machine learning techniques. Further in this chapter, the motivation and goals of this work will be explained in more depth.

1.1. Motivation

Every year about 41000 cyclists are killed in road accidents, that is 3% of global road traffic deaths [14]. In Europe since 2010 the number of cyclist fatalities has fluctuated between 1900 and 2010 yearly [19]. The share of cyclist fatalities has increased from 7% in 2010 to 9% in 2019 of all road accidents in Europe. One of the factors could be that In contrast to other modes of transportation cycling is the only one without a decrease in fatalities number, as represented in Figure 1.1.

Serious injuries in crashes involving a cyclist also received the highest increase of all transport modes in Europe (+24%), as shown in Figure 1.2.

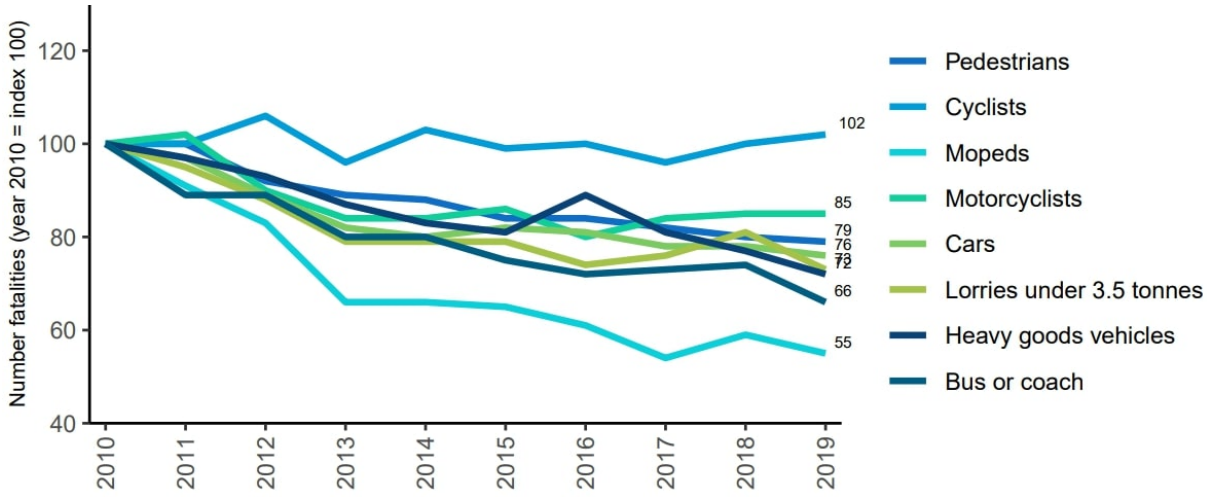


Figure 1.1. Trend of fatalities in crashes involving different transport modes in the EU27 (2010-2019). Source: European Road Safety Observatory, Facts and Figures – Cyclists - 2021 [19].

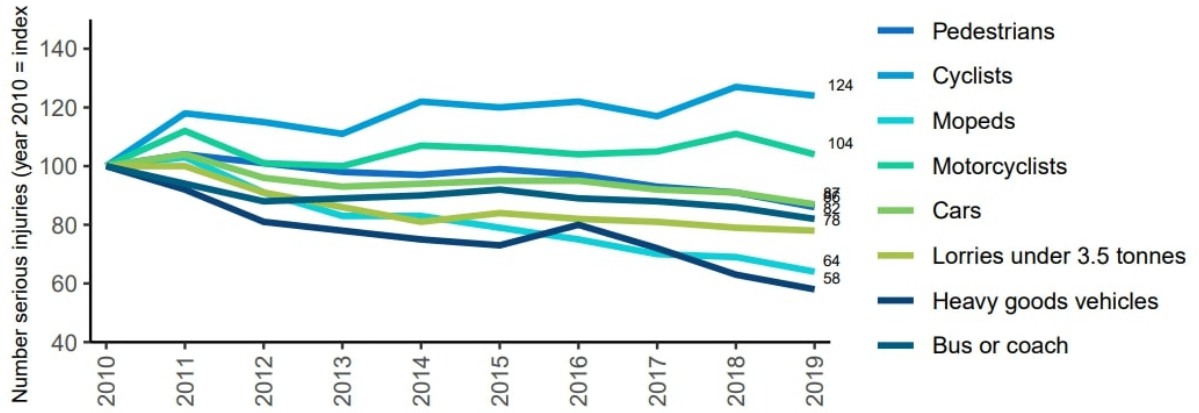


Figure 1.2. Trend of serious injuries in crashes involving buses/coaches, heavy goods vehicles and other transport modes in the EU27 (2010-2019). Source: European Road Safety Observatory, Facts and Figures – Cyclists - 2021 [19].

In traffic accidents with cyclists, cyclists are the victims in 98% of cases. Also 58% of all cyclist fatalities occur on urban roads, 41% on rural roads and less than 1% on motorways.

Between 1990 and 2015 in major cities of Western Europe, North America and South America there has been a major increase in using bikes as a mode of transportation. Strengthening of cycling infrastructure, policies and programs are some of the reasons for it (see Figure 1.3).

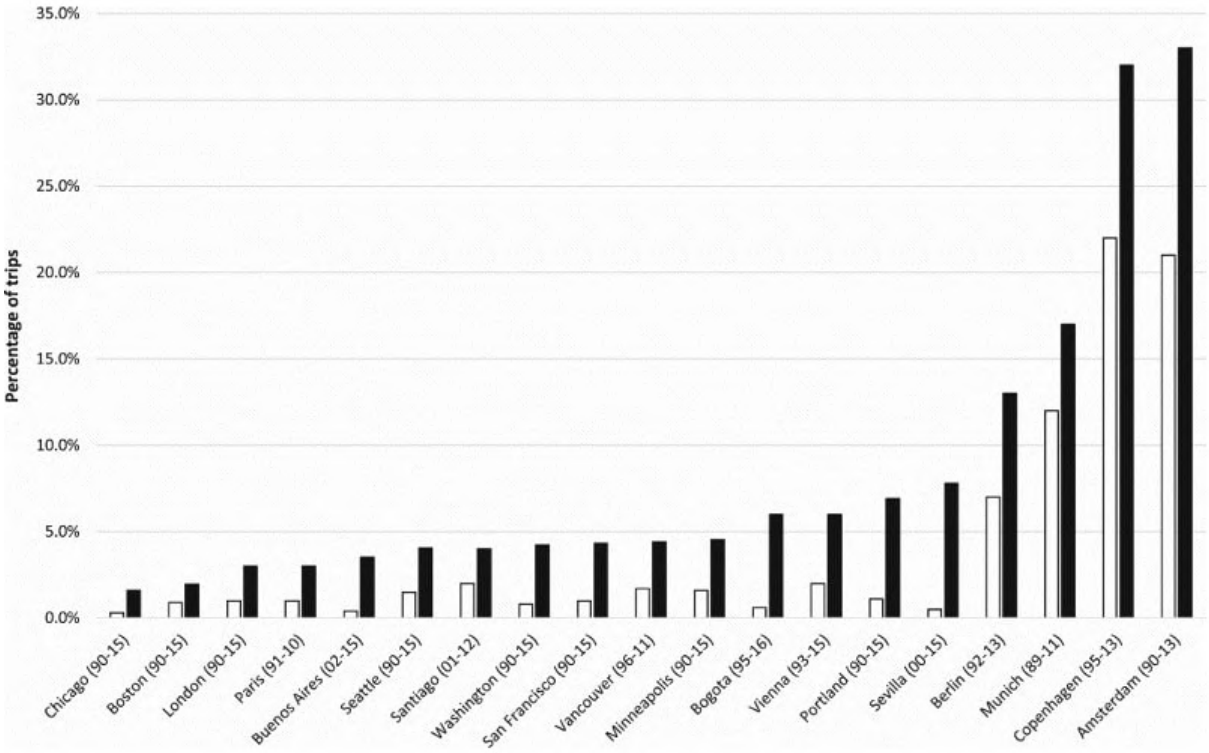


Figure 1.3. Increasing rates of bicycle use in large cities in Europe and the Americas, 1990–2015. Source: Cycling towards a more sustainable transport future [37].

In Copenhagen and Amsterdam - cities that already had high cycling levels, the rates has risen by 10% and 12% respectively. In cities that did not have a significant bike culture like Buenos Aires, Portland, Bogota and Sevilla the cycling increased over six times.

In 2015, Netherlands, over 80% of the serious injuries occurred in accidents with no motor vehicle involvement. About 70% of cyclist road deaths were a result of a collision with a passenger car, van, truck or bus (see Figure 1.4) [29]. In 24% of fatal crashes there was no motor involvement. In 19% cases there was no 3rd party involvement only a cyclist error, infrastructure or a bicycle defect.

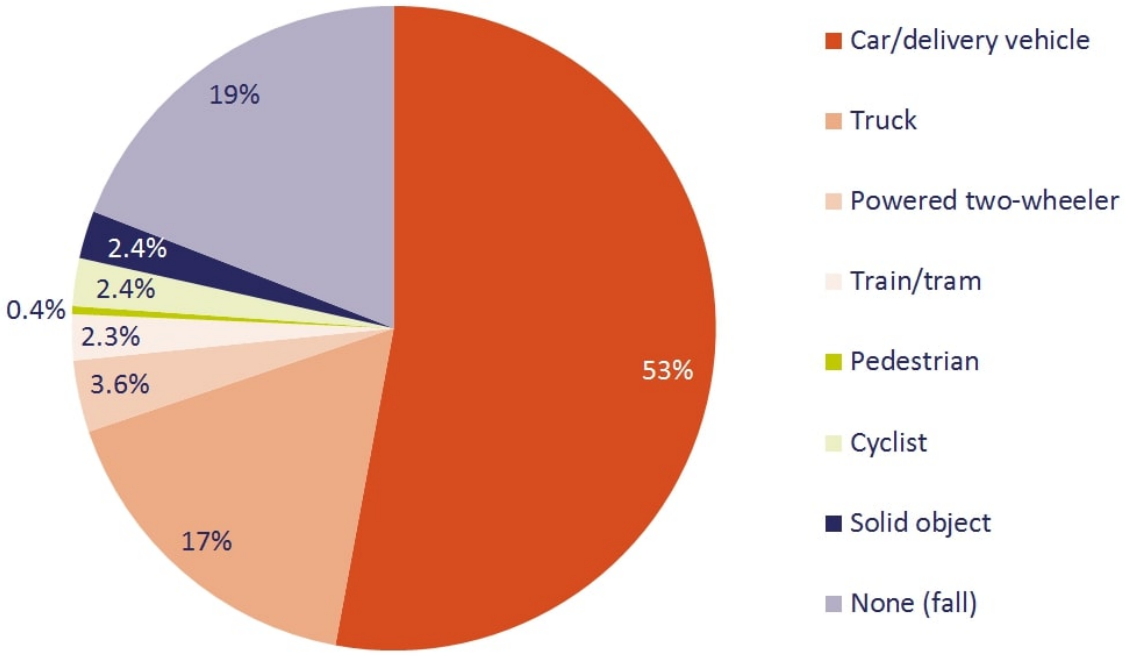


Figure 1.4. Distribution of road deaths among cyclists by crash opponent (if known) in the period 1996-2014. Source: Bicycle fatalities: trends in crashes with and without motor vehicles in the Netherlands [42].

As majority of cyclist deaths are caused by collisions with motor vehicles like cars and trucks and as cycling is becoming more popular there is a need for a solution to reduce the number of accidents involving cyclists and motor vehicles. A solution that could be used not only in autonomous vehicles but also a solution that would assist the driver in detection of cyclists.

1.2. Objectives

The research work will examine machine learning solutions to find the most accurate way to predict the trajectory of cyclists in urban conditions. This will include reviewing methods for this purpose, developing a dataset for conducting experiments, proposing and testing the prediction method and relating it to the state of the art.

1.3. Overview

In this chapter, the introduction highlighted the challenges of accurately predicting the trajectories of cyclists in urban environments. The difficulty arises from the presence of various objects, occlusions, and the unique nature of cyclist objects. The chapter emphasizes the need to address safety concerns due to the alarming statistics of cyclist fatalities and injuries. It also discusses the rise of cycling as a mode of transportation and outlines the objectives of the research, which involve exploring machine learning solutions for accurate

trajectory prediction in urban conditions. In the next chapter the literature related to this work will be reviewed.

2. Literature Review

In this chapter all the fundamental elements of this thesis will be explained in a brief way. A theoretical background necessary for understanding the final solution of predicting the trajectory of cyclists. Topics like object detection, metrics, object tracking and their existing solutions will be discussed here.

2.1. Object detection

Object detection is a task in the field of computer vision used to detect objects in images or video frames. An object detection model predicts bounding boxes that contain information on position, width, height (depending on the format) and also label (name of the objects) and probability score (or confidence score) (see Figure 2.1). Boxes with scores below a certain threshold are then filtered out.

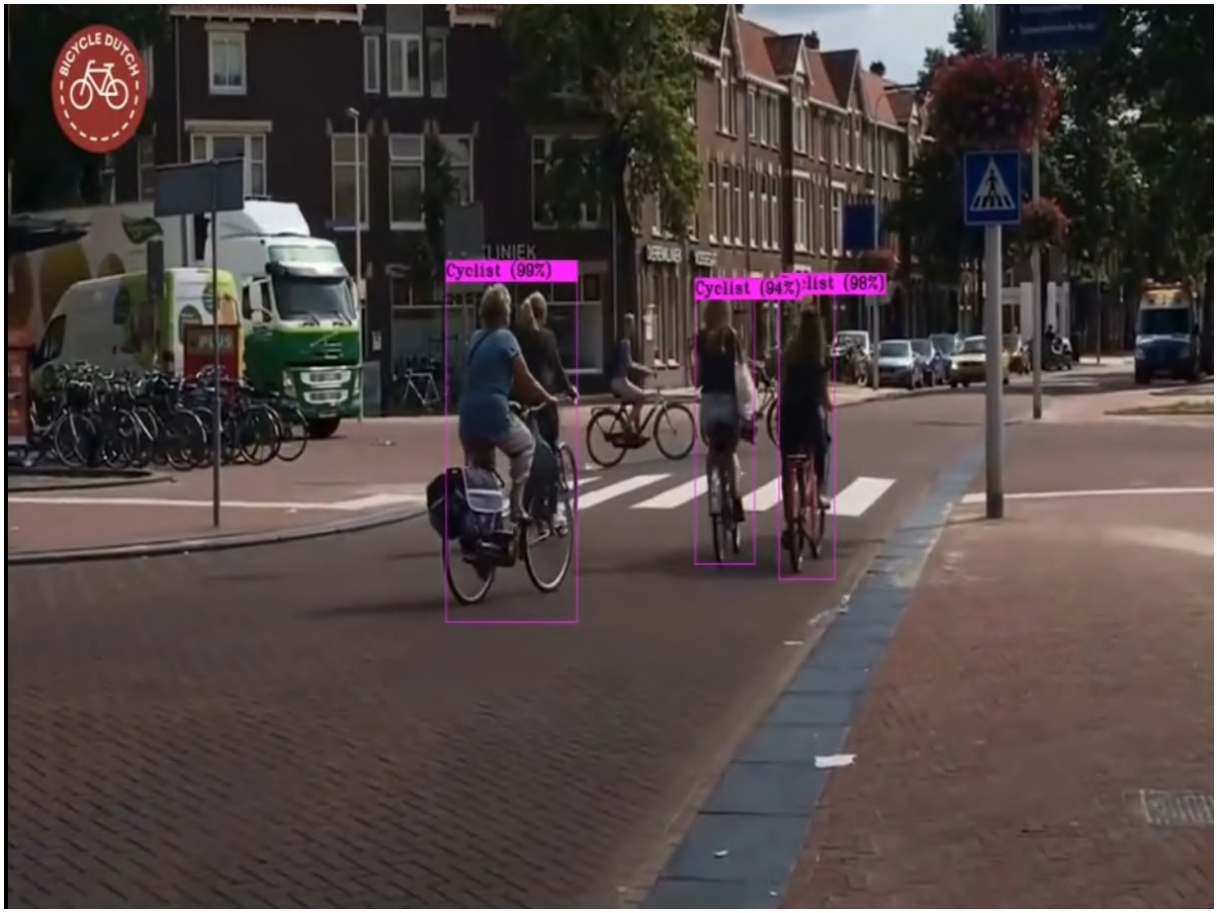


Figure 2.1. Object Detection example. Source: Dutch Cycling youtube video [4]

2.1.1. Image processing techniques and deep learning

Object detection methods could be separated into two categories: Image processing techniques and deep learning methods.

— Image processing methods - Viola–Jones [46], HOG features [16]
  - Advantages - Training data does not have to be annotated.
  - Disadvantages - Low accuracy in complex illumination (shadows), occlusion and complex images.

— Deep learning methods - R-CNN [15], Faster R-CNN [40], YOLO [39], Retina-Net
  - Advantages - More capable of overcoming illumination issues, occlusion and complex scenerios.
  - Disadvantages - Training data must be annotated (hand labeled by a human). In order to train from scratch large object detection models like YoloV4 in darknet (53 convolutional layers) large annotated datasets are needed, like in this case YoloV4 was trained on COCO dataset [33] with 330k annotated images, 1.5 million objects and 80 classes.

Even though the annotation process is tedious and quite costly as it cannot really be automated (neural network trained on annotations created by another neural network can only produce output as accurate as the network the annotations came from in the first place), it is still worth the effort, as the accuracy and speed that comes with deep learning methods is far greater than of the image processing techniques.

2.1.2. Deep learning methods

The development of pedestrian detection has been significantly advanced using deep learning methods. Deep learning architectures can be one of the following two categories: a two stages detector (region proposal approach, see Figure 2.2 (a)) or a single stage detector (non-region proposal approach, see Figure 2.2(b)). Two stage detectors have, as the name suggests, two stages. First stage is to find regions of interest (ROI) typically with edges boxes [51] (anchors) or RPN (Region proposal network). Second step is to classify the regions from previous step (CNN, SVM). Implementations of two stage detectors are: R-CNN (Region-Convolutional Neural Network [15]), Fast R-CNN [41], Faster R-CNN [40], Mask R-CNN [23]. Single stage detectors, on the other hand, directly go to classification and bounding-box regression step without a separate region proposal step. Making it more computationally efficient than two stage detectors. Examples of Single stage detectors are: SSD (Single Shot Detector [35]), YOLO (You Only Look Once [39]), YOLOR [48].

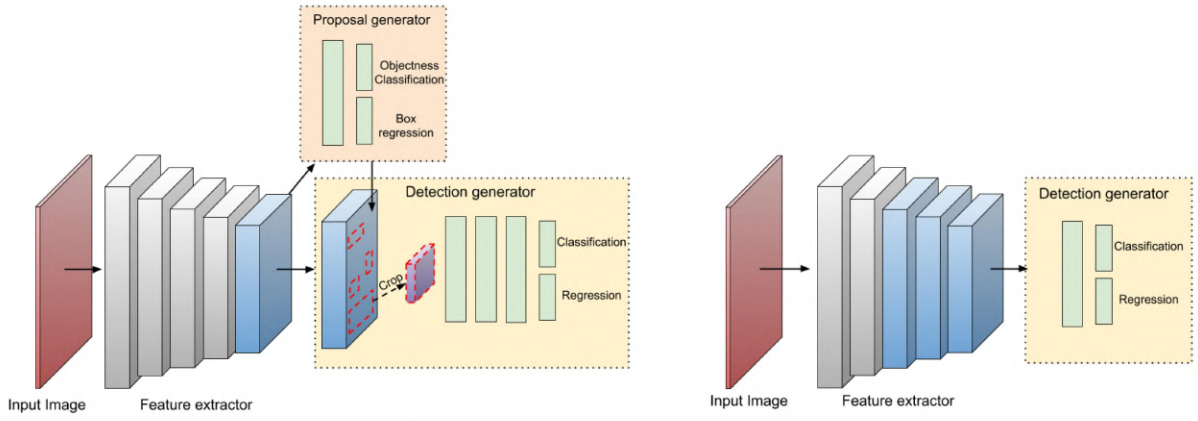


Figure 2.2. Deep learning object detection meta-architectures. Source: On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data [10].

2.1.3. Transfer learning

Default YOLOv4 model detects 80 classes (humans, cats, dogs) as it was trained on COCO dataset. Although classes can be filtered if not needed they do not disappear from the model and they still make the model unnecessarily bigger and therefore slower. Most object detection use cases are specific like road sign detection, vehicles, detecting error in packaging. In those specific topics only few classes are needed, no more, no less. One could think of training a YOLO model from scratch using a custom dataset of road signs only. The problem is that each road sign would need tens of thousands of instances in order for the model gain acceptable accuracy. It would need not only to learn the road signs, but also to understand relationships between lines, curves, shapes and colors, illumination and shadows, occlusion. Human brain has roughly 100 billion neurons and 100 trillion synapses [50] analogous to weights or parameters in neural networks. Visual cortex in primates is estimated to take around 1/3 of brain's cortical surface [32]. Different areas in a brain have different neuron and synapse density, but a rough estimate of 30 trillion synapses (weights) in relation to 12-174 million parameters (weights) of YOLOR [48] shows that computer vision is not an easy task. And the fact that researchers established methods that mimic our visual capabilities with only a fraction of resources that we possess is quite admirable.

That is why most of the deep learning object detection methods enable training custom models with transfer learning. How it works is that the model trained on millions of object instances from COCO has gained a sort of understanding of basic computer vision tasks, like detecting lines, curves, shapes, colors, imprinted in it is weights. Then only a few of layers are made mutable for the second training on a custom dataset. Not enough to teach it to see from scratch but just enough for it to use the shapes it has been trained on previously and use that in the 2nd training phase on custom data. As it is in case in

this thesis, COCO dataset has people and bike instances in it. So the default YOLOv4 model is already familiar with all of the objects and shapes that together make a cyclist, a human and a bike. In that case there is no need for 100k images with cyclists but only a few thousands is sufficient for acceptable accuracy.

It is worth to mention that such custom dataset does not have to necessarily contain similar objects to the dataset the default model was trained on, although the more similar they are the better. A company logo, a road sign, a specific pattern on a sidewalk, it does not matter, as long as there is enough variation in training data (which can be greatly helped with data augmentation) and it can be distinguished easily from other objects, it is possible to detect any object with a modern DL object detection method.

2.1.4. Metrics

In this work various metrics will be used for evaluation of the proposed method. One of them is Mean average precision (mAP), a popular metric used for object detection, including YOLO models. It could be summarised as mean AP (Average Precision) over all classes, where AP is an area below the precision/recall curve [24]. To understand all of this, first it must be explained what precision is.

		Predicted class	
		Positive	**Negative**
Actual class	**Positive**	TP	FN
	Negative	FP	TN

Figure 2.3. Confusion matrix.

Precision measures how many predictions made were actually correct (see Equation 2.1). TP stands for True Positive (see Figure 2.3), which means that a positive prediction was correct. FP (False Positive) means a positive prediction that was incorrect.

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

Recall, on the other hand, measures how well the model finds all the positive (ground truth) objects (see Equation 2.2). TP as it was established stands for a correct positive prediction and FN (False Negative) means a that the model failed to predict an object that was there in reality.

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

For object detection, precision and recall is calculated using IoU (Intersection over Union) value. As show on Figure 2.4, IoU is the overlapping area of two bounding boxes, the predicted and the real one divided over the union of those areas. For example, with IoU threshold set as 0.5, IoU of 0.8 could be classified as a True Positive, and IoU of 0.2 could be classified as a False Positive. Even though the boxes overlap slightly, the object is not really there.

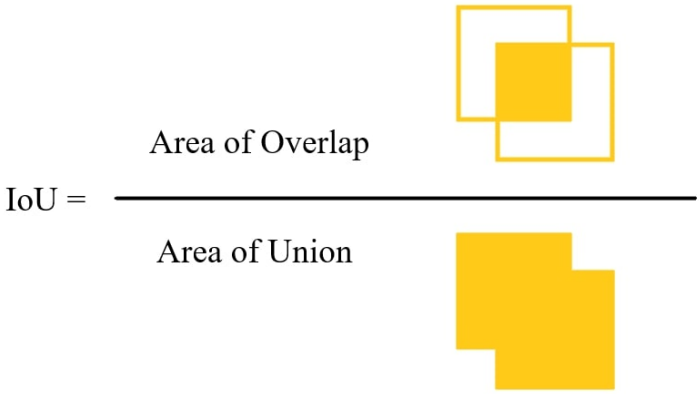


Figure 2.4. Intersection over Union.

An cyclist detection example for mAP calculation. Consider Figure 2.5 below, where the hypothetical model gave three predictions shown as red boxes, with approximate IoU values. The green boxes (bounding boxes) show all the cyclist on the image. P1 and P4 both have IoU of 0, as bikes are not cyclists. For different IoU thresholds the precisions differ:

— If IoU = 0.5 then precision is 50% (2 TP / 4 predictions) and recall is 66.67% (2 TP / 3 ground truths)
— If IoU = 0.7 then precision is 25% (1 TP / 4 predictions) and recall is 33.67% (1 TP / 3 ground truths)
— If IoU = 0.9 then precision is 0% (0 TP / 4 predictions) and recall is also 0% (0 TP / 3 ground truths)

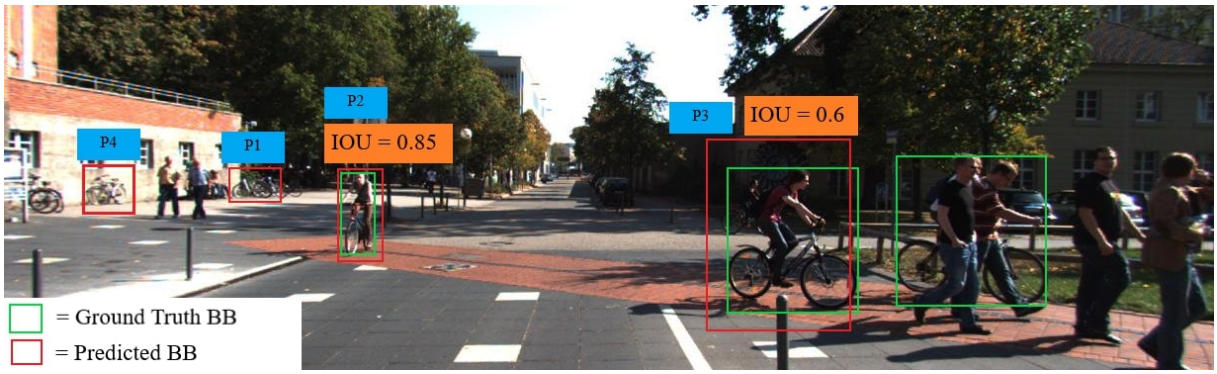


Figure 2.5. Cyclist detection example. Source: Raw image from KITTI dataset [20], annotations selfmade

In our example there is only one image, a precision-recall curve from a single image prediction would be a rectangle. The precisions and recalls from thousands of predictions could look like this Figure 2.6, although with probably different values.

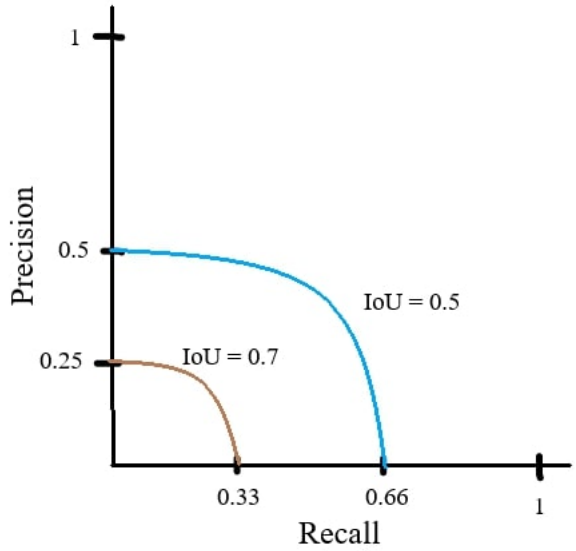


Figure 2.6. Precision-recall curve example.

Then the AP (Average Precision) could be measured as shown in Equation 2.3, where a loop goes through all precision/recalls, calculating the difference between current and last recall and multiplying it by current precision. Calculating AP for only one IoU threshold of 0.5 could be written as AP@0.5. Another example: AP@[0.5:0.95] (or AP50:95) means that AP was calculated with min IoU as 0.5, max IoU 0.95 and a step of 0.05.

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k) \tag{2.3}$$

where:

$$\begin{aligned} Recalls(n) &= 0 \\ Precisions(n) &= 1 \\ n &= \text{number of IoU thresholds} \end{aligned}$$

As AP applies to only one class, mAP is a measure of AP for all classes. When model classifies objects of one class only then mAP = AP, therefore both terms could be used interchangeably.

To give a different perspective on precision and recall, F1 score will also be used for evaluation of the proposed trajectory prediction method. F1 score is a harmonic mean of precision and recall. The intuition behind it is to 'punish' when one of the variables is close to 0, assigning a low F1 score, which can be seen as the blue edges on the plane on Figure 2.7.

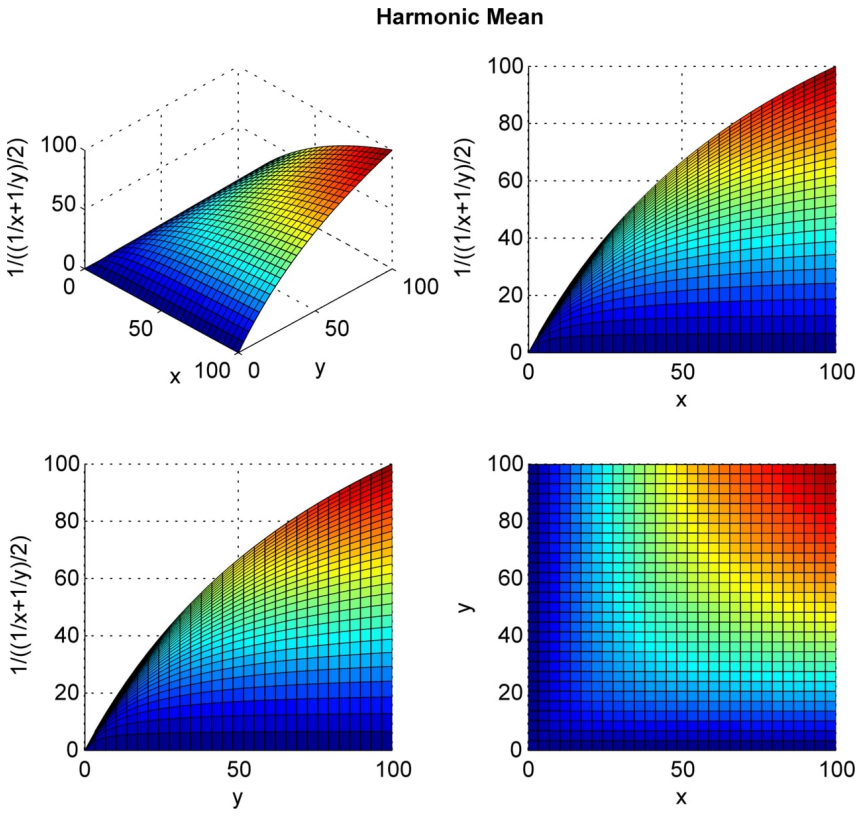


Figure 2.7. Normalised harmonic mean plot where x is precision, y is recall and the vertical axis is F1 score, in percentage points. Source: WikiMedia [2]

F1 score has a following Equation 2.4:

$$Recall = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.4}$$

For a better control over which metric is more important: precision or recall, Equation 2.5 is an extended version:

$$Recall = (1 + \beta^2) * \frac{Precision * Recall}{(\beta^2 * Precision) + Recall} \tag{2.5}$$

If both metrics are equally important, then $\beta = 1$, if precision were to be valued more then β should be in range [0;1], in other case if recall is more important then $\beta > 1$

Example:

- $\beta = 0.5$ - precision is more important
- $\beta = 2$ - recall is more important
- $\beta = 1$ - precision and recall are equally important.

In this work a simplified version of the F1 score was utilized with $\beta = 1$.

2.2. SORT

Object tracking algorithm utlized in this work is SORT [3] - simple, online, and realtime tracking of multiple objects in a video sequence. SORT uses detections from an object detector (in this work it is YOLO) as input, then with Kalman Filter [27] and Hungarian algorithm [31] to associate detections and ultimately return the tracked the objects. SORT algorithm consists of four main steps:

1. Detection - an external object detector has to provide bounding boxes with scores of detected objects - authors of SORT used [40], in this work a newer model will be utlized from the YOLO family
2. Estimation Model - based on the Kalman Fitler, this module is responsible for updating the state of the tracked object, represented in Equation 2.6. Where u and v represent position, s and r scale and aspect ratio and $\dot{u}$, $\dot{v}$, $\dot{s}$ are their derivatives. When a detection is passed, based on the bounding box position shift and score the state components are recalculated by the Kalman Filter resulting in an updated tracker state.
3. Data Association - In assigning detections to tracked objects the assignment cost matrix with IOU distance between detections and predicted bounding boxes from existing tracked objects is computed and solved by the Hungarian algorithm. As the IOU distance favours detection of similar scale it evades the potential issue of occluded detection of smaller scale disrupting the tracking. In this phase also IOU threshold is employed, a minimal value that rejects assignments with IOU overlap lower than the threshold.
4. Creation and Deletion of Track Identities - This module is responsible for creating and deleting tracked objects. For creation an object has to be detected a few times (min hits hyperparameter), with IOU higher than the IOU threshold, before it is to be tracked. Then if the tracked object leaves frame or it has not received any detection in the last frames (max age / T_{Lost} hyperparameter) it is deleted from the tracked objects list.

Figure 2.8. Result of Sort object tracking on recording 0015 of KITTI Object Tracking Dataset [20]

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T \tag{2.6}$$

Figure 2.8 shows an example output of Sort object tracking using detections from a YOLOv7x model.

2.3. Trajectory Prediction

There are multiple approaches to trajectory prediction each depending on the setup and requirements. Is the camera mounted on a moving vehicle like a car or is it stationary, positioned a few meters above the street like a surveillance camera? Is there a need for predicting trajectories for a specific object? What kind of a object is it? How fast does the solution needs to be? Having those questions answered one can begin to choose the proper technique for the task.

2.3.1. Algorithmic approach using Optical Flow and Kalman filter

If the camera is stationary, then it does not necessarily have to utilise object detectors and object trackers to predict their trajectories. For a surveillance camera, if there is no need for detecting specific objects using just Optical Flow and Kalman Filter maybe just enough for the task [34, 8].

2.3.2. Pose Based intent estimation

As cyclist often become visible to approaching vehicles shortly before they are in the collision range an approach that does not need past trajectories to estimate the cyclist direction would be of great value. Pose based approaches predict the intention of cyclists by estimating 3D human pose, achieving the direction estimation earlier [30].

2.3.3. Social GAN

Tennis ball served at Wimbledon, rock falling on a slope or a TV thrown out of the window. All these motions are computable. Some are easier, some harder due to multiple collision, uneven surface and other external factors, but knowing all of those factors it is possible to calculate the objects trajectory. Humans, however can change direction with

intent. This intent cannot be easily calculated or predicted, not for a computer at least. Humans are quite adept at it, we are constantly simulating other peoples minds in our heads, so to traverse through the environment quickly and without unnecessary stops or collisions.

To account for human intent in predicting future trajectories multiple GAN papers have tried to tackle the issue. One of the 1st papers was: Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks [21]. In Figure 2.9 the architecture of the solution is presented. The model consists of three main components:

- **Generator** consists of an encoder and decoder. Encoder takes for input past trajectories and encodes them. Decoder takes for input pooled encoded trajectories and generates future trajectories
- **Pooling module** takes for input the encoded trajectories and outputs a pooled trajectory vector for each person
- **Discriminator** takes for input real and fake trajectories and classifies them as socially acceptable or not

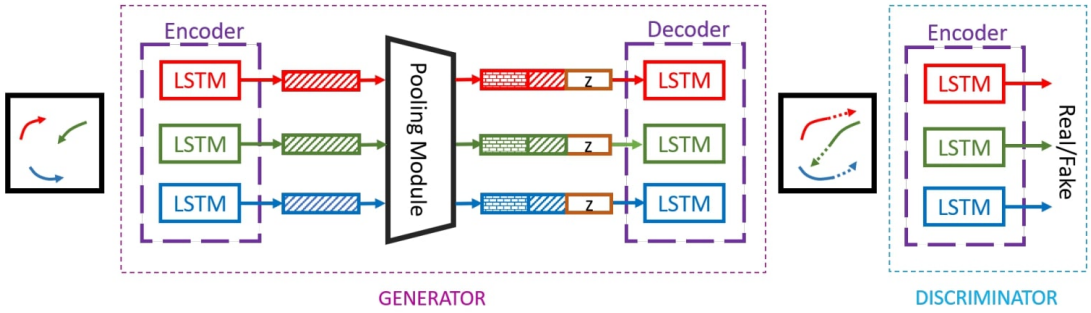


Figure 2: System overview. Our model consists of three key components: Generator (G), Pooling Module, and Discriminator (D). G takes as input past trajectories X_i and encodes the history of the person i as H_i^t. The pooling module takes as input all $H_i^{t_{obs}}$ and outputs a pooled vector P_i for each person. The decoder generates the future trajectory conditioned on $H_i^{t_{obs}}$ and P_i. D takes as input T_{real} or T_{fake} and classifies them as socially acceptable or not (see Figure 3 for PM).

Figure 2.9. System overview of Social GAN. Source: Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks [21].

Although this paper focused on human trajectories, there are other works focusing also on vehicles like TraPHic [11] or Convolutional Social Pooling for Vehicle Trajectory Prediction [17]. Though mind that these solutions do not handle the object detection and tracking, only the trajectory prediction.

2.3.4. RobustTP

An end-to-end algorithm for predicting future trajectories of cyclists could be constructed from three main elements: Object Detection, Object Tracking and a type of a Social GAN for predicting trajectories in a dense urban environment that takes the interactions between objects into account. RobustTP [13] proposed these three elements: YOLO for

object detection, DeepSort for object tracking and LSTM-CNN. A state-of-the-art solution achieving 0.96/1.53 (ADE/FDE - average displacement error and full displacement error in meters) in 3s step and 1.29/1.97 in 5s step on the TRAF dataset [12].

2.4. Conclusion

In this chapter various object detection methods were presented, both the image processing techniques and the ones utilising deep learning, with the latter being more suitable for the task. Several state-of-the-art deep learning object detectors were discussed, falling into one of the two categories: Two stage dectors and Single stage detectors. The difference between these two is that Two stage detectors separate the detection into two steps: finding regions of interest (ROI) and then classifying the proposed regions from previous step using CNN, SVM or any other model (R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN). Single stage detectors on the other hand perform the classification and bounding-box regression without the separate region proposal step, yielding in better performance of the detector (SSD, YOLO).

Training an object detector from scratch on a custom dataset without pretrained weights requires significant time and data. Authors usually publish pretrained weights of their models, enabling transfer learning and achieving good results with fewer data.

Evaluation metrics such as precision, recall, AP:50, AP50:95, and F1 score were explained and illustrated.

SORT (Simple, online, and realtime tracking) used for cyclist tracking in this work was presented. It is an object tracking algorithm utilising Kalman Filter and Hungarian algorithm.

Several techniques that could aide in trajectory prediction of cyclists were presented. Algorithm ones relying solely on Optical Flow and Kalman Filter. 3D human pose estimations for orientation prediction, this approach could be translated to cyclist either. LSTM based networks that take trajectories as inputs and outputs socially acceptable trajectories taking the interactions and human intent into account, like Social GAN. Also a end-to-end solution was exhibited - RobustTP that uses YOLO for object detection, DeepSort for object tracking and a custom LSTM-CNN for trajectory prediction.

In the next chapter methodology of this work will be showcased.

3. Methodology

This chapter will provide an explanation of the methodology employed in this thesis. It encompasses the datasets used data preprocessing, cyclist detection, cyclist tracking and trajectory prediction. Figure 3.1 displays the overview of the proposed method.

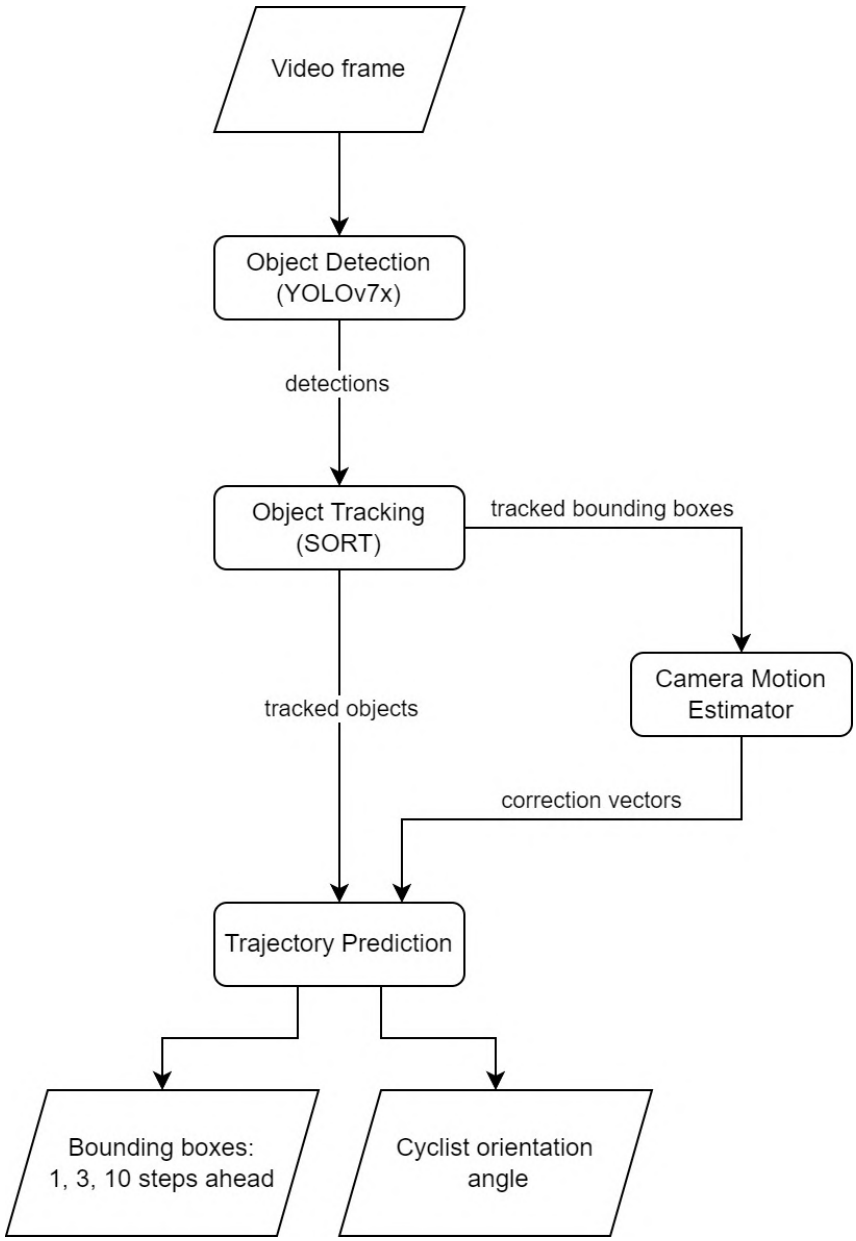


Figure 3.1. Overview of the architecture used in this work for predicting cyclist trajectory.

3.1. Datasets

Unlike pedestrians or vehicles, cyclist annotations are not so popular among the datasets used for object detection. Only two object detection datasets were found that contain cyclist instances. That is the KITTI Vision Benchmark [20] (KITTI Object Tracking Dataset) and Tsinghua-Daimler Cyclist Detection Benchmark [1].

3.1.1. KITTI Object Tracking Dataset

The KITTI Vision Benchmark encompasses a wide array of data, including stereo, optical flow, depth, visual odometry, and 2D and 3D object detection and tracking. All of the data were collected in Germany, specifically in and around the city of Karlsruhe, using two cameras, a Velodyne laser scanner, and GPS. The KITTI Vision Benchmark comprises multiple datasets, each serving a specific purpose. While the KITTI Object Detection Dataset provides images and annotations, the KITTI Object Tracking Dataset offers ordered images and annotations divided into separate recordings. The division into separate recordings is crucial for accurate testing of the proposed solution, thus making it the chosen dataset for further investigations. The specification of the KITTI Object Tracking Dataset is as follows:

- Number of recordings: 21
- Image resolution: 1224x370
- Total number of images: 8008
- Number of images with annotated cyclists: 1412
- Number of cyclist annotations: 1938
- Recording framerate: 10 FPS

Each object annotation comes with information presented in Table 3.1).

As previously mentioned, KITTI Object Tracking Dataset consists of 21 recordings with varying cyclist annotation distribution (see Table 3.2). Some recordings, like recording 15, contain over 25% of all annotations, 10 of out 21 recordings contain no cyclists, but most recordings stay in the range of 50-300.

Values	Name	Description
1	frame	Frame within the sequence where the object appears
1	track id	Unique tracking id of this object within this sequence
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries. Truncation 2 indicates an ignored object (in particular in the beginning or end of a track) introduced by manual labeling.
1	occluded	Integer (0, 1, 2, 3) indicating occlusion state: 0 = fully visible, 1 = partly occluded, 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-π..π]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x, y, z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-π..π]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better

Table 3.1. Annotation format in KITTI Object Tracking Dataset

Recording	Cyclist Detections	Images with Cyclists
0000	154	154
0001	0	0
0002	75	75
0003	0	0
0004	60	49
0005	139	139
0006	0	0
0007	0	0
0008	0	0
0009	0	0
0010	14	14
0011	0	0
0012	41	41
0013	237	162
0014	0	0
0015	537	361
0016	272	130
0017	101	93
0018	0	0
0019	308	194
0020	0	0
Total	**1938**	**1412**

Table 3.2. Distribution of cyclists over recordings in KITTI Object Tracking Dataset

3.1.1.1. Occluded Cyclists

As the images of the KITTI Dataset were captured while driving around the mid-size city of Karlsruhe, it is not surprising that a notable number of objects in the dataset are occluded. Approximately 36% of the detected cyclists are occluded, with 31% experiencing partial occlusion and 5% encountering significant occlusion (refer to Table 3.3). Although the boundaries between these categories are somewhat flexible, they provide a reasonable indication of whether an object can be utilized in the training process or not.

Recording	Total	fully visible	partly occluded	largely occluded	unknown
0000	154	154	0	0	0
0001	0	0	0	0	0
0002	75	0	75	0	0
0003	0	0	0	0	0
0004	60	39	9	10	2
0005	139	139	0	0	0
0006	0	0	0	0	0
0007	0	0	0	0	0
0008	0	0	0	0	0
0009	0	0	0	0	0
0010	14	14	0	0	0
0011	0	0	0	0	0
0012	41	35	6	0	0
0013	237	224	13	0	0
0014	0	0	0	0	0
0015	537	127	376	33	1
0016	272	184	64	24	0
0017	101	38	45	18	0
0018	0	0	0	0	0
0019	308	287	6	15	0
0020	0	0	0	0	0
Total	**1938**	**1241**	**594**	**100**	**3**

Table 3.3. Occluded cyclists in KITTI Object Tracking Dataset

The Figure 3.2 illustrates various occlusion states of cyclists: fully visible, partly occluded, largely occluded and unknown. Fully visible and partly occluded cyclists are suitable for training, whereas 100 cases of largely occluded and 3 cases of unknown appear to provide excess of confusing information, which in the best-case scenario could slow down the training process and, at worst, decrease the overall validation and test mAP.

Figure 3.2. Examples of cyclist occlusion in the KITTI dataset, from left to right: fully visible, partly occluded, largely occluded, unknown.

3.1.1.2. Truncated Cyclists

As it has been mentioned in Table 3.1, there are three types of truncations in this dataset: non-truncated [0], truncated [1] and unknown [2]. Truncation stands for an annotation of objects that are in the process of leaving image boundaries.

Figure 3.3. Context for Figure 3.4

In Figure 3.4, three consecutive annotations of a cyclist object were marked by KITTI annotators as non-truncated [0], whereas in Figure 3.5 annotations were marked as truncated [1]. In the Table 3.4 the images and annotations of truncated cyclist counts over all of the KITTI recordings are displayed.

Another example of non-truncated annotations in Figure 3.7 and truncated ones in Figure 3.8, where the last truncated annotation (frame 40) is in 90% empty space.

Figure 3.4. Non-truncated annotations from recording 0016. Annotations are from the frames: 6, 7, 8, counting from left to right. In Figure 3.3 is the full frame for context.

Figure 3.5. Truncated annotations from recording 0016. Cyclist is moving out of the left bottom side of the frame. Annotations are from the frames: 9, 10, 11, counting from left to right.

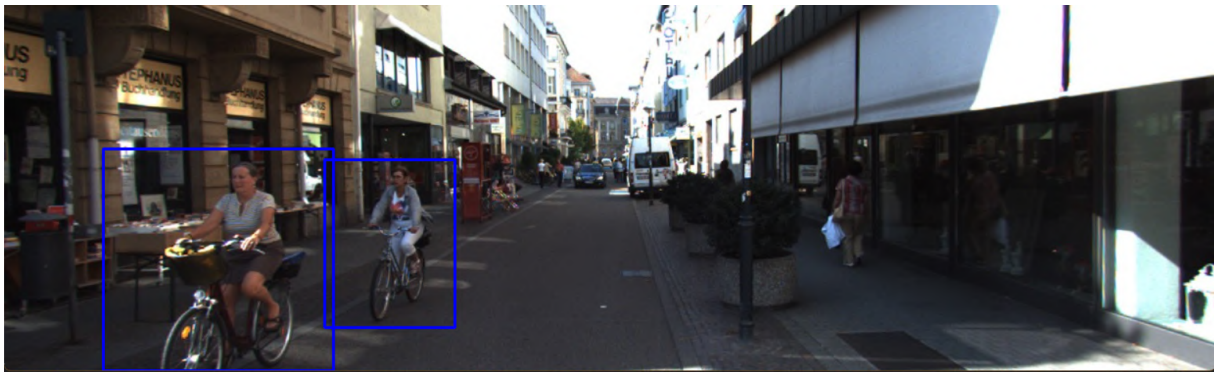


Figure 3.6. Context for Figure 3.7

Figure 3.7. Non-truncated annotations from recording 0019. Annotations are from the frames: 35, 36, 37, counting from left to right. In Figure 3.6 is the full frame for context.

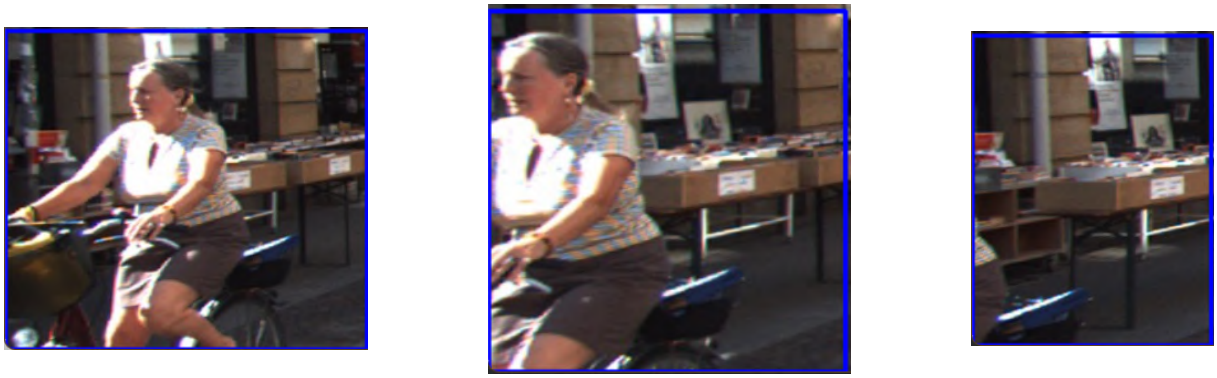


Figure 3.8. Truncated annotations from recording 0016 where cyclist is moving out of the left bottom side of the frame. Annotations are from the frames: 38, 39, 40, counting from left to right.

Figure 3.9. Context for Figure 3.10

Figure 3.10. Truncated annotations from recording 0002 where cyclist is stationary on the right side and goes out of the frame as the car is moving forward. Annotations are from the frames: 135, 141, 145, 146 counting from left to right. In Figure 3.9 is the full frame for context.

Recording	Non-truncated	Truncated	Unknown
0000	154	0	0
0001	0	0	0
0002	63	12	0
0003	0	0	0
0004	56	4	0
0005	138	1	0
0006	0	0	0
0007	0	0	0
0008	0	0	0
0009	0	0	0
0010	13	1	0
0011	0	0	0
0012	38	3	0
0013	230	7	0
0014	0	0	0
0015	530	7	0
0016	258	14	0
0017	101	0	0
0018	0	0	0
0019	279	29	0
0020	0	0	0
Total	**1860**	**78**	**0**

Table 3.4. Truncated cyclists in KITTI Object Tracking Dataset

The example of truncated annotations in Figure 3.10 is more severe than in Figure 3.4 or Figure 3.7. In previous cases, most truncated annotations retained visibility of over 50% for the cyclist object. However, in this scenario, all truncated annotations exhibit visibility between 10% and 50%, showing only a part of the bike with no person riding it. Such data is not suitable for training and may cause the object detector to falsely classify bikes or people as cyclists.

3.1.2. Tsinghua-Daimler Cyclist Detection Benchmark

The Tsinghua-Daimler Cyclist Benchmark provides a benchmark dataset for cyclist detection [1]. The dataset contains 22161 annotated cyclist instances in over 30000 images, recorded from a moving vehicle in the urban traffic of Beijing (see Table 3.5).

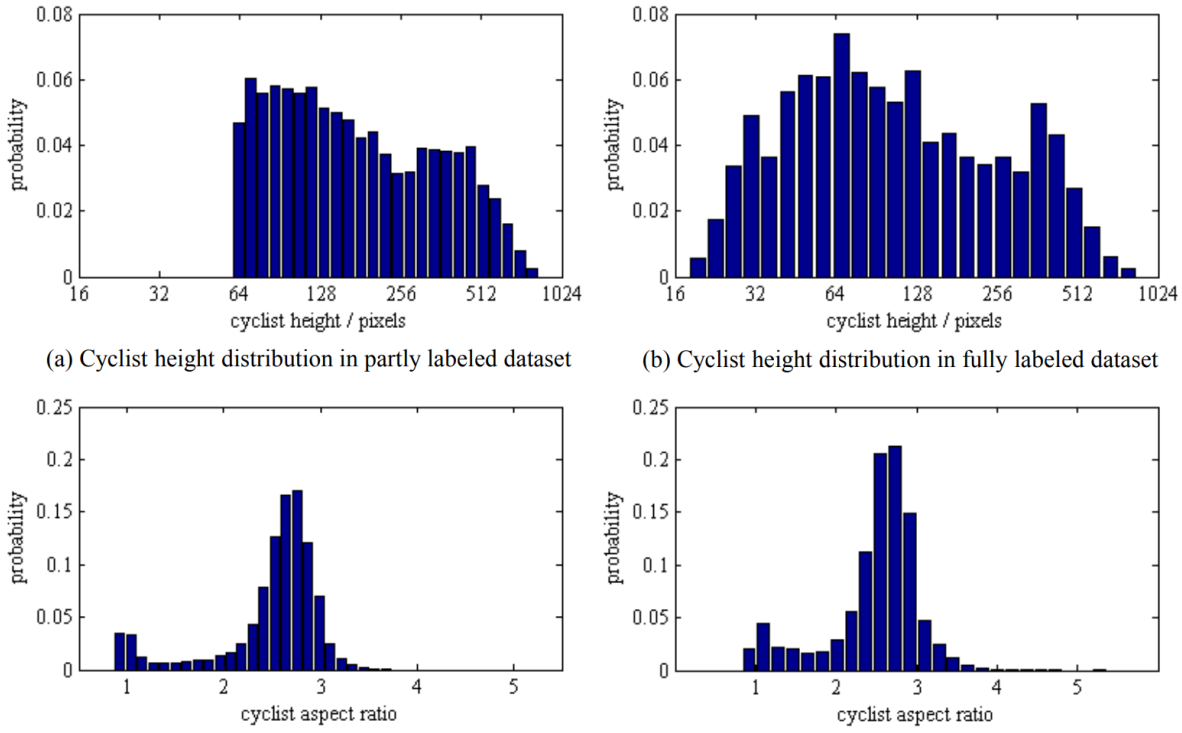
Dataset Statistics	Training	Validation	Test	Non-VRU	**Total**
image frames	9741	1019	2914	1000	**14674**
cyclist BBs	16202	1314	4657	0	**22173**
pedestrian BBs	0 (ignored)	1541	7401	0	**8942**
other rider BBs	0 (ignored)	190	1105	0	**1295**
# total BBs	**16202**	**3045**	**13163**	**0**	**32410**

Table 3.5. Dataset statistics of Tsinghua-Daimler dataset. Source: Tsinghua-Daimler Cyclist Detection Benchmark

The dataset consists of four subsets (copied from the Tsinghua-Daimler website [1]):

- Train Usually used for training, contains 9741 images with annotations only for "cyclist". Only cyclists which are fully visible (occlusion<10%) and higher than 60 pixels have been labeled here.
- Valid 1019 images to be used for validation of hyper parameters. Annotations for ("pedestrian", "cyclist", "motorcyclist", "tricyclist", "wheelchairuser", "mopedrider"). Only objects higher than 20 pixels have been labeled here.
- Test 2914 images normally used for testing with annotations for ("pedestrian", "cyclist", "motorcyclist", "tricyclist", "wheelchairuser", "mopedrider"). Only objects higher than 20 pixels have been labeled here, not occluded more than 80% and not truncated more than 50%.
- NonVRU 1000 images in which no object of interest ("pedestrian", "cyclist", "motorcyclist", "tricyclist", "wheelchairuser", "mopedrider") is present.

In Figure 3.11 the distribution charts of bounding box width, heights and aspect ratios across partly labeled dataset (training) and the fully labeled dataset (valid and test) is presented.



(a) Cyclist height distribution in partly labeled dataset

(b) Cyclist height distribution in fully labeled dataset

(c) Cyclist aspect ratio distribution in partly labeled dataset

(d) Cyclist aspect ratio distribution in fully labeled dataset

Figure 3.11. Cyclist scale distribution of the Tsinghua-Daimler dataset. Source: Tsinghua-Daimler Cyclist Detection Benchmark [1].

Figure 3.12. Training frames with missing annotations due to max 10% occlusion threshold. Source: Tsinghua-Daimler Cyclist Detection Benchmark [1].

Figure 3.13. Test frames with annotations with max occlusion threshold = 80%, though in some rare cases cyclists seem to be occluded up to 90%. Source: Tsinghua-Daimler Cyclist Detection Benchmark [1].

3.1.2.1. Dataset overview

In the training samples Figure 3.12 quite a significant number of cyclists are not annotated due to a quite restrictive max occlusion threshold = 10%. Perfectly recognizable objects that are not annotated in the dataset may be detrimental to an object detector model which learns not only where the cyclists are in the image, but also where they are not. It'd be quite difficult to learn to detect a fully unobstructed cyclist and not detect cyclists with 80% or less occlusion, because both share similar features. For some object detectors it may not matter significantly, but it probably would for YOLO.

Test samples in Figure 3.13 have most of the occluded cyclists annotated, according to the Tsinghua-Daimler dataset authors, up to 80%, though in same rare cases it would appear up to 90%. Which could also give a difficult time for YOLO to train (if the test dataset would be used for training). Despite those rare occlusion cases, test and valid dataset are quite accurate. In all Tsinghua-Daimler samples bounding boxes seem to encapsulate the cyclist more accurately than KITTI does, which is an another advantage. Objects smaller than 60px could prove difficult for YOLO to train on. Removing small objects would reduce the number of frames from 2914 to 2046 and number of labels from 4657 to 1751, which would still make a bigger training dataset than the one from KITTI (in that case the small Tsinghua-Daimler validation set could be used as a test dataset).

3.1.3. Summary

KITTI tracking dataset was chosen for further tests, because unlike Tsinghua-Daimler dataset it provides cyclists orientations which could be used for both better trajectory

prediction and for cyclist orientation angle estimation. Apart from that Tsinghua-Daimler could still prove usefull for training the YOLO object detector as it has more cyclist annotations than KITTI. Even if the occluded cyclists with missing annotations in the Tsinghua-Daimler training dataset would cause low mAP, then still the valid and test sets do not suffer from that issue. Unfortunately the quality of the Tsinghua-Daimlers dataset valid and test sets has been noticed too late and there was no time to properly utilize the Tsinghua-Daimler dataset in this work.

3.2. Dataset preprocessing

Upon deciding on how to prepare the dataset for training an object detector model, it is important to decide which data may prove useful for learning and which detrimental. YOLO authors in the README of the YOLO Darknet repository on Github [7] give a few insights on how to achieve better mAP:

- Increase network resolution (as it was done in this paper for YOLOv7 model - 640x640, after initialy having 416x416 for input in YOLOv4)
- Check that each object is mandatory labeled in the dataset (that was the reason for discarding the Tsinghua-Daimler dataset initially)
- For each desired object for detection there must be at least 1 similar object in the training dataset with about the same: shape, side of object, relative size, angle of rotation, tilt, illumination (fortunately data augmentation is built-in both into Darknet's YOLOv4 and YOLOv7 which helps with the case of insufficient amount of cyclist data in KITTI dataset)
- Preferably there should be 2000 different images for each class or more (as a result of data preprocessing and train/test splitting the number of images with cyclists dropped from 1412 to 1116 images. That's assuming authors meant images and not annotations, which there are 1457 in the train/valid set, see Figure 3.14)
- It is desirable that training dataset includes images with non-labeled objects - use as many as there are images with objects (in this paper it was done as recommended, 1116 images with cyclists and 1116 without cyclists, 2232 total)

Figure 3.14 presents the data pipeline that has transformed the raw KITTI tracking dataset into train/valid and test splits. For summary of the number of bounding boxes and images refer to Table 3.6. The pipeline was implemented as follows:

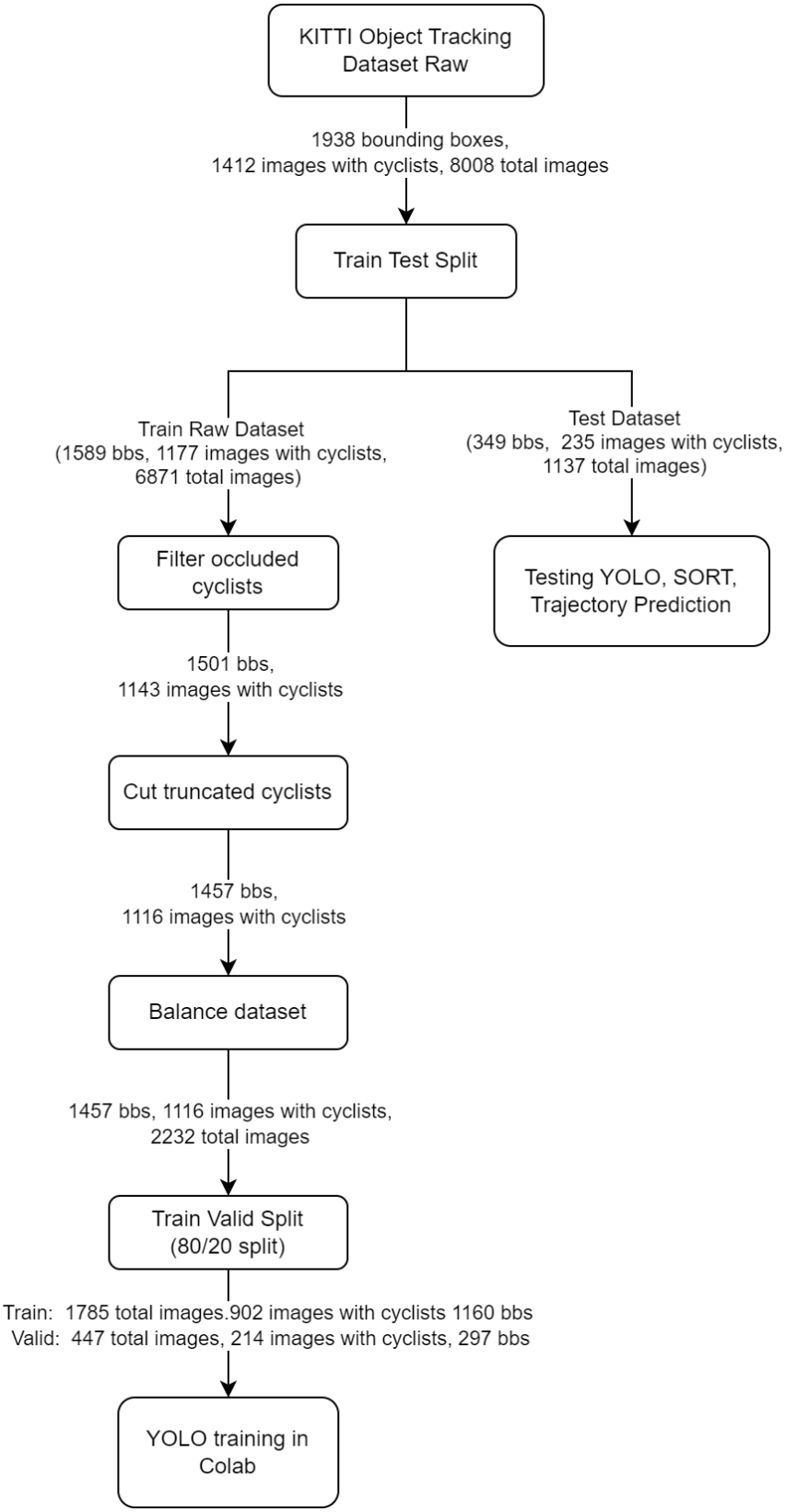


Figure 3.14. KITTI Object Tracking dataset preprocessing pipeline

Data preprocessing pipeline steps:

1. **Train Test Split** - The Raw KITTI Object Tracking Dataset was split into train and test sets. The test set utilized recordings 0012 and 0019, which accounted for 19% of the total bounding boxes. These recordings were selected due to the diverse range of cyclists they contained and the presence of a moving camera in recording 0019, which was crucial for testing the Camera Motion Estimator.
2. **Filter Occluded Cyclists** - As demonstrated in Figure 3.2, extensively occluded cyclists [2] and a few unknown [3] annotations were filtered from the training set. This filtering was carried out to prevent any disruption to the training process, following the recommendations provided in Darknet's README [7].
3. **Cut Truncated Cyclists** - As depicted in: Figure 3.10, Figure 3.5, and Figure 3.8, numerous truncated cyclists were suitable for training, while others were not. Similar to the extensively occluded cyclists, if a cyclist was visible in only 25%, it was deemed insufficient for training as there would not be enough information to differentiate the cyclist from a person or a bike. The following possibilities were considered:
   A. Removal of labels with truncation set to 1.
   B. Removal of all frames containing truncated labels.
   C. Modification of labels with truncation set to 1 by replacing them with blacked-out regions in the frame. This approach mitigates data loss compared to Option B and eliminates unwanted annotations (examples in Figure 3.15 and Figure 3.16).
   D. Assuming that truncated cyclists pose no significant impact on the results, leaving all truncated labels in the training/validation set.
   E. Manual screening of poorly truncated labels, retaining the accurate ones.

   Options A and D are considered unsuitable as they do not effectively address the issue and result in the retention of unwanted data in the training set. While Option E is likely to yield the best results, it involves a time-consuming process. Consequently, the only options left are: Option B and C. Option B would resolve the issue, but it would also involve the removal of valid cyclist annotations that coexist in the same frames as the truncated ones. This loss cannot be afforded given the limited training data available. On the other hand, Option C removes the truncated annotations and keeps the other non truncated objects from the same frames in dataset, ensuring data integrity without unnecessary loss. Thus, **Option C is the course of action** that has been chosen.
4. **Balance the Dataset** - This step, also recommended by the YOLO authors [7], involves ensuring an equal number of images with cyclists and images without cyclists in the dataset. Initially, in an attempt to compensate for the lack of data, all of the empty and unlabeled frames were used for training. However, this approach not only unnecessarily extended the training process but also resulted in worse test

results. As a result, it was decided to balance the labeled and unlabeled frames equally.

5. **Train Valid Split** - In this step, the preprocessed dataset was randomly divided into training and validation sets, with approximately 80% of the images allocated to the train set and approximately 20% to the validation set.

Figure 3.15. Example 1: the first image illustrates a non-truncated cyclist on the left and a slightly truncated cyclist on the right. The second image demonstrates the outcome of removing the truncated cyclist from the frame and eliminating the corresponding annotation from the training set. In this instance, removing the truncated cyclist was not necessary since the cyclist was mostly visible. However, one can either choose to remove all truncated cyclists or manually select the good and bad ones.

Steps	Total images	Images with cyclists	Cyclist bounding boxes
KITTI Raw Dataset	8008	1412	1938
Train Raw Dataset	6871	1177	1589
Filter occluded cyclists	6871	1143	1501
Cut truncated cyclists	6871	1116	1457
Balance dataset	2232	1116	1457
Train	**1785**	**902**	**1160**
Valid	**447**	**214**	**297**
Test	**1137**	**235**	**349**

Table 3.6. Dataset preprocessing pipeline data

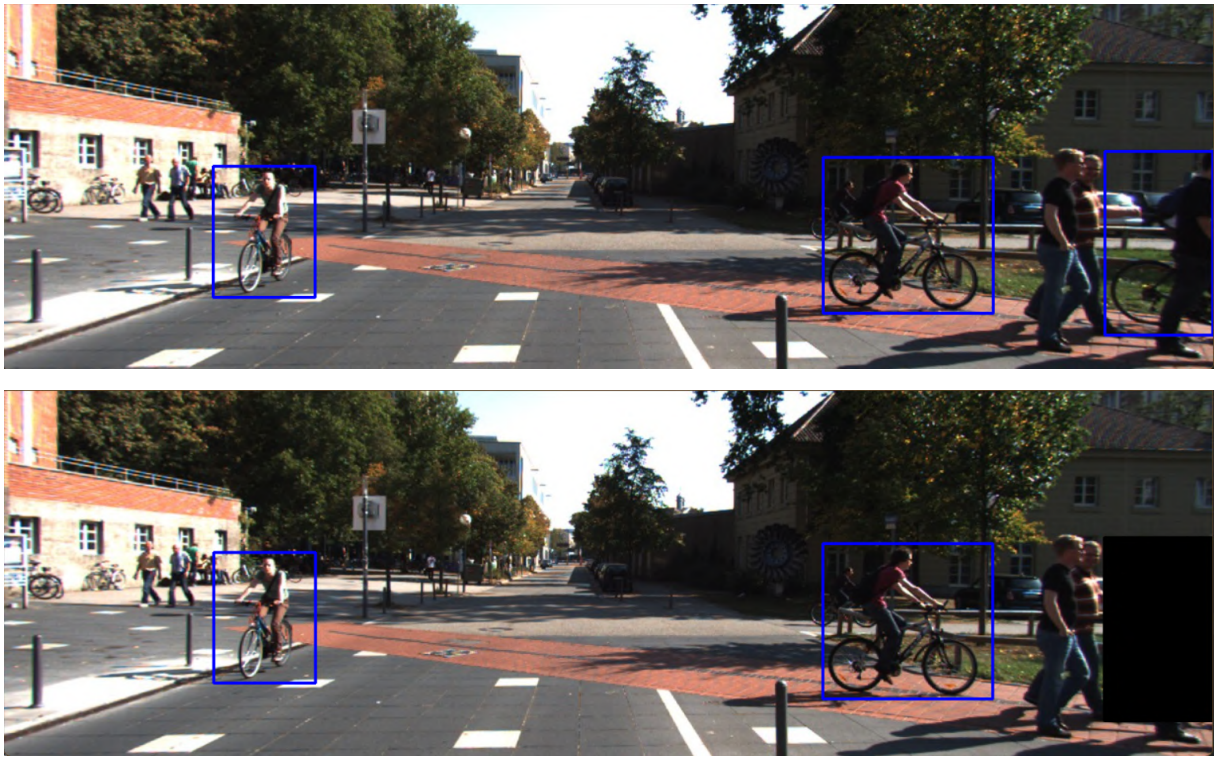


Figure 3.16. Example 2: Similarly, the first image depicts an unchanged frame containing three cyclist objects. However, the cyclist on the right is severely truncated, making him indistinguishable from a bicycle. Consequently, the removal of this cyclist from the frame is deemed necessary. In the second image, the truncated cyclist has been successfully removed from the frame, without losing any other valuable annotations.

3.3. Cyclist detection

Initially, the YOLOv4 model implemented in Darknet was utilized for object detection in this paper. However, in the summer of 2022, YOLOv7 was released, offering higher resolutions, improved mean Average Precision (mAP) in faster inference speeds compared to its predecessors. Performance comparisons with traditional RCNN family models are shown in Figure 3.17. Additionally, Figure 3.18 demonstrates the enhanced speed and accuracy of YOLOv7 compared to YOLOv5, which is a PyTorch port of YOLOv4 with additional improvements contributed by individuals not directly associated with the original YOLO authors [26]. Despite this, YOLOv5 gained popularity in recent years, partly due to its ease of setup, training, and integration that brings PyTorch. Whilst setting up Darknet often takes hours if not days of troubleshooting on Windows platforms (in Google Colab and Linux it is a matter of minutes).

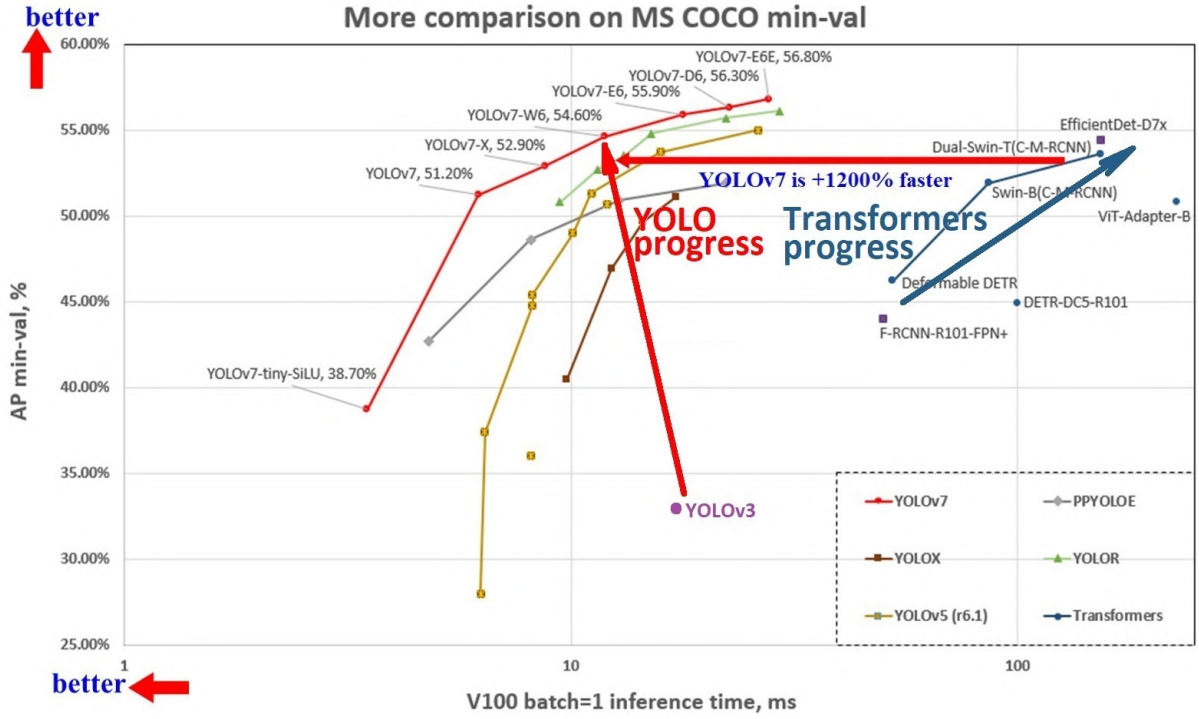


Figure 3.17. Comparison of YOLOv7 with other real-time object detectors. Source: YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors [47].

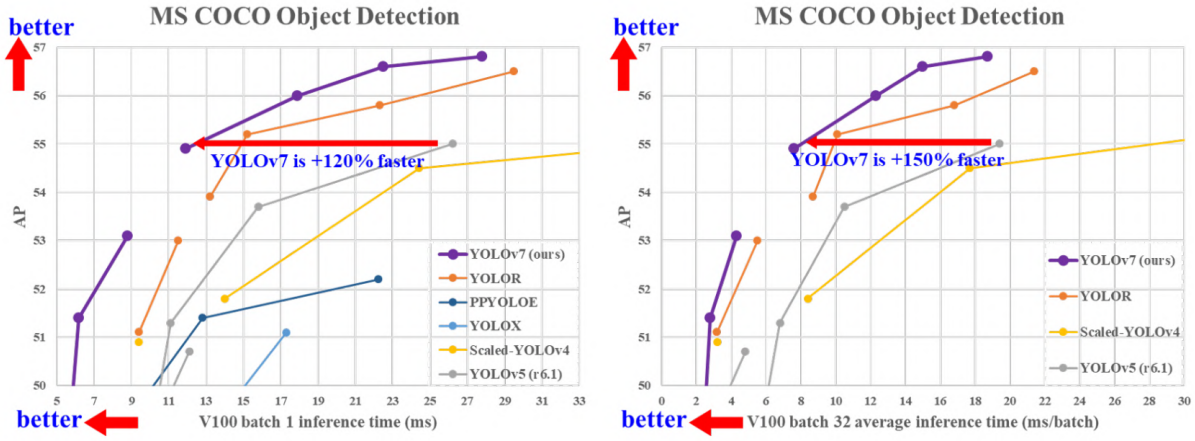


Figure 3.18. Comparison of YOLOv7 with other YOLO object detectors. Source: YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors [47].

In the beginning of 2023, the authors of YOLOv5 released YOLOv8 [25]. It is worth noting that the comparison they provided on GitHub Figure 3.19 does not include all the YOLOv7 model sizes (as the authors of YOLOv5 and YOLOv8 did not publish research papers). The comparison shows improvements in both mean average precision (mAP) and speed compared to YOLOv5. However, it is challenging to discern the specific improvements over YOLOv7 due to the lack of detailed descriptions for each data point in the figure. Nonetheless, they did provide a performance table with their results (see Table 3.7).

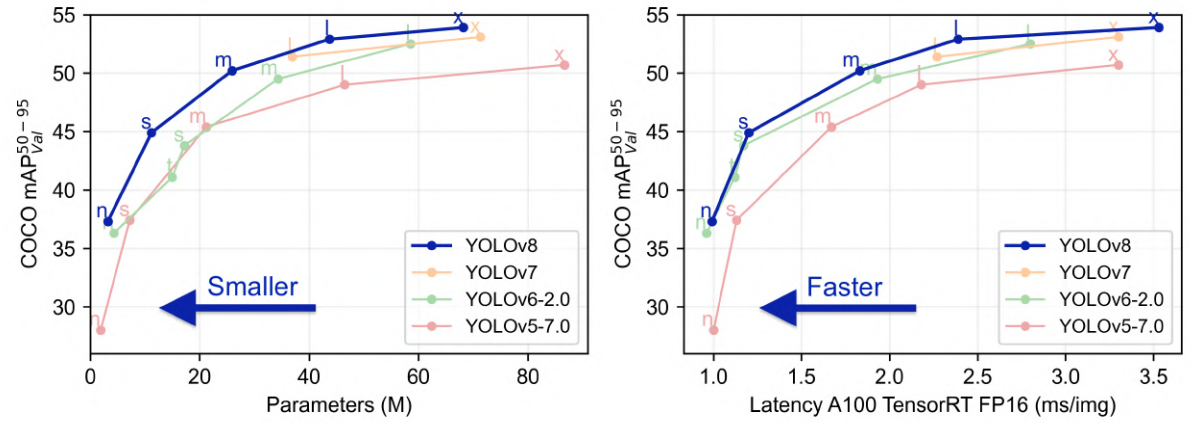


Figure 3.19. YOLOv8 comparison to the other object detectors from YOLO family. Source: Ultralytics YOLOv8 Github repository [25]

Examining the results presented by the authors of YOLOv7 in Table 3.8, the parameter numbers reveal that YOLOv7 (36.9 million) and YOLOv7x (71.3 million), with corresponding $AP^{val}_{50:95}$ values of 51.2 and 52.9 respectively. These values align with the 'l' and 'x' models depicted in Figure 3.19. Comparatively, YOLOv8l achieves an $AP^{val}_{50:95}$ of

Model	Size	$mAP^{val}_{50:95}$	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Table 3.7. YOLOv8 Model Performance Metrics. Source: Ultralytics YOLOv8 Github repository [25].

52.9, while YOLOv8x achieves an $AP^{val}_{50:95}$ of 53.9, indicating a slight improvement in AP at slightly reduced speeds.

Model	#Param.	FLOPs	Size	FPS	APtest / APval
YOLOv5-N (r6.1) [23]	1.9M	4.5G	640	159	- / 28.0%
YOLOv5-S (r6.1) [23]	7.2M	16.5G	640	156	- / 37.4%
YOLOv5-M (r6.1) [23]	21.2M	49.0G	640	122	- / 45.4%
YOLOv5-L (r6.1) [23]	46.5M	109.1G	640	99	- / 49.0%
YOLOv5-X (r6.1) [23]	86.7M	205.7G	640	83	- / 50.7%
YOLOv7-tiny-SiLU	6.2M	13.8G	640	286	38.7% / 38.7%
YOLOv7	36.9M	104.7G	640	161	51.4% / 51.2%
YOLOv7-X	71.3M	189.9G	640	114	53.1% / 52.9%
YOLOv5-N6 (r6.1) [23]	3.2M	18.4G	1280	123	- / 36.0%
YOLOv5-S6 (r6.1) [23]	12.6M	67.2G	1280	122	- / 44.8%
YOLOv5-M6 (r6.1) [23]	35.7M	200.0G	1280	90	- / 51.3%
YOLOv5-L6 (r6.1) [23]	76.8M	445.6G	1280	63	- / 53.7%
YOLOv5-X6 (r6.1) [23]	140.7M	839.2G	1280	38	- / 55.0%
YOLOv7-W6	70.4M	360.0G	1280	84	54.9% / 54.6%
YOLOv7-E6	97.2M	515.2G	1280	56	56.0% / 55.9%
YOLOv7-D6	154.7M	806.8G	1280	44	56.6% / 56.3%
YOLOv7-E6E	151.7M	843.2G	1280	36	56.8% / 56.8%

Table 3.8. YOLOv7 comparison to other YOLO family object detectors (shortened). Source: YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors [47].

The code snippet below (see Listing 3.1) presents the function used to calculate the best.pt weights. Which can be simplified into Equation 3.1, where both mAPs are calculated on validation dataset and with default confidence threshold of 0.001.

$$Fitness = mAP@0.5 * 0.1 + mAP@0.5:0.95 * 0.9 \tag{3.1}$$

```
def fitness(x):
    # Model fitness as a weighted combination of metrics
    # weights for [P, R, mAP@0.5, mAP@0.5:0.95]
```

```
w = [0.0, 0.0, 0.1, 0.9]
return (x[:, :4] * w).sum(1)
```

Listing 3.1. Fitness function used in YOLOv7 for determining the best.pt weights

Summary: YOLOv4, YOLOv5, YOLOv7, and YOLOv8 models are excellent real-time state-of-the-art object detectors, offering higher accuracy and speed compared to the RCNN family models. Considering that YOLOv4 and YOLOv7 were published in research papers, they were chosen as a more solid foundation for this paper, rather than the unpublished YOLOv5 and YOLOv8.

3.4. Cyclist Tracking

The Cyclist Tracking module utilized in this study is based on SORT (Simple Online and Realtime Tracking), a state-of-the-art algorithm for 2D multiple object tracking. SORT enables the tracking of cyclist positions and bounding boxes across frames, which is essential for trajectory estimation. Minimal modifications were made to SORT for the purpose of storing the history of bounding boxes and the last known orientation angle of the cyclists within the trackers. The input to the Cyclist Tracking module consists of bounding boxes with scores predicted by YOLO, and it produces trackers with bounding box history and the most recent orientation angle (see Figure 3.20).

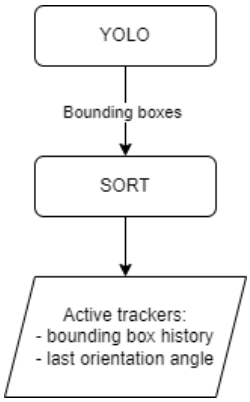


Figure 3.20. Cyclist Tracking module diagram

3.5. Cyclist Trajectory Prediction

The cyclist trajectory prediction module consists of two sub-modules: Trajectory Prediction and Camera Motion Estimation. The Trajectory Prediction sub-module is responsible for predicting the bounding boxes of cyclists in the next 1, 3, and 10 frames, as well as estimating the orientation angle of the cyclists. This is accomplished by utilizing the motion correction vectors obtained from the Camera Motion Estimator sub-module (Figure 3.21).

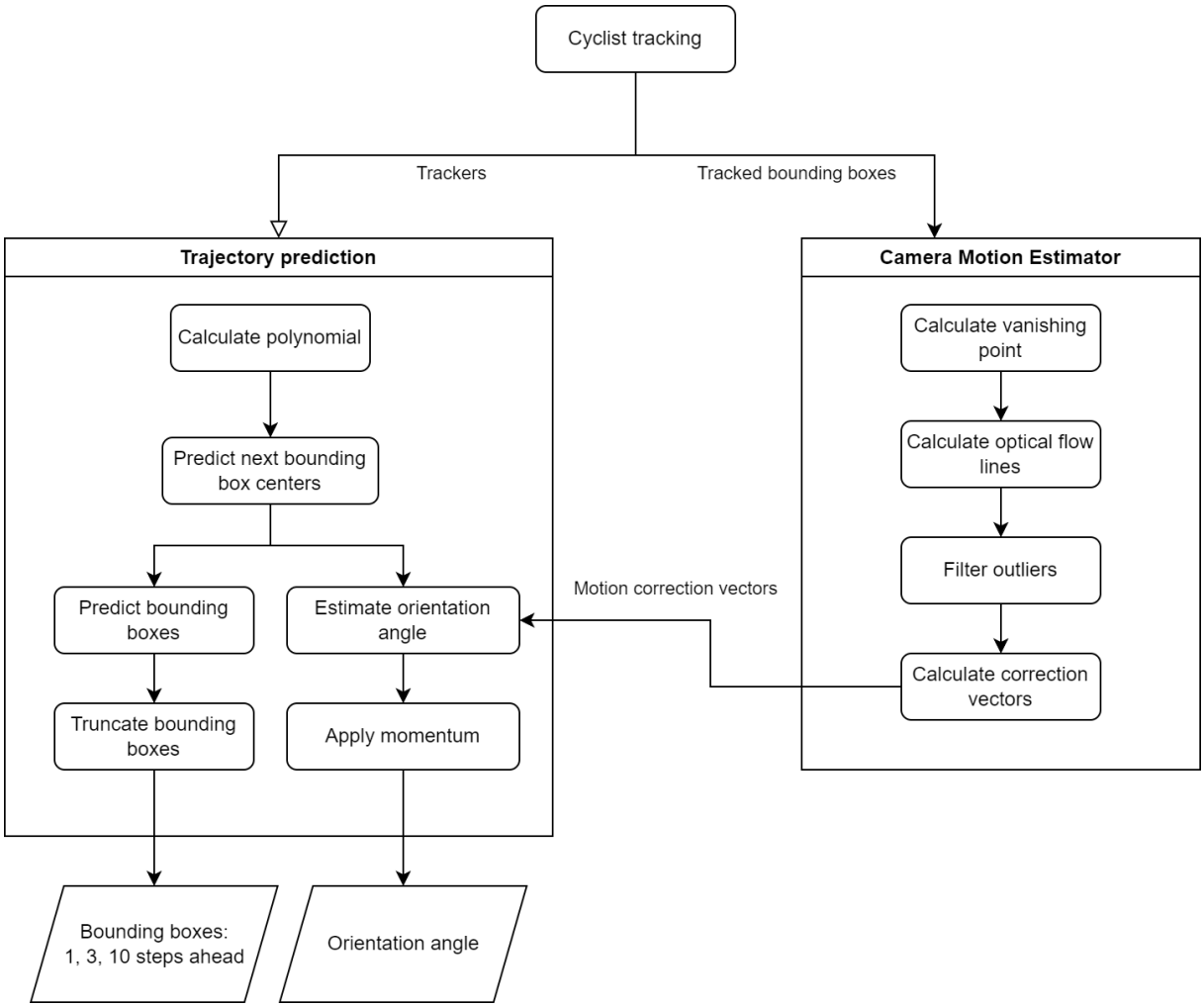


Figure 3.21. Architecture of Cyclist Trajectory Prediction module made of the Trajectory Prediction itself and the Camera Motion Estimator.

3.5.1. Camera Movement Estimator

The estimation of camera movement is not necessarily needed for predicting the bounding boxes in future frames for a specific cyclist, because as the car with the camera is moving either forward, left or right, YOLO detects the cyclists each frame, SORT tracks them and Trajectory Prediction module is simply aligning the future bounding boxes on the line created by the recent history of tracked bounding boxes. Incorporating camera motion correction vectors in the opposite direction of camera movement would likely decrease the mean Average Precision (mAP) rather than increase it. Given the constraints of a 2D, single image, and a moving camera, which introduces imperfections, it is important to note that while humans can easily consider variables such as cyclist's orientation and speed, their own direction and speed, and based on that they predict which object will be where at which time step in a 3D space, it is a challenging task to reconstruct the entire 3D environment having only one 2D camera for an input (humans possess two cameras of their own which allows them for better depth estimation).

It is a relatively straightforward task to predict the future positions of a cyclist if he is moving across the frame, from left to right and the camera is stationary. Then his movement on the frame, if his speed is constant, is also constant. But as the camera moves through the environment it adds motion to the objects on the frame that is not their own motion anymore. It is a combination of these two motions that is causing the complication. That is what this paper tries to achieve. To separate those two motions to two separate vectors in order to get to the real motion of the cyclist.

Despite the potential failure to improve cyclists orientation angle in the presence of a moving camera, this paper aims to serve as a solid foundation for future researchers.

3.5.1.1. Calculate vanishing Point

The first step shown in Figure 3.21 is Calculating the vanishing point. The point to which all lines present on the image converge and cross each other, resulting in a center horizon point. It will be used for various reasons: first, it will help us discard all optical lines below the horizon line, as those lines are too noisy to be used (moving people, cars). And secondly it will help with estimation the direction of the correction vector in the case when camera is moving forward - towards the vanishing point, or backwards - away from the vanishing point.

It is important to note that the shifting of objects in the frame due to camera movement does not solely originate from the estimated vanishing point, but rather from the direction in which the camera is heading. To illustrate this concept, let us consider an example where a car has a camera positioned at the front, perfectly aligned with the car's intended direction. Initially, if the vanishing point is determined from the edges of a long, straight road, the objects in the frame will appear to shift relative to that vanishing point. However, if the car veers slightly to the left, causing the vanishing point to shift to the right, the objects ahead (such as trees, posts, lamps, and people) will appear to shift in the frame not from the estimated vanishing point but from the center point towards which the car is now heading. In this scenario, the camera is aligned with the car, making that center point coincide with the center of the frame. However, relying solely on perfect camera alignment may introduce inaccuracies. Additionally, cars typically stay within their lanes, and when they deviate, they initiate a turn. In such cases, left or right turn signals are activated, and optical line vector orientations are utilized for the correction vectors. Henceforth, vanishing point will be considered as the center towards which the car is heading and the origin of the optical lines in reality as the same thing.

As previously mentioned a camera mounted on a car can move in roughly four ways, because of that, movement towards or away from the vanishing point (center) affects each object on the frame differently. In Figure 3.22 the relationship between the camera motion is presented, the objects and their change in position on the frame. When the camera turns left, then every object on the frame shifts right. Analogically when camera turns

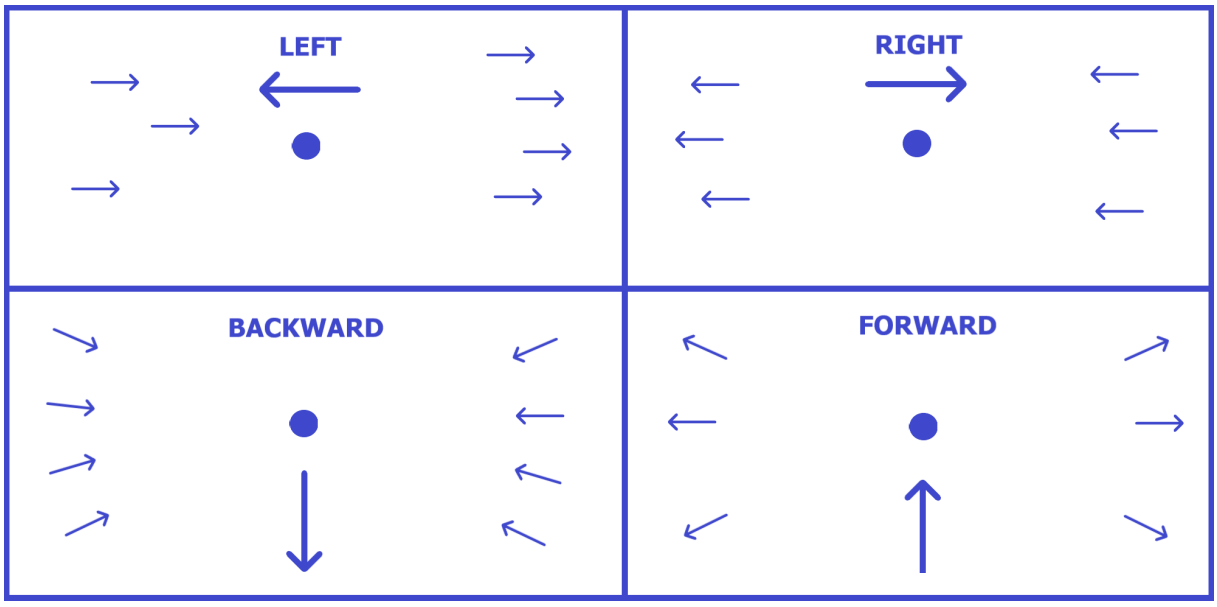


Figure 3.22. Display of how camera motion affects the change in position of objects in the frame.

right, the objects shift left. However, when the camera is moving forward, then all of the objects that once were closer to the frame center (the vanishing point), get swept to the edges of the frame (mind that the vanishing point does not have to be at the center, if the camera would point slightly more to the ground then it would be placed higher on the frame image). The vanishing point estimation algorithm used in this paper is based on VanishingPoint Github repository [38] with a few modifications.

Estimating vanishing point algorithm:

1. Perform Canny Edge Detection on the frame [9].
2. Run probabilistic HoughLine function for estimating lines on the image (see Figure 3.23).
3. Calculate the average vanishing point where all of the Hough lines are crossing each other [18].
4. Add bottom and top margins for the vanishing point (left and right are necessary is in this paper the x position of the vanishing point is locked in the width/2, with only y being movable).
5. Average the position of the vanishing point across past frames, so it is less vulnerable to the outlying estimates.

3.5.1.2. Calculate optical flow lines

OpenCV proves invaluable in simplifying the implementation of the essential algorithms for computing optical flow lines. The algorithm can be described as follows:

1. Convert the RGB frame to grey image to reduce computation.

Figure 3.23. Display of Hough Lines (blue lines), pointing to a vanishing point (red circle)

2. Determine strong corners on the image that can easily be distinguished and tracked across frames. OpenCV offers goodFeaturesToTrack() function that does just that, taking points from the last frame as input and outputting the tracked ones in the same order. Function goodFeaturesToTrack() under the hood is using the Harris corner detector [22].
3. Calculate optical lines with calcOpticalFlowPyrLK() - OpenCV function based on the iterative Lucas-Kanade method with pyramids [8].

In Figure 3.24 and Figure 3.25, the results of the algorithm can be observed. On the Figure 3.25 the resemblance of the optical lines can be observed as in the sketch in Figure 3.22, where the camera is moving forward.

Figure 3.24. Example 1: optical flow

3.5.1.3. Filter outliers

In this step the outliers get removed from the optical lines calculated in the previous step. The algorithm for it is as follows:

1. Calculate standard deviation for all optical line lengths
2. Remove optical lines in which the difference between it's length and the average length is higher than two standard deviations.

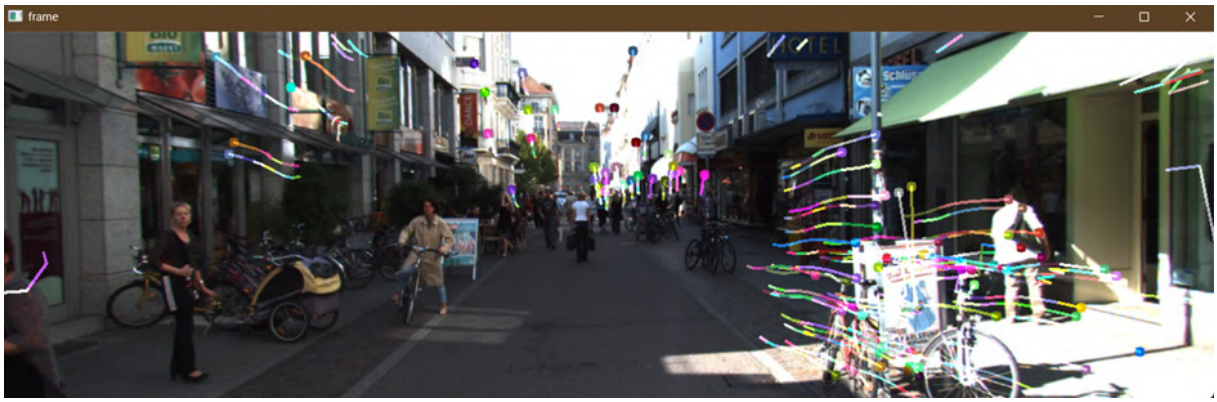


Figure 3.25. Example 2: optical flow

The results of the algorithm can be viewed in Figure 3.26, where the optical lines below the vanishing point has already been filtered. By discarding the vectors based on their length difference higher than 2*standard deviation, only the most outlying vectors are removed. Filtering by the angle difference instead of the length itself could also be an option, but accidentally the outliers could align with the correct optical lines, so the length seems to be better indicator for that particular case.

Figure 3.26. Figure displays the removal of the outliers (red) from the optical lines (green are correct).

3.5.1.4. Calculate correction vectors

The algorithm for calculating the correction vectors:

1. Calculate average optical line vectors for the left and right side separately.
2. Calculate whether the camera is moving left, right, forward, backward or if it moving at all, refer to Figure 3.22.
3. Calculate correction vectors:
   a. If the camera movement is either left or right, then the correction vector is a inverse of the average optical line vector from that side of the frame.

b. If the camera movement is either forward or backward, then the direction of the correction vector is calculated from the vanishing point to the center of the cyclist's bounding box. The length of the vector is then calculated based on the cyclist position, when cyclist is at the left or right edge of the frame the length of the correction vectors equals 200% of the average optical line vector on that side. If the cyclist is at the center of the image, then his correction vector is equal 0. At 1/4 and 3/4 of the width is 100%. The reason for that is the closer to the center an object is, smaller the change in it's position due to the camera motion.

In Figure 3.27 is the result of the algorithm. The correction vector was marked as a yellow arrow at the bottom of the cyclist's bounding box. Because the camera has turned right, the correction vector in that case is simply the inverse of the average optical line of the left side of the image. In the second example in Figure 3.28 the camera is moving forward and it is visible that the correction vectors are pointing to the vanishing point, where the correction vector closer to the center is slighly shorter than the one on the left.

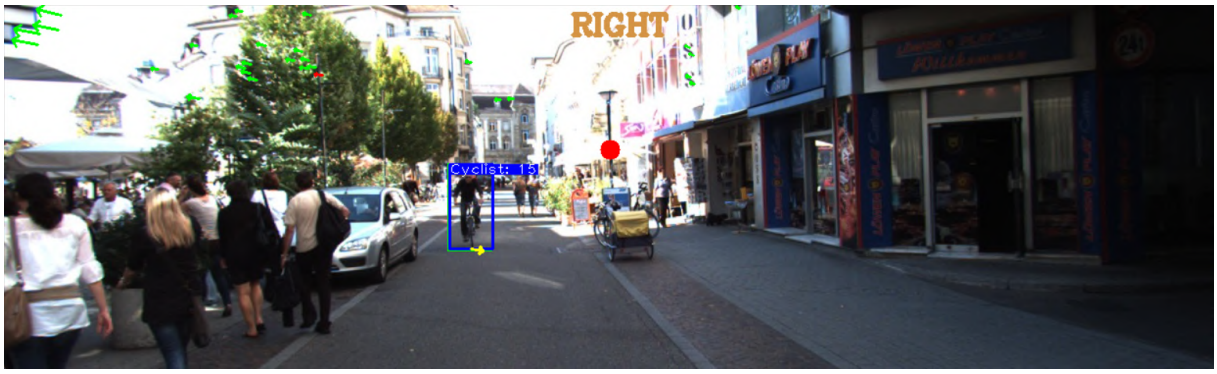


Figure 3.27. Correction vector example 1: Camera is moving to the right

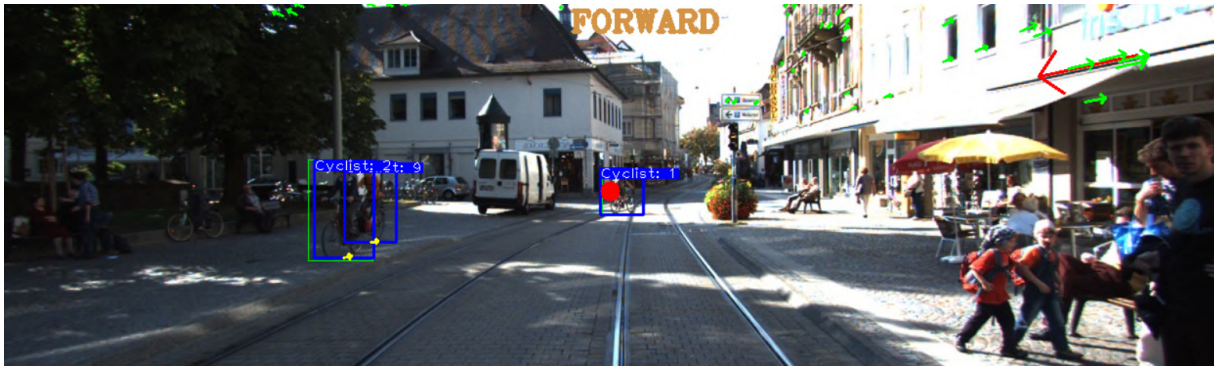


Figure 3.28. Correction vector example 2: Camera is moving forward

3.5.2. Trajectory Prediction

After obtaining the correction vectors the Trajectory Prediction module can now proceed to predict the future bounding boxes and the orientation angle of the cyclist. The algorithm is as follows (refer to Figure 3.21):

1. Calculate the polynomial coefficients (m, b) using the polyfit() function from the numpy package. This function takes the centers of past bounding boxes as input.
2. Predict the next bounding box centers by utilizing the obtained coefficients (m and b) in a simple line equation (y = mx + b). The x value equals the average distance between centers in the past frames multiplied by the desired number of steps forward (e.g., 10 steps).
3. Predict the bounding boxes as follows:
   a. Calculate the bounding boxes using the predicted bounding box centers and the previous widths and heights of the bounding boxes.
   b. Truncate the predicted bounding boxes if they overlap with the frame borders and discard any predictions that fall outside of the frame boundaries.
4. Estimate the orientation angle:
   a. Calculate the orientation angle using the past centers of bounding boxes and incorporate it with the correction vector obtained from the Camera Motion Estimator.
   b. Apply momentum to the orientation angle by blending it with the newly calculated angle using a weighted momentum, which helps to smooth out sudden changes in direction.

Figure 3.29 presents an illustrative example of the output generated by the Trajectory Prediction module. The blue bounding box is the output of the SORT algorithm. The blue circles behind the cyclist are the past bounding box centers. The red circles are the future bounding box centers. The green bounding boxes are the ground truth future bounding boxes (1, 3, 10 steps ahead). The red bounding boxes are predictions for the next steps. The yellow arrow in the middle is the estimated orientation angle. Blue arrow in the center of the cyclist is the ground truth orientation of the cyclist. And as previously mentioned, the red dot in the middle is the estimated vanishing point, and the text in top of the frame describes the camera movement.

In Figure 3.30 the truncation of the bounding boxes can be observed.

Another example of trajectory prediction can be viewed in Figure 3.31, in this case, as the cyclists are getting closer to the camera, their bounding boxes are getting bigger, which the Trajectory Prediction does not take into account. In this solution future bounding boxes remain the same size as the last bounding box predicted by SORT. There were trials of utilizing interpolated bounding box widths and heights, predicted in a similar way as the centers are currently predicted, but they unfortunately were unstable and were not considered for future experiments.

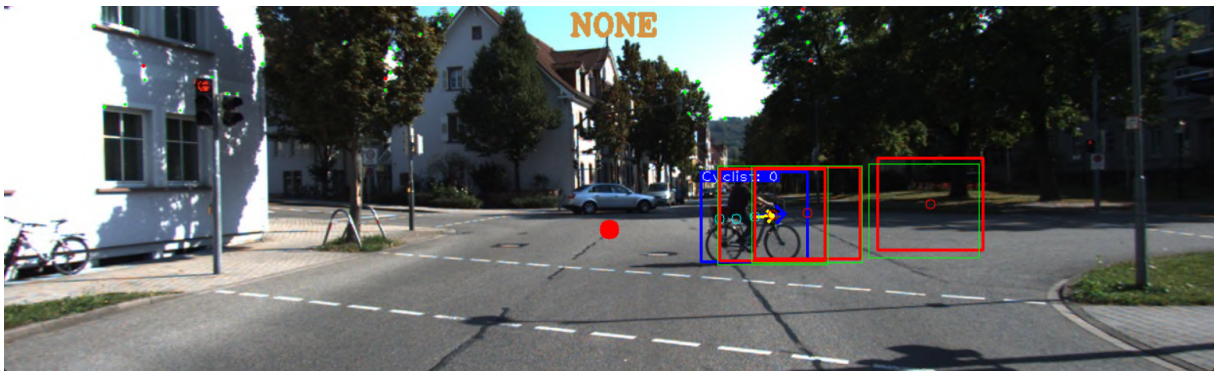


Figure 3.29. Trajectory prediction: example 1

Figure 3.30. Trajectory prediction: example 2, truncation of the predicted bounding boxes

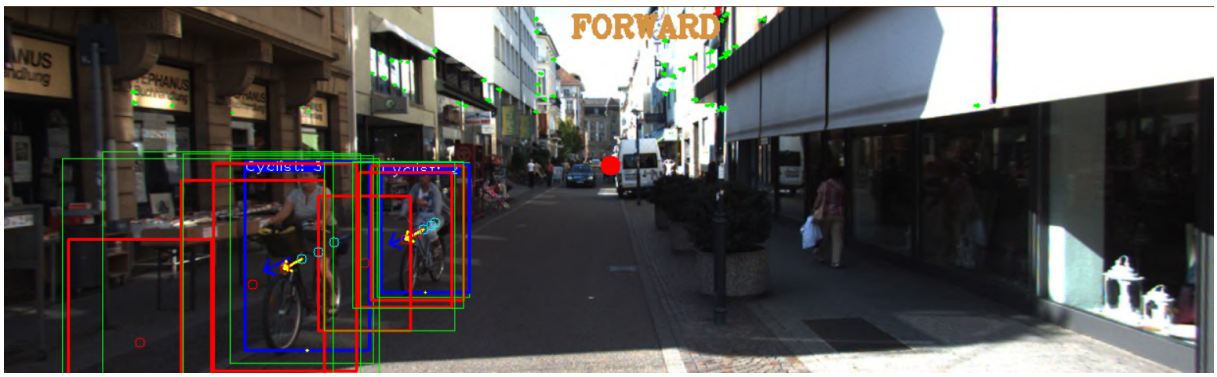


Figure 3.31. Trajectory prediction: example 3

3.6. Evaluation metrics

Each proposed model requires proper evaluation metrics so it can be easily compared to other solutions. As YOLO is using the COCO AP metrics to evaluate it's models, so it was decided to not change it. It was decided to use two default COCO metrics AP50 (AP with IOU threshold set at 0.5) and AP50:95 (averaged AP from IOU threshold = 0.5, to 0.95 with a step of 0.05), which both are calculated over 101 recall thresholds (from 0 to 1). An example evaluation result could follow as in Table 3.9 or as in the case of trajectory prediction in Table 3.10.

Scale	AP_{50}^{val}	$AP_{50:95}^{val}$	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$
0.1	**0.993**	**0.876**	0.59	0.427	0.726	0.63	0.675
0.5	0.992	0.858	0.646	0.44	0.647	0.668	0.657
0.9	0.979	0.815	**0.71**	**0.483**	**0.765**	**0.693**	**0.727**

Table 3.9. Example of metrics used to evaluate Object Detection and Object Tracking

Step	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$	Angle MAE	Angle RMSE
1	0.603	0.363	0.798	0.658	0.721	22.136	35.504
3	0.484	0.171	0.683	0.566	0.619	22.136	35.504
10	0.043	0.013	0.200	0.162	0.179	22.136	35.504
AVG	**0.376**	**0.182**	**0.56**	**0.462**	**0.507**	**22.136**	**35.504**

Table 3.10. Example of metrics used for evaluating Trajectory Prediction

— Evaluation metrics explanation:

- AP_{50}^{val} - this notation means that AP precision was calculated over validation dataset with IOU threshold of 50%.
- $AP_{50:95}^{val}$ - AP on validation set average over IOU thresholds ranging from 0.5 to 0.95.
- AP_{50}^{test} and $AP_{50:95}^{test}$ - same as in previous cases, only AP was calculated over test set.
- Precision - unlike precision used in COCO metrics, it is not averaged precision over all frames, it is simply TP over all frames divided by TP + FP, as described in the subsection 2.1.4.
- Recall - similarly as precision, TP / (TP + FN).
- F1 score - like AP it combines precision and recall, but in a way that does not get affected but differences in IOU's and it has proved quite useful in determining various values and thresholds in hyperparameter tuning.
- Angle Mean Absolute Error (in degrees) - This metric is used for evaluating Trajectory Prediction only. It calculates the absolute difference between the predicted orientation angle of a cyclist and the real cyclist orientation angle, which is derived from the bottom of the cyclist's 3D boxes (as the default orientation angle provided by KITTI seemed to be set up in the incorrect plane). The metric then averages

these absolute errors over all frames. Mind that the real orientation angle and the direction in which the predicted bounding boxes in the future frames are positioned is not the same case (for example, cyclist is in on the left side, orientated to the right, stationary and the camera is passing him by on the right. On the frame he will shift left, while his orientation is to the right).

- Angle Root Mean Square Error (in degrees) - similarly as angle MAE, angle RMSE metric evaluates the difference between predicted cyclist orientation angle and calculated angle derived from the bottom of 3D bounding box label. As MAE provides an easy to understand evaluation metric, RMSE gives a better idea about outlying angles. If the predicted arrow is diverting slightly from the ground truth, but overall points in the same direction it is not a problem. But if an arrow points perfectly in the same direction half of the times, This is important because if an arrow consistently points in the same direction half of the time, but points in the opposite direction by 180 degrees the other half, then it is an issue, An issue that RMSE should reveal and bring to daylight.

Library used for calculating these metrics (except the Mean absolute angle error) is a pip package called Mean Average Precision [43]. As it did not provide TP, FP, TN, FN counts the package had to be modified to enable calculation of precision, recall and f1 score used in this paper. To ensure that the changes in the package were correct, another popular Github repository was used for validating the results, called Object Detection Metrics Review [44], with a published paper[36].

3.7. Conclusion

This chapter presents the methodology employed in this research. Two cyclist datasets, namely the KITTI Object Tracking Dataset and the Tsinghua-Daimler Cyclist Detection Benchmark, were selected for the purpose of this study, and their contents were thoroughly analyzed.

The KITTI Object Tracking Dataset suffers from a scarcity of cyclist annotations and images. While filtering highly occluded cyclists from the training set was relatively straightforward due to the availability of a 4-level occlusion parameter, the truncation parameter only had two levels: 0 (non-truncated) and 1 (truncated). This binary classification failed to encompass the full range of possible truncation scenarios. To address this issue, a proposed solution was implemented to remove truncated cyclists from frames while preserving other cyclists within the same frame, thus preventing the entire frame from being discarded.

Conversely, the Tsinghua-Daimler Cyclist Dataset contained a substantial number of cyclist annotations, with 22,173 cyclist annotations and 14,674 images. However, the annotations in the training set had a 10% occlusion threshold, meaning that any cyclist occluded by 10% or more was not annotated but still appeared in the frame. This

contradicts the suggestion made by the author of YOLOv4 in the Darknet YOLOv4 GitHub repository [7], which emphasizes the importance of annotating all objects, including cyclists visible up to 90% in the frame. Unfortunately, the small validation and test sets, which are annotated up to 80% occlusion, were discovered too late in the process of this research to be fully utilized.

In the Cyclist Detection section, various YOLO editions were compared. Initially, YOLOv4 was used for object detection in this study. However, in the summer of 2022, YOLOv7 was released, offering improved average precision (AP) and speed compared to its predecessors. YOLOv5 and YOLOv8 models, published by Ultralytics [26] [25], were also considered as potential alternatives. However, due to the lack of sufficient research papers supporting their claimed improvements in accuracy and performance, YOLOv4 and YOLOv7 were chosen as the preferred models for this research, and YOLOv5 and YOLOv8 were excluded from further consideration.

In the Cyclist Tracking section, due to time contrictions only one object tracking algorithm was considered - SORT (Simple Online and Realtime Tracking), a state-of-the-art algorithm for 2D multiple object tracking. It was slightly modified in the sake of this paper with added bounding box history and last orientation angle to the trackers.

Cyclist Trajectory Prediction has two main objectives. First, to predict bounding boxes in the future frames (three steps were chosen: 1 frame ahead, 3 and 10). Second, to estimate the real orientation angle of each cyclist. To achieve the second objective information about the camera motion (as the camera in this dataset is not stationary, but is mounted in front of a moving car) was needed and it was provided by the Camera Motion Estimator module. This module uses the estimated vanishing point, calculates optical lines indicating the movement on the image, filters outliers and calculates the correction vectors based on the camera motion. Those correction vectors are then merged with the interpolated direction angle from the past bounding boxes. Then the estimated orientation angle is merged with the estimated angle from the last frame with a momentum to counter sudden changes in direction. Then the future bounding boxes and orientation angle are send to the Metrics module for evaluation.

The Metrics module employed a selection of seven primary metrics to evaluate cyclist trajectory prediction. Specifically, two COCO metrics, namely AP50 and AP50:95, were utilized, which are widely used for the evaluation of object detectors. Precision, recall, and F1 score were calculated based on the total counts of true positives, true negatives, false positives, and false negatives, considering an intersection over union (IOU) threshold of 0.5. In contrast, COCO calculates precisions and recalls independently for each frame, using the same values for generating precision-recall (PR) curves. These precisions and recalls are then averaged, leading to metrics such as AR1, AR10, and AR100. The inclusion of the F1 score was deemed valuable as it provides an alternative perspective to the default COCO metrics on the balance between precision and recall.

In the context of object detection, object tracking, and the prediction of future bounding boxes, the aforementioned five metrics were employed: AP50, AP50:95, Precision, Recall, and F1 score. Additionally, for evaluating the orientation angle predicted by the Trajectory Prediction module, the Mean Absolute Error of the estimated orientation angle in degrees was utilized as the final metric.

In the next chapter the results of this work will be presented, including training, hyperparameter optimization and testing the final solution.

4. Results

In this chapter all the modules mentioned in the methodology will be thoroughly tested and evaluated. That includes YOLO - object detection, Sort - object tracking and a custom Trajectory Prediction module.

4.1. Object Detection

This section will focus on summarising the training, testing and optimizing the hyper-parameters of two state-of-the-art object detectors: YOLOv4 and YOLOv7.

```
[net]
batch=64
subdivisions=4
# Training
#width=512
#height=512
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 0.7
exposure =0.4
hue=0.015

learning_rate=0.0013
burn_in=1000
policy=steps
scales=.1,.1

# for tsinghua
#max_batches = 11000
#steps=8800,9900

#for kitti
max_batches = 6000
steps=4800,5400


cutmix=0.15
mosaic=1
```

```
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.1  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.3  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 0.7  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.2  # image translation (+/- fraction)
scale: 0.9  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.15  # image mixup (probability)
copy_paste: 0.0  # image copy paste (probability)
paste_in: 0.15  # image copy paste (probability), use 0 for faster training
loss_ota: 0 # use ComputeLossOTA, use 0 for faster training
```

Figure 4.1. Fragment of YOLOv4 .cfg and YOLOv7 hyper hyperparameter .yaml

Most of the hyperparameters of YOLOv4 and YOLOv7 config files were left untouched (see Figure 4.1).

Setup:

- YOLOv4 input size: 416x416.
- YOLOv7 input size: 640x640.
- YOLOv7 max batches: 6000 In YOLOv4 number of max batches corresponds to number of classes * 2000, but no less than number of training images and no less than 6000 [6].

- YOLOv7 training epochs num: 100.
- Train Valid Split: 80/20%.
- Test recordings: 0012 and 0019 - 349 cyclist annotations, 235 images with cyclists and 1137 images in total.
- YOLOv7 loss_ota: 0 - an important part as when this hyperparameter was set to 1, the model exhibited major issues with training.
- YOLOv7 weights - YOLOv7x with 71.3M parameters Table 3.8.
- YOLOv4 weights - YOLOv4 with 64.4M parameters.
- Weights used for testing were chosen by the YOLO models (best.pt for YOLOv7 and yolov4-obj_best.weights for YOLOv4).
- YOLOv4 and YOLOv7 were trained in Google Colab on Tesla A100 GPU with 40GB of VRAM. Training YOLOv4 implemented in Darknet took approximately 2 hours, training YOLOv7 (yolov7x weights) took 45 minutes on average, on the balanced training set (2232 images total).
- Weights and Biases [5] were used for gathering experiment results from training.
- OpenCV compiled with CUDA support - necessary for fast image loading and processing speed. Also necessary for running the darknet YOLOv4 model on GPU in Python, OpenCV. Without it YOLOv4 runs on CPU.
- Test platform specs: CPU - Ryzen 3600, GPU - RTX 3070 8GB, RAM - 16GB 3200 MHz.

For more information see section 3.2 and section 3.3.

Parameters tested in the Object detection module:

- Image scale (Data augmentation) - a data augmentation technique used in YOLO. It changes the training image sizes whilst preserving the aspect ratio.
- Mosaic (Data augmentation) - a data augmentation technique that merges four random training images into one training sample. A widely used technique increasing the training speed and the accuracy overall in YOLO models.
- Confidence threshold - each YOLO detection has a confidence score from 0 to 1, the higher it is, the more probable it is that in this place, this object exists. Confidence threshold simply categorizes whether that detection is correct or not. The higher it is, the less objects are detected, but the more precise the detections are.
- Non-max suppression threshold - a threshold used for collapsing overlapping bounding boxes. As YOLO is outputting multiple bounding boxes for each object and non-max suppression.

4.1.1. Training results and data augmentation in YOLOv7 test

In this section the training results of YOLOv7 will be presented along with a test of few data augmentation techniques built-in the model.

Setup:

- Training confidence threshold = 0.001
- Test confidence threshold = 0.1
- Non-max suppression threshold = 0.6

4.1.1.1. Image Scale test

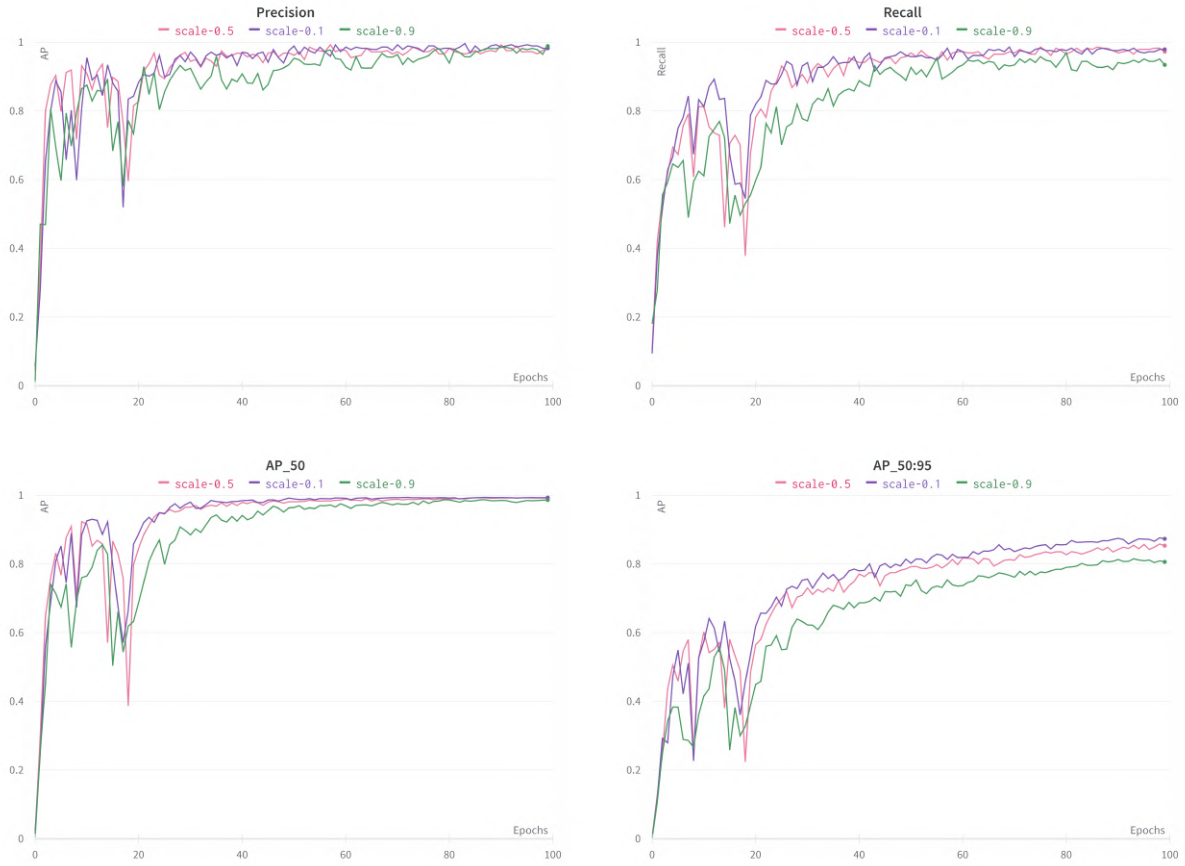


Figure 4.2. Training with various image scales results

Scale	AP_{50}^{val}	$AP_{50:95}^{val}$	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$
0.1	**0.993**	**0.876**	0.59	0.427	0.726	0.63	0.675
0.5	0.992	0.858	0.646	0.44	0.647	0.668	0.657
0.9	0.979	0.815	**0.71**	**0.483**	**0.765**	**0.693**	**0.727**

Table 4.1. Image scale augmentation validation and test results

Summary: In Figure 4.2 are presented the results of testing the image scaling data augmentation technique on YOLOv7 model. The charts show clearly that the higher the image scaling is, the slower the learning is. Though all of the charts used the validation data for evaluation, one may wonder if there is a possibility of overfitting. Usually overfitting occurs when the validation accuracy is decreasing whilst the training accuracy increases. But in this case the validation set is contaminated with frames from the same recordings

as the training set. That results in same cyclists appearing in both sets, only in slightly different positions and angles. In Table 4.1 apart from the validation AP also metrics evaluated on test dataset are presented, showing that indeed the overfitting, even in the validation set is present. To no surprise the more variability in the training data that image scaling augmentation offers the less overfitting there is.

4.1.1.2. Mosaic test

Figure 4.3. Training with mosaic augmentation off and on results

Scale	AP_{50}^{val}	$AP_{50:95}^{val}$	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$
0	0.86	0.581	0.389	0.238	0.457	0.433	0.445
1.0	0.979	0.815	0.71	0.483	0.765	0.693	0.727

Table 4.2. Mosaic augmentation validation and test results

Summary: In Figure 4.3 are presented the results of training with mosaic augmentation off and on (1 - on, 0 - off). Clearly mosaic augmentation not only immensely speeds up the training process (4 images in 1) but also without mosaic YOLO settles at lower validation AP levels. Evaluation on test set also reveals that test AP, Precision, Recall and F1 score is also lower without mosaic augmentation (see Table 4.2).

4.1.2. YOLO v4

In this section confidence threshold and non max suppression will be tested with YOLOv4.

4.1.2.1. Confidence threshold test

In this test, YOLOv4 will be evaluated on various confidence thresholds. Setup:

- Image input size: 416x416
- Non max suppression threshold: 0.5

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$
0.001	0.589	0.357	0.588	0.668	0.626
0.1	0.589	0.357	0.588	0.668	0.626
0.2	**0.589**	0.357	0.588	**0.668**	0.626
0.3	0.583	0.355	0.625	0.659	0.642
0.4	0.577	0.351	0.659	0.648	0.653
0.5	0.570	0.347	0.692	0.636	0.663
0.6	0.570	0.346	0.717	0.630	**0.671**
0.7	0.541	0.334	0.739	0.599	0.661
0.8	0.526	0.326	0.758	0.573	0.653
0.9	0.473	0.300	**0.785**	0.501	0.612

Table 4.3. Results of the Confidence Threshold Experiment for YOLOv4

Summary:

The results in Table 4.3 show that as the confidence threshold increases, both mAP50 and mAP50:95 values gradually decrease. Precision increases as the threshold increases, indicating fewer false positives. However, recall decreases, indicating a decrease in the number of true positives identified. This suggests that raising the confidence threshold leads to more accurate but fewer detections. The best F1 score of 0.671 was calculated for threshold 0.6, a decent middle ground maximizing both precision and recall.

4.1.2.2. Non Maximum Suppression threshold test

In this test, YOLOv4 will be evaluated on various mon maximum suppression IOU thresholds. Setup:

- Image input size: 416x416
- Confidence threshold=0.4

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	F1 $score_{50}^{test}$
0.1	0.562	0.342	0.715	0.625	0.667
0.2	0.563	0.343	0.716	0.628	0.669
0.3	0.570	0.346	0.717	0.630	0.671
0.4	0.570	0.346	0.717	0.630	0.671
0.5	**0.570**	**0.346**	**0.717**	0.630	**0.671**
0.6	0.570	0.347	0.714	0.630	0.670
0.7	0.568	0.346	0.701	0.630	0.664
0.8	0.530	0.331	0.521	0.630	0.571
0.9	0.403	0.267	0.279	**0.636**	0.387

Table 4.4. Results of the Confidence Threshold Experiment for YOLOv4

Summary:

The results in Table 4.4 illustrate the performance of the model at different NMS thresholds. As the NMS threshold increases, both mAP50 and mAP50:95 values remain relatively stable, suggesting that the threshold has a minimal impact on these metrics. Precision gradually decreases as the NMS threshold increases, indicating more false positives being included in the detections. On the other hand, recall shows a slight increase with higher NMS thresholds, implying that more true positives are detected but at the cost of potential duplicate or redundant detections. The NMS threshold with highest mAP is 0.5 and it will be used in the next experiments.

4.1.3. YOLO v7

In this section confidence threshold and non max suppression will be tested with YOLOv7 model.

4.1.3.1. Confidence threshold test

In this test, YOLOv7 will be evaluated on various confidence thresholds.

Setup:

- Non max suppression threshold: 0.5

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$
0.001	**0.736**	**0.491**	0.036	**0.937**	0.069
0.1	0.720	0.484	0.301	0.837	0.443
0.2	0.717	0.481	0.412	0.822	0.549
0.3	0.708	0.477	0.487	0.802	0.606
0.4	0.698	0.472	0.561	0.782	0.653
0.5	0.692	0.469	0.629	0.771	0.692
0.6	0.666	0.456	0.691	0.731	0.710
0.7	0.637	0.441	0.748	0.699	**0.723**
0.8	0.598	0.418	0.810	0.648	0.720
0.9	0.505	0.358	**0.873**	0.530	0.660

Table 4.5. Results of the Confidence Threshold Experiment for YOLOv7

Summary:

Result in Table 4.5 show that adjusting the confidence threshold in the YOLOv7 [47] model, similarly as in YOLOv4, allows for a trade-off between precision and recall. A higher threshold improves precision but decreases recall, while a lower threshold increases recall but may lead to more false positives. F1 score metric gives a more balanced overview of model performance, it combines precision and recall but at the same time it is not favouring recall as AP does. Highest F1 score was obtained when the confidence threshold=0.7.

4.1.3.2. Non-max suppression threshold test

In this section Non-max suppression IOU threshold will be tested on two confidence thresholds: 0.3, 0.7.

Setup:

- Confidence threshold: 0.3

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$
0.1	0.686	0.465	0.484	0.777	0.596
0.2	0.693	0.470	0.487	0.788	0.602
0.3	0.703	0.476	0.489	0.799	0.607
0.4	**0.708**	**0.477**	**0.489**	**0.802**	**0.607**
0.5	0.708	0.477	0.487	0.802	0.606
0.6	0.708	0.477	0.484	0.802	0.604
0.7	0.707	0.477	0.481	0.802	0.602
0.8	0.706	0.476	0.467	0.802	0.590
0.9	0.695	0.472	0.388	0.802	0.523

Table 4.6. Confidence Threshold = 0.3 test results for YOLOv7

Setup:

- Confidence threshold: 0.7

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	F1 $score_{50}^{test}$
0.1	0.628	0.435	0.748	0.688	0.716
0.2	0.636	0.439	0.747	0.693	0.719
0.3	0.637	0.441	0.748	0.699	0.723
0.4	0.637	0.441	0.748	0.699	0.723
0.5	**0.637**	**0.441**	**0.748**	**0.699**	**0.723**
0.6	0.637	0.441	0.746	0.699	0.722
0.7	0.637	0.441	0.746	0.699	0.722
0.8	0.637	0.441	0.742	0.699	0.720
0.9	0.631	0.438	0.707	0.699	0.703

Table 4.7. Confidence Threshold = 0.7 test results for YOLOv7

Summary:

Comparing the results in Table 4.6 and Table 4.7, there is no significant difference whether the confidence threshold is high or low. In both cases when non max suppression IOU threshold is set in the range 0.4-0.6 the results are fairly the same.

4.1.4. Summary

Two models were tested on the KITTI dataset, YOLOv4 and YOLOv7. The input image sizes were as follows: YOLOv4 - 416x416 and YOLOv7 - 640x640, which gives a significant advantage to the latter. For a proper test between those two, tests should performed on the same input sizes, but due to the longer training durations in YOLOv4 and overall lack of time it was not possible. For more information on YOLOv4 and YOLOv7 performance refer to section 3.3.

4.2. Object Tracking

In this section, cyclist tracking with Sort (object tracking algorithm) will be evaluated.

Setup:

- Object Detection model: YOLOv7
- Object Tracking algorithm: SORT
- YOLOv7 confidence threshold: 0.7
- YOLOv7 non-max suppression threshold: 0.5

Parameters tested in Object Tracking module:

- Max age - maximum number of frames without detection after which a cyclist tracker is deleted
- Min hits - minimum, consecutive number of frames in which a cyclist object must appear in order to be tracked
- Sort IOU threshold - IOU threshold used by Sort to associate YOLO detections to tracked cyclists

4.2.1. Max age test

This test shows how the maximum number of consecutive frames without detections of the tracked cyclist is changing the results (see Table 4.8).

Setup:

- Min hits: 3
- SORT IOU threshold: 0.3

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	F1 $score_{50}^{test}$
1	0.552	0.322	0.849	0.610	0.710
2	0.551	0.316	0.846	0.613	0.711
3	0.547	0.314	0.843	0.613	0.710
4	0.547	0.314	0.843	0.613	0.710
5	0.547	0.310	0.843	0.613	0.710
6	0.547	0.310	0.843	0.613	0.710
7	0.547	0.310	0.843	0.613	0.710
8	0.547	0.310	0.843	0.613	0.710
9	0.547	0.310	0.843	0.613	0.710
10	0.547	0.310	0.843	0.613	0.710

Table 4.8. Max age test results

Summary:

Due to the fact that by default Sort returns tracked objects only if they were detected this frame or in the last frame, there is no difference in the results if max-age is higher than 2 (see Table 4.8).

4.2.2. Min hits test

This test shows how the minimum amount of consecutive cyclist detections changes the results (see Table 4.9).

Setup:

- Max age: 5
- Sort IOU threshold: 0.3

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	F1 $score_{50}^{test}$
1	0.576	0.304	0.761	**0.713**	**0.737**
2	**0.578**	**0.311**	0.802	0.673	0.732
3	0.557	0.308	0.825	0.633	0.716
4	0.547	0.304	0.840	0.616	0.711
5	0.534	0.301	0.850	0.599	0.703
6	0.530	0.296	0.860	0.582	0.694
7	0.523	0.294	0.875	0.562	0.684
8	0.508	0.293	0.888	0.547	0.677
9	0.500	0.285	0.894	0.533	0.668
10	0.482	0.279	**0.900**	0.519	0.658

Table 4.9. Min hits test results

Summary:

Table 4.9 shows that min hits hyperparameter is positively correlated with precision and negatively correlated with recall. For increasing precision the reason is that the more detections is necessary to start tracking an object, the more confident Sort is that this object really is a Cyclist (avoiding FP), but at the same time the more detections that could be a Cyclist (FN) is missed.

4.2.3. Sort IOU Threshold test

This test evaluates the change in Sort accuracy with different IOU thresholds (see Table 4.10).

Setup:

- Max age: 5
- Min hits: 2

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	F1 $score_{50}^{test}$
0.1	0.574	0.306	0.799	0.673	0.731
0.2	0.578	0.311	0.802	0.673	0.732
0.3	0.578	0.311	0.802	0.673	0.732
0.4	0.578	0.311	0.802	0.673	0.732
0.5	**0.579**	0.311	**0.805**	**0.673**	**0.733**
0.6	0.559	**0.322**	0.800	0.653	0.719
0.7	0.469	0.289	0.778	0.573	0.660
0.8	0.210	0.131	0.685	0.292	0.410
0.9	0.079	0.055	0.691	0.109	0.188

Table 4.10. Sort IOU Threshold test results

Summary:

Table 4.10 shows the best results for AP50, precision, recall and f1 score appear for IOU threshold=0.5, but interestingly best results for $AP_{50:95}^{test}$ appear when the threshold is higher at 0.6. Sort IOU threshold=0.5 will be used in the next tests.

4.3. Trajectory Prediction

In this final section, cyclist trajectory prediction will be evaluated. Both, the ability to predict future bounding boxes of cyclists on the frame and their real orientation angles. In the hyperparameter tests the metrics where averaged over all three prediction steps: 1, 3 and 10 steps. For more information refer to subsection 3.5.2.

Setup:

- Object Detection model = YOLOv7 (weights: yolov7x)
- Object Tracking algorithm = SORT
- YOLOv7 confidence threshold = 0.7
- YOLOv7 non-max suppression threshold = 0.4
- Max age = 5
- Min hits = 2
- Sort IOU threshold = 0.5
- Prediction steps = [1, 3, 10]

Parameters tested in the Trajectory Prediction module:

- max-num-of-past-bbs-for-direction - Max number of past bounding boxes used for estimating the direction for predicting future centers of bounding boxes.
- max-num-of-past-bbs-for-avg-distance - Max number of past bounding boxes used for estimating the average distance covered by a cyclist in a single frame (for predicting future centers of bounding boxes)
- angle-momentum - the weight of that last estimated orientation angle. The bigger the momentum value, the smoother and slower the changes of the orientation angle.
- correction-vector-weight - The weight of camera motion correction vectors applied to the orientation angle estimated from predicted future bounding box centers. The higher the weight, the bigger the impact of the correction vectors.

4.3.1. Max number of past bounding boxes for calculating average distance test

In this test, max-num-of-past-bbs-for-avg-distance hyperparameter will be tested (see Table 4.11).

Setup:

- max-num-of-past-bbs-for-direction = 3
- angle-momentum = 0.5
- correction-vector-weight = 1

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$	Angle MAE	Angle RMSE
2	**0.442**	**0.248**	**0.609**	**0.523**	**0.562**	**40.410**	**63.486**
3	0.442	0.247	0.607	0.522	0.561	41.465	64.977
4	0.441	0.245	0.609	0.523	0.562	42.426	66.381
5	0.434	0.242	0.604	0.519	0.557	44.307	68.614
6	0.430	0.240	0.602	0.517	0.556	43.484	66.534
7	0.425	0.237	0.599	0.515	0.554	46.387	69.524
8	0.425	0.235	0.598	0.514	0.553	47.217	70.203
9	0.419	0.233	0.593	0.510	0.548	46.849	69.756
10	0.417	0.231	0.591	0.509	0.546	47.859	70.818

Table 4.11. Trajectory Prediction Results

4.3.2. Max number of past bounding boxes for calculating direction test

In this test, max-num-of-past-bbs-for-direction will be tested (see Table 4.12).

Setup:

- max-num-of-past-bbs-for-avg-distance = 2
- angle-momentum = 0.5
- correction-vector-weight = 1

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$	Angle MAE	Angle RMSE
2	**0.446**	0.247	0.608	0.522	0.561	45.069	68.860
3	0.442	**0.248**	**0.609**	**0.523**	**0.562**	40.567	63.505
4	0.442	0.248	0.607	0.521	0.561	39.958	63.558
5	0.442	0.247	0.607	0.521	0.561	**38.200**	**61.356**
6	0.442	0.246	0.606	0.521	0.560	38.828	61.661
7	0.442	0.245	0.606	0.520	0.559	41.035	63.815
8	0.442	0.245	0.606	0.520	0.559	40.636	62.970
9	0.441	0.246	0.605	0.519	0.558	39.542	61.556
10	0.439	0.245	0.601	0.516	0.555	39.743	61.693

Table 4.12. Trajectory Prediction Results

4.3.3. Correction vectors weight test

In this test, the impact of the camera motion correction vector weight hyperparameter on the trajectory prediction will be evaluated.

Setup:

- max-num-of-past-bbs-for-avg-distance = 2
- max-num-of-past-bbs-for-direction = 3
- angle-momentum = 0.5

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$	Angle MAE	Angle RMSE
0.0	0.442	0.247	0.607	0.521	0.561	29.431	**48.954**
0.1	0.442	0.247	0.607	0.521	0.561	29.300	49.952
0.2	0.442	0.247	0.607	0.521	0.561	**28.804**	49.807
0.3	0.442	0.247	0.607	0.521	0.561	29.124	50.578
0.4	0.442	0.247	0.607	0.521	0.561	29.560	51.518
0.5	0.442	0.247	0.607	0.521	0.561	30.667	53.430
0.6	0.442	0.247	0.607	0.521	0.561	31.769	54.839
0.7	0.442	0.247	0.607	0.521	0.561	32.619	55.729
0.8	0.442	0.247	0.607	0.521	0.561	34.185	57.339
0.9	0.442	0.247	0.607	0.521	0.561	36.069	59.136
1.0	0.442	0.247	0.607	0.521	0.561	38.183	61.067

Table 4.13. Correction vectors weight test results (tested on test recordings - 0012, 0019)

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$	Angle MAE	Angle RMSE
0.0	0.777	0.587	0.883	0.809	0.844	65.166	88.678
0.1	0.777	0.587	0.883	0.809	0.844	65.038	88.585
0.2	0.777	0.587	0.883	0.809	0.844	64.895	88.458
0.3	0.777	0.587	0.883	0.809	0.844	64.758	88.277
0.4	0.777	0.587	0.883	0.809	0.844	64.584	88.080
0.5	0.777	0.587	0.883	0.809	0.844	**64.169**	**87.819**
0.6	0.777	0.587	0.883	0.809	0.844	64.506	88.158
0.7	0.777	0.587	0.883	0.809	0.844	64.415	88.136
0.8	0.777	0.587	0.883	0.809	0.844	64.232	88.038
0.9	0.777	0.587	0.883	0.809	0.844	64.088	87.963
1.0	0.777	0.587	0.883	0.809	0.844	**64.021**	**87.938**

Table 4.14. Correction vectors weight test results (tested on training recording - 0015)

Summary: Table 4.13 shows that camera motion correction vectors bring a slight improvement to the Angle MAE when the weight is set at 0.2. RMSE on the other hand does not experience local minimum at 0.2 but is is steadily increasing as the weight increases from 0 to 1, which might suggest that the correction vectors produce a higher variance in estimated angles. Increasing the accuracy of angle estimations in some cases and decreasing it in others. That might indicate some hidden issues with the implementation. Or that, test set consisting of two short recordings is not big enough to properly test the impact of correction vectors.

For that reason a second test was performed, but in this case a recording 0015 from the training set was used (see Table 4.14). Naturally the AP, precision, recall and f1 are higher as YOLOv7x model was trained on it. The angle MAE and RMSE provide interesting results. Firstly the MAE and RMSE are approximately two times greater than in the 1st test, which partially may be a result of the stationary cyclist that appears on the frame for a significant amount of time (see Figure 4.5). This time the best MAE and RMSE is reached when the weight is set at either 0.5 or 1, two local minima for one hyperparameter is a quite unusual result. That only indicates that the test set is not big enough. Where

in recordings 0012, 0019 the best weight was 0.2 then in this case it is either 0.5 or 1, a major difference inconsistency. It is also observable that in the second test, the variance across different weights is significantly smaller than in the test set. See Figure 4.4, where the cyclist on the left part of the frame is riding to the right, but the bounding boxes are to his left. Without the correction vectors the estimated orientation angle would also point to the left. Another example in Figure 4.6.

Overall incorporating the Camera Motion Estimator correction vectors into the solution (at proper thresholds) decreases angle MAE up to 2% but increases RMSE on the test set. However on the train set (recording 0015), there is reduce of both MAE and RMSE at all weight thresholds, up to 2% MAE and 1% RMSE improvement.

4.3.4. Angle momentum test

In this test, the impact of the angle momentum hyperparameter on the trajectory prediction will be investigated (see subsection 4.3.4).

Setup:

- max-num-of-past-bbs-for-avg-distance = 2
- max-num-of-past-bbs-for-direction = 3
- correction-vector-weight = 0.2

Threshold	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	F1 $score_{50}^{test}$	Angle MAE	Angle RMSE
0.0	**0.442**	**0.247**	**0.607**	0.521	**0.561**	**22.136**	**35.504**
0.1	0.442	0.247	0.607	0.521	0.561	22.665	36.973
0.2	0.442	0.247	0.607	0.521	0.561	23.791	39.330
0.3	0.442	0.247	0.607	0.521	0.561	25.330	42.632
0.4	0.442	0.247	0.607	0.521	0.561	27.186	46.646
0.5	0.442	0.247	0.607	0.521	0.561	28.770	49.744
0.6	0.442	0.247	0.607	0.521	0.561	30.550	52.980
0.7	0.442	0.247	0.607	0.521	0.561	32.237	55.533
0.8	0.442	0.247	0.607	0.521	0.561	34.556	58.816
0.9	0.442	0.247	0.607	0.521	0.561	43.470	66.821
1.0	0.442	0.247	0.607	0.521	0.561	57.133	80.084

Table 4.15. Trajectory Prediction Results

Summary: Surprisingly using angle momentum for smoothing the estimated orientation angle only worsens the results, both angle MAE and RMSE errors increasing as the momentum increases. It could be the case that once again the test set is not big enough and lacks all the edge cases in which smoothing the angle would prove beneficial. Either way angle momentum = 0 was chosen for further tests.

4.3.5. Trajectory Prediction final results

This section shows final results of the trajectory prediction at Table 4.16 for test set and for train set Table 4.17, where future bounding boxes were predicted 1, 3 and 10 steps ahead.

Setup:

- conf-threshold = 0.7
- nms-threshold = 0.4
- max-age = 5
- min-hits = 2
- sort-iou-threshold = 0.5
- min-hits = 2
- max-num-of-past-bbs-for-avg-distance = 2
- max-num-of-past-bbs-for-direction = 3
- correction-vector-weight = 0.2
- angle-momentum = 0.0

Step	AP_{50}^{test}	$AP_{50:95}^{test}$	$Precision_{50}^{test}$	$Recall_{50}^{test}$	$F1\ score_{50}^{test}$	Angle MAE	Angle RMSE
1	0.603	0.363	0.798	0.658	0.721	22.136	35.504
3	0.484	0.171	0.683	0.566	0.619	22.136	35.504
10	0.043	0.013	0.200	0.162	0.179	22.136	35.504
AVG	**0.376**	**0.182**	**0.56**	**0.462**	**0.507**	**22.136**	**35.504**

Table 4.16. Trajectory Prediction Final Results on test set (recordings 0012 and 0019)

Step	AP_{50}^{train}	$AP_{50:95}^{train}$	$Precision_{50}^{train}$	$Recall_{50}^{train}$	$F1\ score_{50}^{train}$	Angle MAE	Angle RMSE
0	0.900	0.780	0.979	0.906	0.941	42.105	68.047
1	0.855	0.679	0.978	0.868	0.919	42.105	68.047
3	0.808	0.492	0.948	0.839	0.890	42.105	68.047
10	0.316	0.149	0.565	0.465	0.510	42.105	68.047
AVG	**0.720**	**0.525**	**0.867**	**0.769**	**0.815**	**42.105**	**68.047**

Table 4.17. Trajectory Prediction final results on the train set (recordings 0015 and 0016)

YOLOv7x	Sort	Motion Estimation	Trajectory Prediction	Total Time	FPS
35.21ms	**0.25ms**	**19.75ms**	**0.05ms**	**55.26ms**	**18.1**

Table 4.18. Performance results: inference times for processing one frame. Tested on Ryzen 3600 and RTX 3070 8GB.

Summary: One of the reasons for low scores on the step = 10 in Table 4.16, apart from the obvious that 10 frames is quite a lot, (especially when there are slight jumps between the frames) is that as an object is moving towards the camera, it's position shift on the frame is not linear. When the cyclist is close to center of the frame (see Figure 4.9) there is minimal change from the camera's perspective. Only when the cyclist is moving away from the center, getting closer to the camera, his speed on the frame is increasing, which this Trajectory Prediction model does not account for and it shows in the test results.

Although the test recording 0012 has a cyclist that is easy to predict as he is moving from left to right (see Figure 4.7, Figure 4.11), recording 0019 is filled with cyclists going towards the camera on a narrow path (see Figure 4.8, Figure 4.10), making the trajectory prediction with this method severely more difficult as the predicted bounding boxes centers (red circles) are basically on top of each other (due to the fact that past bounding boxes were on top of each other too). More examples: Figure 4.12, Figure 4.13.

In Table 4.18 are the presented the performance results of the entire trajectory prediction pipeline, achieved on Ryzen 3600 and RTX 3070 8GB. Most of the inference time is caused by the object detector YOLOv7x and the Camera Motion Estimator. The results are still realtime, which was one of the goals of the paper. As the Camera Motion Estimator is not bringing a major improvement in estimating the cyclist orientation angle there is room for other methods to replace it.

Figure 4.4. Effect of the Camera estimation correction vector on the cyclist on the left part of the frame. The yellow vector at the bottom indicates the correction vector pointing correctly to the right (enlarged three times for better visibility), while the estimated future bounding boxes are positioned to the left. Yellow arrow in the middle of the cyclist is the predicted orientation angle and thanks to the correction vector it is pointing to the right as well (recording 0015). For more information on the various arrows and boxes refer to subsection 3.5.2

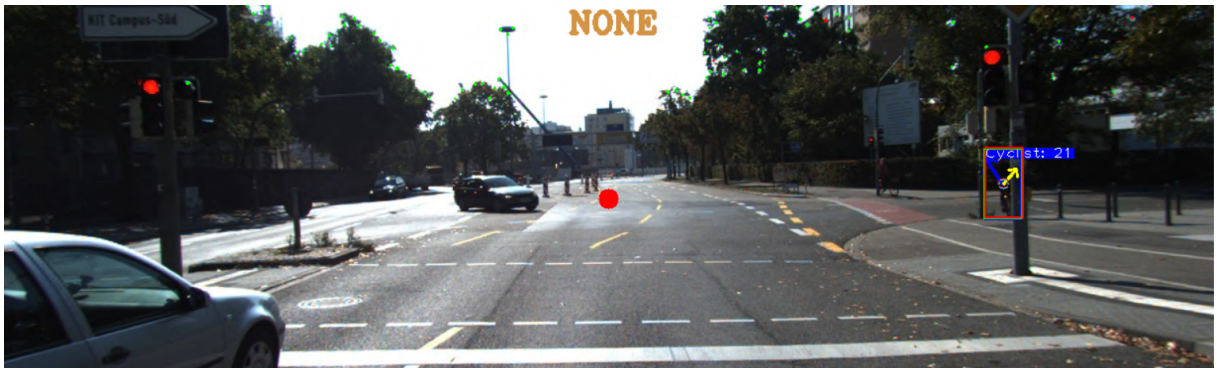


Figure 4.5. Stationary cyclist causing high Angle MSE and RMSE in recording 0015

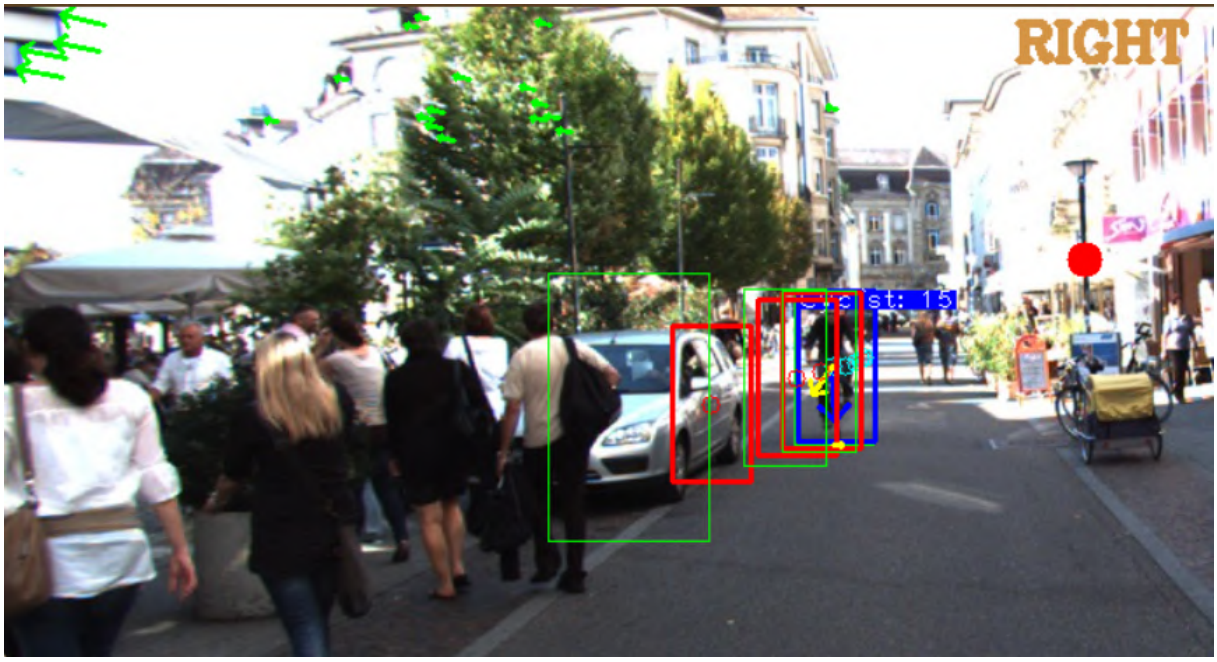


Figure 4.6. Another example for correction vectors correcting the estimated orientation angle.

Figure 4.7. Trajectory prediction example 1, cyclist moving to the right, camera stationary: test recording 0012

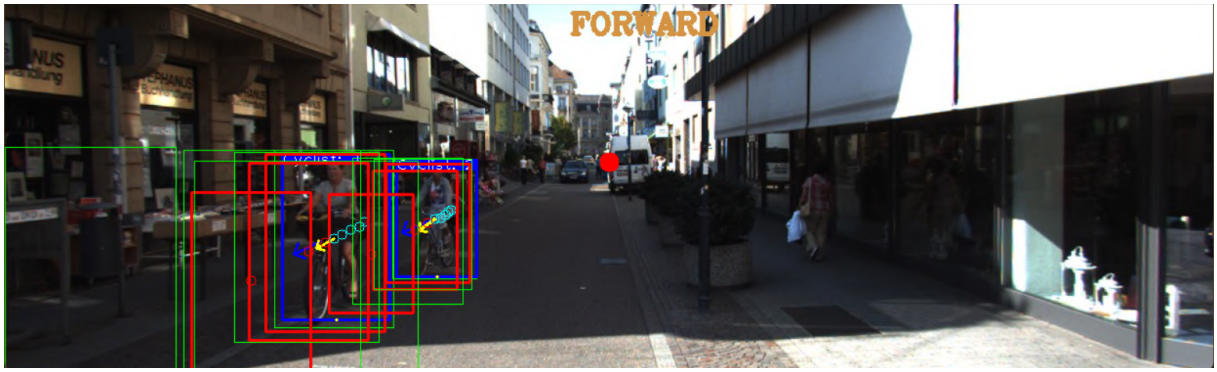


Figure 4.8. Trajectory prediction example 2, cyclist moving toward the camera from the left, camera is also moving forward: test recording 0019

Figure 4.9. Trajectory prediction example 3, cyclist moving toward the camera from the left, camera moving forward: test recording 0019

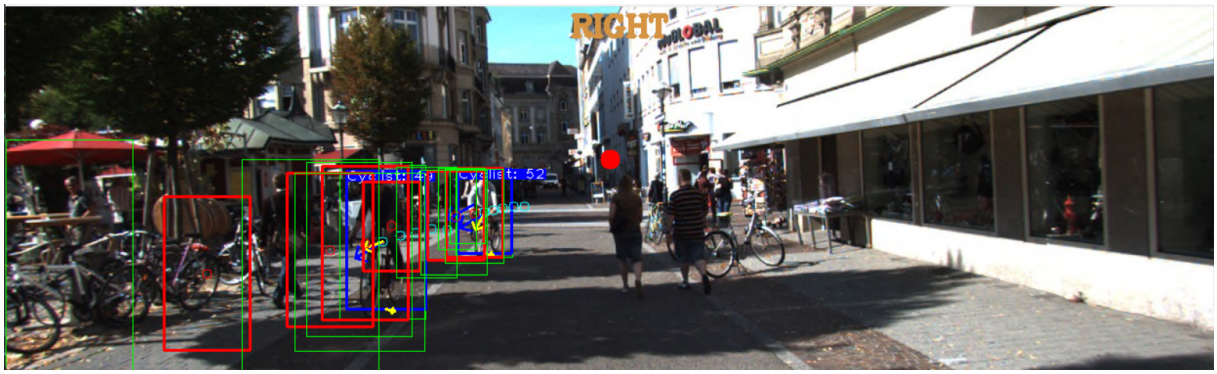


Figure 4.10. Trajectory prediction example 4, three cyclists: two visible and one occluded approaching the camera from left, camera turning to the right: test recording 0019

Figure 4.11. Trajectory prediction example 5: train recording 0015. Bounding boxes removed for the sake of visibility, red circles indicate future positions. Blue circles represent the past positions. Yellow and blue arrows, representing the predicted and ground truth cyclist direction

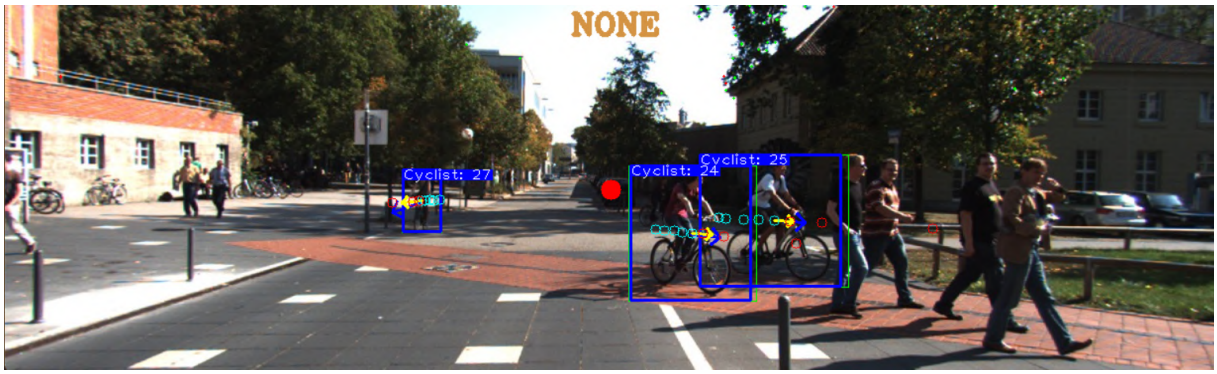


Figure 4.12. Trajectory prediction example 6, without bounding boxes, three cyclists, one on the left moving towards the camera, two in the middle moving to the right, stationary camera: train recording 0016

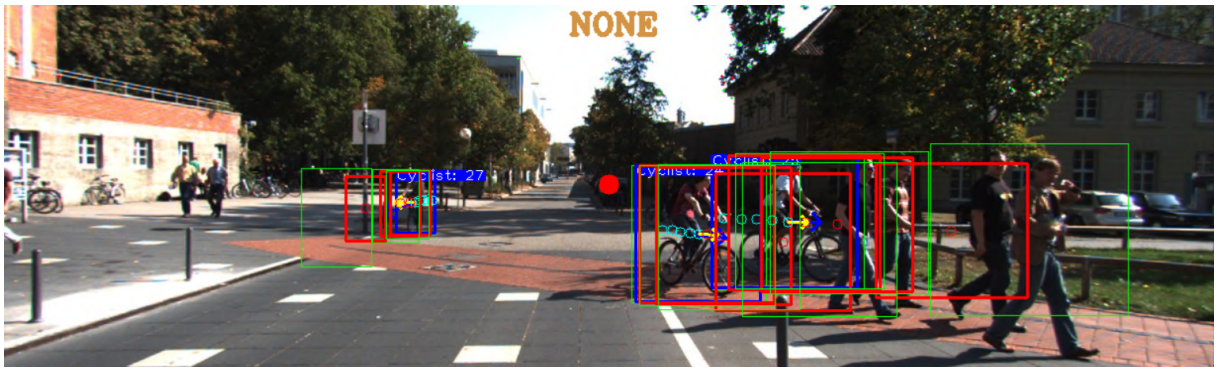


Figure 4.13. Trajectory prediction example 7, same case as in Figure 4.12 but with bounding boxes: train recording 0016

4.4. Conclusion

In this chapter all of the modules used for cyclist trajectory prediction were thoroughly evaluated and tested.

In the Object Detection section the training process of YOLOv7 (YOLOv7x weights) process was explained and illustrated. Two data augmentation techniques were tested: scale and mosaic, showing that both are crucial in reducing overfitting, especially when there is not enough data to train on. Then two hyperparameters were tested on YOLOv4 and YOLOv7x: confidence threshold and non max suppression IOU threshold. Showing how higher confidence threshold increases precision and decreases recall. In those tests and the ones further on, F1 score proved to be a great alternative to the default AP, giving a different perspective on precision and recall. Though YOLOv7 to no surprise produced better results than YOLOv4 one has to keep in mind that the comparison was not fair from the beginning, as YOLOv4 had image input size of 416x416, whilst YOLOv7 had 640x640. Also a bigger version of YOLOv7 was used - YOLOv7x. Without these advantages, the results would be closer, though still one would expect the YOLOv7 to prevail.

In the Object Tracking section, a well known state-of-the-art object tracking algorithm was employed - Sort (Simple, online, and realtime tracking), an extension of the Kalman Filter algorithm. To evaluate cyclist tracking it was decided not to use MOTA metrics [28] as tracking specific cyclist is not ultimate goal of this paper. The goal is to predict their trajectories, both in the form of future bounding boxes on the frame and in the form of an estimation of the real orientation angle, the real direction in which the cyclist is heading. In order to achieve better results, three hyperparameters were tested: max age, min hits and Sort IOU threshold. The model achieved best results when the minimum number of hits needed for a cyclist yolo detection to start being tracked was either one or two hits. Which is to be expected, as the longer Sort would wait with tracking, the more bounding boxes would not be tracked resulting in mediocre precision rise and a major cost in recall.

In the Trajectory Prediction section, a custom made Camera Motion Estimator and a Trajectory Prediction modules were evaluated. Two additional metrics were introduced: Angle MAE (Mean Absolute Error) and Angle RMSE (Root Mean Square Error) in order to measure how well the solution is predicting the cyclist's real orientation angle. Four hyperparameters were tested: max-num-of-past-bbs-for-direction, max-num-of-past-bbs-for-avg-distance, angle momentum and correction vectors weight. The tests showed that generally the lower number of past bounding boxes used in interpolating the direction and distance in which the cyclist is moving, the better. For direction the best and lowest possible number was 2. For distance it was 2-3 for best AP and 5 for best angle MAE and RMSE. Camera motion correction vectors weight tests has shown that the optimal weight value depends on the recording. For test recordings 0012 and 0019 the best weight was 0.2, but for train recording 0015 the best were 0.5 and 1.0, a complete opposite. It shows either or both of two things: not enough data in the test set and some

issues or lack of hyperparameter optimization in the Camera Motion Module. Tests also showed that the best results on test set were obtained on the angle momentum = 0.

The final results show a decrease in accuracy as the prediction step increases, with F1 score of 0.658 at step 1 to 0.18 at step 10 on the test set. The results were significantly better on training set F1 score 0.919 on step 1 and 0.510 on step 10 (recordings 0015 and 0016), partly due higher YOLO object detection AP, but also recordings 0015 and 0016 include easier to predict cases of cyclists moving often perpendicularly to the camera which is not only easier to detect for YOLO, but also it is easier to predict it's trajectory, as the speed of the object on the frame is constant (if cyclist's real speed is constant and the camera is stationary). Also the decrease of the angle MAE from around 40 degrees before hyperparameter optimization to 22.13 degrees is quite remarkable.

In the next chapter the key findings, suggestions for future work and final conclusion will be discussed.

5. Conclusion and Future Research

In this final chapter this entire work is concluded with the findings summary, suggestions for future research and final conclusion.

5.1. Summary of findings

The analysis of the KITTI Object Tracking brought multiple insights. KITTI annotations were quite extensive, 2D and 3D bounding boxes, 3D location (position of the cyclists in a 3D space) and the observation angle of the object - which seemed visibly less accurate than a orientation angle extracted from the bottom of the 3D bounding box (top and center parts of the 3D bounding box did not seem to indicate the cyclists direction as well as the bottom one). Truncation annotations in KITTI have only two levels: truncated and not truncated, which in edge cases produces cyclists visible in 10%. In some cases it is just empty air, usually when the cyclist has just left the frame. In another case a cyclist was visible in 90% and was also marked as a truncated object. In that case it was better to just remove all truncated cyclists in order to avoid issues during training YOLO. Overall KITTI is a great dataset, with accurate annotations (although the bounding boxes sometimes do not coat the cyclists precisely, leaving visible gaps) but lacks a little in the number of cyclist annotations.

Tsinghua-Daimler cyclist dataset is quite large, over 22000 cyclist annotations. Unfortunately over 16000 of those that are in the training set are unusable due to missing annotations of cyclists. That originates from the 10% occlusion threshold set by the authors of the dataset. Which in effect causes any cyclist that is occluded in more that 10% to not have an annotation, but still appear on the frame anyway. Although the training set has lots of missing annotations, the valid and test sets are considerably more precise. With over 6000 cyclist annotations, 80% occlusion threshold and visibly precise bounding boxes with no gaps unlike KITTI it shows great potential for training an object detector. Though Tsinghua dataset does not have as extensive labels as KITTI has, only bounding boxes and cyclist ids.

Training Darknet YOLO models is a quite time consuming process on Windows. One has to first setup Nvidia CUDA Toolkit and cuDNN, build OpenCV with CUDA support using CMake and at last build Darknet. Whilst each step is suffering with incompatibilities between various OpenCV and CUDA versions, making the entire process long and hard to reproduce. A recommended approach would be to user either linux or Google Colab, in both cases the entire process takes only a few minutes instead of days of troubleshooting.

YOLO implementations in PyTorch are also considerably easier to install and to work with.

Predicting the trajectories of cyclists moving in parallel to the camera, either forward or backward, close to the frame center is significantly harder to predict than the cyclists moving perpendicularly to the camera. The reason behind it is quite simple, if the object is not moving on the frame significantly and the only movement that the Trajectory Prediction algorithm detects is a slight inaccuracy in bounding box detection and on top of that if the cyclist is moving towards the camera and camera additionally moves towards the cyclist, then in a matter of few frames cyclist's bounding box moves on the frame substantially quicker, accelerating, even though in reality he is moving at a constant pace. To tackle the moving camera issue the Camera Motion Estimator was implemented. By calculating the vanishing point and the optical flow of the surrounding buildings it returned an adjusted correction vector for each cyclist depending on where he is in the frame. Cyclists closer to the center are less affected by camera's forward and backward movement, than the cyclists closer to the sides of the frame. It did result in a minor improvement by reducing the Mean Squared Error (MSE) of the cyclist's orientation angle by 2% on the test set and no visible improvement of the Root Mean Squared Error (RMSE) on the test set. However, on the train set that included more perpendicular trajectories both MSE and RMSE had improved results of 2% and 1% respectively.

Predicting future bounding boxes on the other hand produces decent results if the cyclist is moving left or right on the frame, less so if it is towards or away from the camera being close to the center, as due to the camera perspective there is not big enough of a difference on the frame to predict the trajectory. Also the model predicting future bounding boxes of a cyclist that is moving towards the camera and is about to pass it, is not taking into account the acceleration of the object's bounding box that is produced on the frame, because of the camera's perspective, even though in reality the cyclist might be moving at constant speed. In other cases the model is yielding decent results, to an extent.

Implementing and integrating evaluation metrics should be performed at the early stages of any kind of work. Relying solely on visual evaluation during the method implementation may prove to be misleading and cause bad time management, and bad decisions. Having the evaluation metrics running from the beginning of the method implementation helps to see how each change in the method affects the overall results. It gives a better picture of how each element contributes to the overall method performance. Unfortunately evaluation metrics in this work for trajectory prediction were integrated only before the stage of testing and hyperparameter optimization. Which revealed that the impact of Camera Motion Estimator on the results was exaggerated while the impact of the Trajectory Prediction method was underestimated. Having put more time into researching better methods for trajectory prediction, instead of focusing solely on the Camera Motion Estimator, could have yielded significantly better results.

5.2. Suggestions for future research

As KITTI Object Tracking dataset provides 3D annotations for cyclists, which were utilized to extract the orientation angles of cyclists, it can be also be used to train a CNN model. Either a regression or a classification model (where an angle would be put into separate ranges: 0-30, 30-60.. degrees) to determine the cyclist orientation. By utilizing transfer learning on one of the models trained on the COCO dataset that contains humans and bikes could yield decent results, for example EfficientNetV2 [45]. It could then predict the real direction of the cyclist, mitigating the obstacle posed by camera movement.

Utilizing more data in training the YOLOv7 object detector would also prove beneficial to the overall results. Either by creating a custom cyclist dataset with varying illumination, orientation and occlusion or utilizing the existing datasets like Tsinghua-Daimler Cyclist dataset. Using only the validation and test data could be enough. Manually removing the frames with missing annotations from Tsinghua's training set could be another way to increase the amount of good data.

Instead of training YOLO to detect cyclists only, another approach could be to detect people and bikes as well. It could increase the test AP of YOLO as there would be not only more data in the training set, but the model would possibly be more eager to classify a person standing next to a bike as a person and a bike as separate objects instead of classifying them as a cyclist.

DeepSort [49] would be a great alternative to Sort as on top of basic object tracking provided by Sort it also has a built-in convolutional neural network that learns the features of the tracked objects, which the default Sort does not do. Knowing how the tracked cyclist looks like should help greatly if YOLO does not manage to detect a cyclist for some frames either due to occlusion, difficult illumination or for any other reason. Though using DeepSort could affect the performance negatively, so in that case switching back to faster YOLO models like the default YOLOv7 or YOLv7-tiny instead of YOLOv7x could compensate for that.

There is definitely place for improving both the inference speed and accuracy of Camera Motion Estimator and it's correction vectors. There were a few hyperparameters and values that were not optimized due to time constrictions. Also a different algorithm could be used for trajectory prediction. Instead of calculating the next bounding box centers linearly it could also include acceleration. Another extension would be to predict a trajectory curve instead of a line. The bounding boxes themselves instead of being fixed in size could also change according to past bounding boxes.

5.3. Conclusion

The goal of this work was to explore various machine learning models to find the most accurate way to predict the trajectory of cyclists in urban conditions. That includes finding a dataset, proposing and testing the prediction method and relating it to the state of the art.

Dataset chosen for that task was KITTI Object Tracking dataset containing almost 2000 annotated cyclists, located in a city of Karlsruhe, Germany. Two state-of-the-art object detectors were compared for the task: YOLOv4 and YOLOv7, with the latter yielding better detection results on the KITTI dataset. A solution for Trajectory Prediction was implemented. It consists of four main elements: Object Detection (YOLOv7), Object Tracking (Sort), Camera Motion Estimator (vanishing point, optical flow) and Trajectory Prediction (polynomial and linear interpolation) resulting in predicting future bounding boxes of cyclists in the next 1, 3 and 10 frames (as the framerate of KITTI dataset is 10 FPS, that results in: 0.1s, 0.3s and 1s steps ahead) and in predicting the real orientation angle of the cyclist. Various hyperparamters of these modules were thoroughly tested and optimized in order to improve the trajectory prediction accuracy. Resulting in AP_{50}^{test}=0.376, $AP_{50:95}^{test}$=0.182, $Precision_{50}^{test}$=0.56, $Recall_{50}^{test}$=0.462, F1 $score_{50}^{test}$=0.507, Angle MAE=22.14 and Angle RMSE=35.5 averaged over all steps (1, 3, 10) on the test dataset.

Bibliography

[1] Tsinghua-daimler cyclist detection benchmark. http://www.gavrila.net/Datasets/Daimler_Pedestrian_Benchmark_D/Tsinghua-Daimler_Cyclist_Detec/tsinghua-daimler_cyclist_detec.html. In Proc. of the IEEE Intelligent Vehicles Symposium (IV), Gothenburg, Sweden, pp.1028-1033, 2016.

[2] Andong87 and Wikimedia Commons. Harmonic mean 3d plot from 0 to 100. https://commons.wikimedia.org/wiki/File:Harmonic_mean_3D_plot_from_0_to_100.png, 2013.

[3] Bewley, A., Ge, Z., Ott, L., Ramos, F., Upcroft, B. . Simple online and realtime tracking. , 2016. In 2016 IEEE International Conference on Image Processing (ICIP) (pp. 3464-3468).

[4] BicycleDutch. Dutch Cycling. https://www.youtube.com/watch?v=ayPDlDi9Ug4, 2016. Youtube video used to test the method established in this work.

[5] L. Biewald and C. V. Pelt. Weights and biases. https://wandb.ai/site.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Darknet yolo - how to train (to detect your custom objects). https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects.

[7] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Darknet yolo- when should i stop training? https://github.com/AlexeyAB/darknet#when-should-i-stop-training.

[8] L. Bruce and K. Takeo. An iterative image registration technique with an application to stereo vision (ijcai). volume 81, 04 1981.

[9] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[10] M. Carranza-García, J. Torres-Mateo, P. Lara-Benítez, and J. García-Gutiérrez. On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data. *Remote Sensing*, 13(1), 2021.

[11] R. Chandra, U. Bhattacharya, A. Bera, and D. Manocha. TraPHic: Trajectory prediction in dense and heterogeneous traffic using weighted interactions. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019.

[12] R. Chandra, U. Bhattacharya, A. Bera, and D. Manocha. Traphic: Trajectory prediction in dense and heterogeneous traffic using weighted interactions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[13] R. Chandra, U. Bhattacharya, C. Roncal, A. Bera, and D. Manocha. Robusttp: End-to-end trajectory prediction for heterogeneous road-agents in dense traffic with noisy sensor inputs, 2019.

[14] Cyclist safety: an information resource for decision-makers and practitioners. Geneva: World Health Organization, 2020. Licence: CC BY-NC-SA 3.0 IGO.

[15] Dai, J.; Li, Y.; He, K.; Sun, J. R-FCN: Object Detection via Region-based Fully Convolutional

Networks., 2016. In Proceedings of the IEEE conference on Advances in Neural Information Processing, Barcelona, Spain, 5–10 December 2016; pp. 379–387.

[16] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:886–893, 2005.

[17] N. Deo and M. M. Trivedi. Convolutional social pooling for vehicle trajectory prediction, 2018.

[18] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[19] European Commission. Facts and Figures Cyclists. European Road Safety Observatory. Brussels, European Commission, Directorate General for Transport., 2021.

[20] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[21] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks, 2018.

[22] C. G. Harris and M. J. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[23] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. `https://arxiv.org/abs/1703.06870`, 2017.

[24] P. Henderson and V. Ferrari. End-to-end training of object class detectors for mean average precision, 2016.

[25] G. Jocher and Ultralytics. Ultralytics yolov8 github repository. `https://github.com/ultralytics/ultralytics`.

[26] G. Jocher and Ultralytics. Yolov5 pytorch github repository. `https://github.com/ultralytics/yolov5`.

[27] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[28] B. Keni and S. Rainer. Evaluating multiple object tracking performance: The clear mot metrics. `https://jivp-eurasipjournals.springeropen.com/articles/10.1155/2008/246309#citeas`, 5 2008. J Image Video Proc 2008, 246309 (2008). https://doi.org/10.1155/2008/246309.

[29] Korving, H.; Goldenbeld, Ch.; Schagen, I.N.L.G. van; Weijermars, W.A.M.; Bijleveld, F.D.; Wesseling, S.; Bos, N.M.; Stipdonk, H.L. Monitor Verkeersveiligheid 2016 - Achtergrondinformatie en onderzoeksverantwoording, 2016.

[30] V. Kress, J. Jung, S. Zernetsch, K. Doll, and B. Sick. Pose based start intention detection of cyclists. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2381–2386, 2019.

[31] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52, 1955.

[32] R. Kupers and M. Ptito. Compensatory plasticity and cross-modal reorganization following

early visual deprivation. *Neuroscience & Biobehavioral Reviews*, 41:36–52, 2014. Multisensory integration, sensory substitution and visual rehabilitation.

[33] 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. cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list.

[34] Liu, Shuo. Object trajectory estimation using optical flow. `https://digitalcommons.usu.edu/etd/462`, 2009. All Graduate Theses and Dissertations. 462.

[35] Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot MultiBox Detector., 2016. European Conference on Computer Vision; Springer: Cham, Switzerland, 2016; pp. 21–37.

[36] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3), 2021.

[37] Pucher J, Buehler R. Cycling towards a more sustainable transport future. Transport Reviews., 2017.

[38] K. Rahul. Vanishing point github repository. `https://github.com/KEDIARAHUL135/VanishingPoint`.

[39] Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection., 2015. arXiv 2015, arXiv:1506.02640.

[40] Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks., 2017. IEEE Trans. Pattern Anal. Mach. Intell. 2017, 39.

[41] G. Ross. Fast r-cnn. `https://arxiv.org/abs/1504.08083`, 2015.

[42] Schepers, P.; Stipdonk, H.; Methorst, R.; Olivier, J. Bicycle fatalities: trends in crashes with and without motor vehicles in the Netherlands, 2017.

[43] B. Sergei. map: Mean average precision for object detection. `https://github.com/bes-dev/mean_average_precision`.

[44] B. Sergei. map: Mean average precision for object detection. `https://github.com/bes-dev/mean_average_precision`.

[45] M. Tan and Q. V. Le. Efficientnetv2: Smaller models and faster training. *CoRR*, abs/2104.00298, 2021.

[46] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[47] C.-Y. Wang, A. Bochkovskiy, and M. L. Hong-Yuan. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022.

[48] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao. You only learn one representation: Unified network for multiple tasks. `https://arxiv.org/abs/2105.04206`, `https://resources.wolframcloud.com/NeuralNetRepository/resources/YOLOR-Trained-on-MS-COCO-Data/`, 2021.

[49] Wojke, N., Bewley, A., Paulus, D. . Simple Online and Realtime Tracking with a Deep

Association Metric, 2017. In 2017 IEEE International Conference on Image Processing (ICIP) (pp. 3464-3468).

[50] J. Zhang. Basic neural units of the brain: Neurons, synapses and action potential. `https://doi.org/10.48550/arxiv.1906.01703`, 2019.

[51] C. L. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 391–405, Cham, 2014. Springer International Publishing.

Figures

Tables