Computer Vision-Based Dangerous Scenes Detection System for Advanced Driving Assistance

Alberto Trabcchin

Supervisors

Prof. Dr. Yasutaka Fujimoto
Department of Electrical and Computer Engineering
Yokohama National University, Japan

Prof. Dr. Monica Reggiani
Department of Management and Engineering
University of Padova, Italy

Co-Supervisor

Prof. Dr. Stefano Michieletto
Department of Management and Engineering
University of Padova, Italy

Double Degree Master's Thesis

Department of Electrical and Computer Engineering
Yokohama National University
Japan
September 15, 2024

Acknowledgements

I am deeply grateful for the opportunity to conduct my research in the laboratory of Professor Yasutaka Fujimoto as part of a double-degree program. This experience has been profoundly enriching both personally and professionally. I would like to express my heartfelt thanks to the Professor for welcoming me into his laboratory and giving me the chance to meet and work with inspirational individuals, including Flavis, Besong, Martin, and Brice. Our spontaneous conversations about research and our futures were both enjoyable and enlightening.

Morning coffee talks with Eriko Hino were a wonderful way to start each day with her positive humor. I also appreciate the effort she puts in as the secretary of the laboratory.

My gratitude extends to Professor Roberto Oboe for coordinating this experience abroad and connecting me with the laboratory.

Inoue and Ushiyama were instrumental in helping me navigate a culture so different from my own. Their assistance made my transition much smoother.

I am profoundly grateful to Professor Monica Reggiani and her group, including Professor Stefano Michieletto, for their technical and moral support during challenging times in my research.

Special thanks to Elaine, Prateek, and Hassan from Magna International for their support from the company side during my thesis.

I am profoundly thankful to my family and my girlfriend, Giulia, who played a key role in this journey. Their unwavering support from day one has been indispensable.

Weekends spent having insightful talks with Marco on our shared interests were incredibly motivating. Also your support during difficult times helped me stay focused on my goals.

Lastly, but not least, I thank all the other new friends I made in Japan. Your presence and support have made this experience unforgettable.

Abstract

This thesis presents a comprehensive study on the development of a computer vision-based system for Advanced Driver Assistance Systems (ADAS). The research initially explored a classical computer vision approach, which involved employing detection and tracking algorithms and monocular depth estimation to perceive the external environment. Moreover, the study focused on integrating the driver's attention state through its gaze projected from a pair of eye-tracking glasses onto the external scene, through a roof-mounted camera. The primary objective was analyze the driver's behavior by comparing the its gaze with the locations of vulnerable road users, thereby proposing an initial safety scheme.

Despite the promising conceptual framework, it was observed that many of the conventional methods for extracting indirect features were not sufficiently robust in real-world driving scenarios. These methods struggled particularly under varying light conditions and during critical situations, highlighting significant limitations. In response to these challenges, the research transitioned to a deep learning-based approach. The core investigation centered on the capabilities of a Vision Transformer (ViT) in extracting human decision-making biases inherent in detecting dangerous driving situations. This approach was further augmented by employing semi-supervised learning techniques, which enabled the effective utilization of vast amounts of easily accessible unlabeled data, thus addressing the challenges associated with the expensive and labor-intensive labeling process.

The outcomes of this research illustrate the substantial potential of the attention mechanism, on which vision transformer is based, in enhancing the robustness and reliability of ADAS. The study also opens up new avenues for future research by identifying and addressing the challenges encountered during the development process. These findings underscore the critical role of computer vision in the advancement of ADAS, emphasizing its significance in improving driver assistance systems through more accurate and dependable perception mechanisms.

In conclusion, this thesis contributes to the field of ADAS by demonstrating how modern computer vision techniques can be effectively integrated into driver assistance systems. It highlights the potential of deep learning, especially in overcoming the limitations of traditional methods, and points to future innovations aimed at achieving safer and more efficient driving experiences.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ACC Adaptive Cruise Control

ADAS Advanced Driver Assistance Systems

AEB Automatic Emergency Braking

ASIFT Affine Scale-Invariant Feature Transform

AUC Area Under the Curve

BSD Blind Spot Detection

CAGR Compound Annual Growth Rate

CDF Cumulative Density Function

CNN Convolutional Neural Network

ConvAE Convolutional Autoencoder

ConvLSTM-AE Convolutional LSTM Autoencoder

DMS Driver Monitoring System

FPR False Positive Rate

GAN Generative Adversarial Network

LDW Lane Departure Warning

LKA Lane Keeping Assistance

LSTM Long Short-Term Memory

MCC Matthews Correlation Coefficient

MPL Meta Pseudo-Labels

OEM Original Equipment Manufacturer

p.d.f. Probability Density Function

ROC Receiver Operating Characteristic

SIFT Scale-Invariant Feature Transform

SL Supervised Learning

SURF Speeded-Up Robust Features

TPR True Positive Rate

TSR Traffic Sign Recognition

VAD Video Anomaly Detection

VAR Video Action Recognition

ViT Vision Transformer

Chapter 1

Introduction

1.1 Advanced Driver Assistance Systems

Advanced Driver Assistance Systems (ADAS) are technologies that improve road safety by enhancing both vehicle and driver capabilities. These systems support the driver in the driving process, reducing the likelihood of human error and increasing road safety. ADAS includes a range of features from basic functions such as automatic headlights and rain-sensing windshield wipers to more complex systems like adaptive cruise control, lane departure warning, and collision avoidance systems.

The evolution of ADAS has been fueled by advancements in sensors, computing power, and connectivity. These technologies work together to provide vehicles with the ability to not only sense their environment but also to analyze and respond to potential hazards. As such, ADAS is seen as a crucial step towards fully autonomous vehicles, providing a safer, more comfortable driving experience.

1.1.1 Market Size and Growth

The market for ADAS is experiencing robust growth as these technologies become more integral to new vehicle designs, emphasizing safety and driving efficiency.

In 2024, the global ADAS market is projected to be worth around $64.05 billion, with expectations to expand significantly at a Compound Annual Growth Rate (CAGR) of 12.7% reaching approximately $211.71 billion by 2034 [25]. This growth is given by advancements in autonomous driving technology and an increasing emphasis on vehicle safety from both consumers and functional safety certifications. Significant investments are also being made in the development and implementation of ADAS technologies across various global markets, including Japan, China, the United States, and Europe [25].

Considering the market value and growth rate of ADAS technologies, ADAS systems must also be economically viable on a large scale. This necessitates the use of cost-effective sensors such as cameras and radars, which offer a balance between performance and affordability. While there are more accurate options available, such as LiDAR sensors, these are typically more expensive and are primarily used for self-driving car experiments in research laboratories. The high cost of LiDAR sensors makes them less suitable for widespread deployment in consumer vehicles. Therefore, the challenge lies in leveraging affordable technologies like cameras and radars to their

Market	CAGR
Worldwide	12.7%
Japan	13.6%
China	13.5%
United States	12.5%
Canada	9.7%
Germany	8.8%
Spain	8.5%

Table 1.1: CAGR of the ADAS market in different countries [25].

maximum potential, to deliver effective ADAS systems that can be widely adopted.

1.1.2 Market Ecosystem

The market ecosystem for ADAS can be conceptually divided into several layers, each representing a different segment of the value chain. These layers range from Original Equipment Manufacturers (OEMs) to the suppliers of the key components such as sensors and processors. Here's a breakdown of these layers and the main companies in each field:

Original Equipment Manufacturers (OEMs)

Original Equipment Manufacturers are companies that integrate ADAS into their vehicles. They either develop some of their own ADAS technologies or incorporate systems designed by Tier 1 suppliers. Some of the leading OEMs in the ADAS market include Honda, Toyota, Nissan, General Motors, Tesla, Ford, Volkswagen, Volvo, BMW, and Mercedes-Benz.

Tier 1 Suppliers

Tier 1 suppliers develop complete systems or significant components that are directly supplied to OEMs. These companies often develop the software and hardware integration necessary for ADAS. Main Tier 1 suppliers in the ADAS market include Bosch, Continental, Aptiv, Denso, Magna, and Valeo.

Autonomous Technology Developers

This layer includes companies specifically focused on developing software and platforms that enable autonomous driving capabilities beyond standard ADAS features. Key players in this segment include Waymo (Google/Alphabet), Cruise (General Motors), Argo AI (Ford, Volkswagen), Aurora, and Zoox (Amazon).

Sensing and Perception

Companies in this segment focus on developing sensors and perception technologies that allow vehicles to perceive their environment, which is essential for both ADAS and

fully autonomous functionalities. Main providers are Luminar (LiDARs), Velodyne (LiDARs), Mobileye (cameras), Foresight Autonomous (cameras).

Processors and Computing

This segment involves the manufacturers of the advanced computing systems that process inputs from various sensors to make real-time driving decisions. Key players include NVIDIA (GPUs and SoCs), Intel (through Mobileye), Qualcomm (processors), NXP Semiconductors (microcontrollers), and Renesas (microcontrollers and SoCs).

Sensors

This layer includes manufacturers of the various types of sensors used in ADAS and autonomous vehicles, such as cameras, radar, LiDAR, and ultrasonic sensors. Main companies in this segment include Bosch (radar and video sensors), Continental (radar sensors), Sony (image sensors), Texas Instruments (radar chips), and ON Semiconductor (image sensors for automotive cameras).

1.1.3 Key Components of ADAS

ADAS systems are composed of several key components that work together to enhance vehicle safety and driving experience. These components include sensors, control units, actuators, human-machine interface, and connectivity.

Sensors

Sensors help the vehicle perceive its environment by collecting data on the surrounding objects and road conditions. The main types of sensors used in ADAS systems are:

- **Radar Sensors**: These sensors use radio waves to detect the distance and speed of objects around the vehicle. They are particularly useful for measuring the relative speed of other vehicles and detecting objects in poor weather conditions.
- **Cameras**: Cameras provide visual data that is used for object recognition, lane detection, traffic sign recognition, and other visual tasks. They are essential for understanding the vehicle's surroundings and identifying potential hazards.
- **Ultrasonic Sensors**: These sensors are used for close-range detection, primarily in parking assistance systems. They help the vehicle detect obstacles when maneuvering at low speeds.
- **LiDAR**: LiDAR sensors use laser pulses to create a 3D map of the vehicle's surroundings. They are particularly useful for creating detailed maps of the environment and detecting objects at longer ranges.

Control Units

Control units are responsible for processing the data from sensors and cameras, making real-time decisions, and sending commands to the vehicle's actuators. Decision making algorithms are implemented in these units to interpret sensor data and determine the appropriate response to different driving scenarios.

Actuators

Actuators are the components that control the vehicle's braking, steering, and acceleration systems based on the commands received from the control units. These actuators are responsible for executing the decisions made by the ADAS to ensure safe and efficient driving.

Human-Machine Interface

The human-machine interface is the system through which the driver interacts with the ADAS. This interface includes displays, alerts, and notifications that inform the driver about the status of the vehicle and provide warnings or assistance when needed. The interface is crucial for ensuring that the driver remains aware of the vehicle's behavior and can intervene when necessary.

Connectivity

Connectivity is an essential component of modern ADAS systems, enabling communication between the vehicle and external systems. This connectivity allows the vehicle to receive real-time traffic information, software updates, and remote assistance. It also enables vehicle-to-vehicle and vehicle-to-infrastructure communication, which is crucial for advanced safety features like collision avoidance and traffic management.

1.1.4 Common ADAS Features

ADAS systems offer a wide range of features that enhance vehicle safety, comfort, and efficiency. Some of the most common ADAS features include: Adaptive Cruise Control (ACC), Lane Departure Warning (LDW), Blind Spot Detection (BSD), Automatic Emergency Braking (AEB), Traffic Sign Recognition (TSR), Driver Monitoring System (DMS), parking assistance, adaptive headlights, and more. Most important functionalities are described in detail.

Adaptive Cruise Control (ACC)

ACC is a feature that maintains a set speed and adjusts it to keep a safe distance from the vehicle ahead. It uses sensors to detect the distance and speed of the vehicle in front and automatically adjusts the vehicle's speed to maintain a safe following distance.

Lane Departure Warning (LDW)

LDW is a system that alerts the driver if the vehicle begins to drift out of its lane without signaling. It uses cameras or sensors to monitor the vehicle's position within the lane and provides visual or audible warnings if the vehicle starts to veer off course. Some vehicles also have Lane Keeping Assistance (LKA) systems that can automatically steer the vehicle back into its lane if the driver does not respond to the warnings.

Blind Spot Detection (BSD)

BSD is a system that warns the driver of vehicles in the blind spot during lane changes. It uses sensors to detect vehicles in adjacent lanes that may not be visible in the side mirrors and provides visual or audible alerts to prevent collisions during lane changes.

Automatic Emergency Braking (AEB)

AEB is a safety feature that detects imminent collisions with vehicles, pedestrians, or other obstacles and automatically applies the brakes to prevent or mitigate the impact. This system helps reduce the severity of accidents by providing an additional layer of protection when the driver fails to react in time.

Traffic Sign Recognition (TSR)

TSR is a feature that uses cameras or sensors to identify traffic signs such as speed limits, stop signs, and road markings. It displays this information on the vehicle's dashboard or head-up display, helping the driver stay informed about the current road conditions and regulations.

Driver Monitoring System (DMS)

DMS is a system that monitors the driver's attention and alertness while driving. It uses sensors to track the driver's eye movements, head position, and other behavioral cues to detect signs of drowsiness, distraction, or inattention. The system can provide warnings or take corrective actions to prevent accidents caused by driver fatigue or distraction.

1.2 Focus of this work

This thesis delves deep into the development of a computer vision-based system to warn drivers about potential hazards on the road. The first part is more focused on monitoring the driver's behavior and attention through its gaze, and outside-world interactions with vulnerable road users. During this phase, an initial scheme to analyze the driver's gaze and its interaction with the environment is proposed. In particular, the model is based on extracting indirect features from sensors, location of vulnerable users during the time, projection of the driver's gaze to the scene, and estimation of the depth of the scene. However, we also highlight the limitations of this approach, mainly due to the complexity of the task, and the unreliability of some algorithms used for extracting features from the scene.

In the second part, we move to a deep learning-based approach to solve the problem of detecting potential hazards on the road. The idea is to have a model that can approximate human-like decision making adding some biases to the training data. We focus on training a model able to capture the spatial information of the scene, and to predict the presence of anomalies in the driving environment. We also explore the possibility of leveraging the large quantity of unlabelled data to increase the performance of predictions through semi-supervised learning techniques. At the end, we also explore the potential of few-shots learning with large pre-trained models, making

some experiments with GPT-4o. This is to demonstrate how the model's architecture is suitable for the required tasks with enough prior knowledge. It is also interesting to see how the model can adapt to the specific domain without changing its parameters and only thanks to the attention mechanism.

In the second part we also deeply discuss the training pipeline, which consists of data selection and pre-processing, model architecture, and evaluation metrics. In particular, we analyze how common evaluation metrics can be misleading when dealing with unbalanced datasets, and how Receiver Operating Characteristic (ROC) and Area Under the Curve (AUC) are more reliable metrics for evaluating the performance of the model. Therefore, we also provide a training method to deal with this type of datasets, that are prevalent in the field of anomaly detection in driving scenarios.

Another contribution is the discussion of possible future works, depending on the results obtained from the experiments and the limitations encountered during the development of the project.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

- **Chapter 2** provides an overview of the state-of-the-art methods and models for object tracking, image and video classification. It discusses how model complexities and relative computational costs increased in the last decade, comparing their performance on benchmark datasets. It also compares available datasets containing driving scenarios for different tasks. Finally, it goes deep into existing model architectures and datasets for anomaly detection in driving scenarios, which is closely connected to our work.

- **Chapter 3** presents main algorithms and architectures used for developing the project. It is mainly divided into two sections: traditional computer vision and deep learning-based techniques. The former includes explanations of the SIFT algorithm [32] and estimation of the homography matrix. The latter includes the Meta Pseudo-Labels algorithm [38] and the vision transformer architecture [11], with a focus on all stages involving the multi-head self-attention mechanism.

- **Chapter 4** describes all the methodologies and hypothesis for making the experiments; it is also divided in traditional computer vision and deep learning-based techniques. The former includes a general scheme of the driver's attention model, a description of how data is structured and processed, a brief comparison of possible methodologies for estimating the depth, and an explanation of how introducing spatial information of scenes is beneficial. The latter includes a general scheme for representing the task to be solved, the training pipeline for making experiments with the DR(eye)VE dataset, and data pre-processing for both DR(eye)VE [36] and BDD-100K [57] datasets.

- **Chapter 5** shows results obtained from the experiments made with the traditional computer vision and deep learning approaches. For the former case, it includes projection of the gaze, data distribution of the DR(eye)VE dataset, interaction between the driver's gaze and vulnerable road users, accuracy of the

monocular depth estimation of MiDaS [39], and tracking errors with ByteTrack [61]. It also discusses limitations of this method and motivations for moving to deep learning-based techniques. For the latter case, there are experiments comparing supervised and semi-supervised learning on the DR(eye)VE and BDD-100K dataset. There is also a further experiment on few-shots inference of the GPT-4o model with the BDD-100K dataset, to demonstrate potential performances of task-related knowledge distillation from pre-trained large models to smaller ones [22].

- **Chapter 6** concludes the thesis by summarizing the key contributions, discussing possible problems and limitations that affected the experiments, and outlining potential directions for future work. There is also a consideration of how driver's gaze data can be combined with deep learning-based techniques to warn the driver about potential hazards, considering realtime constraints and the need for a reliable system.

- **Appendix A** lists main custom algorithms used for the experiments, including data pre-processing and the training pipeline for updating the quantity of labelled samples in datasets.

- **Appendix B** provides a more formal and detailed explanation of the AUC-ROC from a probabilistic perspective. It is an helpful appendix to better interpret results on experiments with unbalanced datasets and to understand strenghts and limitations of available evaluation metrics.

Chapter 2

Related Work

We review recent work in behavioral intention prediction through standard methods, including object detection and tracking, and more advanced deep learning techniques. Moreover, we introduce advantages and limitations of some methods that are capable to improve scene understanding.

2.1 Available Datasets

Pedestrian detection and tracking are fundamental tasks in the field of computer vision applied to advanced driving assistance systems. If this task is innate to humans, it is still a challenging problem for machines. In addition to the complexity of the task many applications require real-time processing constraints which can affect accuracy of results.

In the computer vision field object detection can be approached through two f different perspectives. The former is to make predictions on general datasets that are used as benchmarks for the community, such as COCO [30]. In this case the goal is to develop a vision model capable to detect a wide range of objects, in many different scenarios. The latter is task-specific oriented, where the goal is to detect a relative small subset of objects in a specific context. This is the case of pedestrian detection in driving scenarios.

Nowadays, in the field of intelligent transportation systems, many datasets have been released to accomplish detection of some common objects, including pedestrians, cyclists, and different types of vehicles. The most common used for benchmarking state of the art algorithms are KITTI [16], Berkeley DeepDrive [57], Waymo Open Dataset [45] and nuScenes [3]. Moreover, each of these datasets has its own peculiarity, such as the quantity of video recordings, the variety of scenarios, the number of annotated objects, and the quality of ground-truth targets, that can be manually annotated or automatically generated.

Despite semi-autonomous driving is getting a lot of attention in the last years and it is expected to continue to grow, the state of the art algorithms are far away from the robustness and reliability that industry requires. Moreover, it has been shown that the performance can drastically drop when models are tested on different datasets, and some customized evaluation metrics could be required depending on the accomplished task [51]. Therefore it is fundamental to deserve as much attention to the quantity and quality of the data as to model design process.

2.2 Multi-Object Tracking

Multi-object tracking (MOT) aims to estimate bounding boxes of objects and associate them an unique ID across multiple time frames. This task is fundamental in perception of the environment because human behaviors are often time-dependent. For example it is important to predict pedestrians' trajectories to avoid collisions or to understand their intentions. Tracking is performed by estimating the future position of objects based on their past spatial and temporal information.

Object detection and tracking are distinct tasks, yet both are essential for the tracking process. Within tracking, there are two main aims: *"tracking by detection"* and *"detection by tracking"*. The former leverages powerful modern object detectors to increase tracking accuracy. Usually only bounding boxes' features are considered for the task. The latter, on the other hand, aims to imporve tracking accuracy with the help of tracking information. After the detection phase, most MOT methods discard some bounding boxes depending on a detection threshold [34, 46]; therefore all objects related to these boxes are lost for the considered time frame. There are also other methods, like ByteTrack [61], that are able to track partially-occluded objects in real-world scenarios with real-time performance.

For example, ByteTrack does not discard all bounding boxes below a certain confidence threshold, but it keeps almost every detection dividing them into high score ones and low score ones. At first high score detection boxes are matched with corresponding tracklets. Then, due to occlusions, resizes and motion blur, remaining unmatched tracklets are combined with low score detection boxes.

2.3 Image Classification

Image classification is a fundamental task in computer vision, as it is also used in the detection pipeline. In the last years, many models based on convolutional neural networks (CNNs) have been proposed, but nowadays research is moving towards more complex classfication tasks using different architectures. The most succesful CNN-based models are, in a chronological order, AlexNet [26], VGG [43], Inception [47], ResNet [20], DenseNet [23] and EfficientNet [48].

AlexNet [26] was among the pioneering models to incorporate deep convolutional layers, stacking five layers that increased the parameter count to 60 million. Additionally, it implemented several strategies to mitigate overfitting, including the application of dropout. Then VGG [43] still increased the number of convolutional layers to 16, reaching the state of the art in the ImageNet dataset [41]. However, its 138 million parameters made it computationally expensive. Inception [47], on the other hand, was able to increase the accuracy reducing the model size through multi-scale convolution transforms with filters of varying sizes. Transformations were applied to input features in parallel, and then concatenated to form the output. Therefore, it was able to capture diverse spatial information reducing the model size to 4 million parameters. Then ResNet [20] brought a revolutional change of CNN architectures, introducing the concept of *residual learning*. In particular it addressed the problem of vanishing and exploding gradients in very deep models by introducing skip connections between layers. This allowed to train a 152-layer CNN model that outperformed shallower models. After ResNet, DenseNet [23] introduced another novel aspect by fully con-

necting all convolutional layers with each other in a forward way. Unlike traditional CNN architectures where layers are connected in a sequential manner, DenseNet establishes direct connections between all layers in a feed-forward manner. This dense connectivity enhances feature reuse and facilitates gradient flow throughout the network, leading to improved parameter efficiency and feature propagation. As a result, DenseNet has shown significant improvements in performance and efficiency compared to earlier architectures. Finally, EfficientNet [48] proposed a new scaling method that uniformly scales depth, width, and resolution leading to a better performance. It was able to achieve state of the art results on ImageNet as well as on CIFAR-100 while being 8.4 times smaller and 6.1 times faster on inference than the best models.

Through this section it was possible to compare main improvements between different CNN architectures, and to understand how they evolved over time. However, by their nature, CNNs are design to extract features from images that can be associated to specific object patterns. Through some hyperparameters, like kernel size, downsampling, and number of layers, they can be adapted to extract features at different scales and with different resolutions. Considering scene understanding in driving scenarios, it is fundamental to caputre relative and absolute spatial relations of sub-areas in the main scene. Therefore, to address this problem, it is necessary to mode towards models designed with different architectures, capable to include these kind of information. One possible solution is using a vision transformer model (ViT) [11], that is based on the transformer architecture [52]. It consists of a self-attention mechanism that allows to tokenize sub-areas of the image and capture their spatial relations. This implies that relations are not just limited as local features, like with CNNs, but can be extended to all locations in the image through the tokens.

In Table 2.1 a comparison of the discussed models is reported. It is possible to see the trending increase of parameters and performance over time, but also how since Inception model size has been reduced thanks to the introduction of new techniques. It is also possible to notice how vision transformer achieves state of the art results with a large model size and computational cost, due to the connections between all tokens for the self-attention mechanism.

2.4 Video Classification

Along with image classification, time-series analysis started gaining attention. The initial focus was on univariate and multi-variate input data not directly related to images. The introduction of Recurrent Neural Networks (RNNs) [24] marked a significant improvement in analyzing this kind of data. Recurrent neural network represents the simplest sequential architecture, and its main peculiarity is the presence of a feedback loop that allows to save an internal state based on information from previous time steps. Then for each current time step the input is combined with the internal state. Despite the potential of RNNs, they are not able to capture long-term information from data sequences. This is due to the vanishing and exploding gradient problem, when there are many feedback loops in the network, and it is similar to the problem described for deep CNNs.

To address this issue, Long Short-Term Memory (LSTM) [21] was introduced. LSTM is a modified version of RNN capable to capture both long and short-term

Model	top-1 [%]	top-5 [%]	Params.	GFLOPS	Size [MB]	Year
AlexNet	56.522	79.066	61.1M	0.71	233.1	2012
VGG11	69.020	88.628	132.9M	7.61	506.8	2014
VGG13	69.928	89.246	133.1M	11.31	507.5	
VGG16	71.592	90.382	138.4M	15.47	527.8	
VGG19	72.376	90.876	143.7M	19.63	548.1	
Inception_v3	77.294	93.45	27.2M	5.71	103.9	2014
ResNet18	69.758	89.078	11.7M	1.81	44.7	2015
ResNet34	73.314	91.420	21.8M	3.66	83.3	
ResNet50	76.130	92.862	25.6M	4.09	97.8	
ResNet101	77.374	93.546	44.6M	7.8	170.5	
ResNet152	78.312	94.046	60.2M	11.51	230.4	
DenseNet121	74.434	91.972	7.9M	2.83	30.8	2016
DenseNet161	77.138	93.560	28.7M	7.73	110.4	
DenseNet169	75.600	92.806	14.2M	3.36	54.7	
DenseNet201	76.896	93.370	20.1M	4.29	77.4	
Effic.Net_b0	77.692	93.532	5.3M	0.39	20.5	2019
Effic.Net_b1	78.642	94.186	7.8M	0.69	30.1	
Effic.Net_b2	80.608	95.310	9.2M	1.09	35.2	
Effic.Net_b3	82.008	96.054	12.3M	1.83	47.2	
Effic.Net_b4	83.384	96.594	19.4M	4.39	74.5	
Effic.Net_b5	83.444	96.628	30.4M	10.27	116.9	
Effic.Net_b6	84.008	96.916	43.1M	19.07	165.4	
Effic.Net_b7	84.122	96.908	66.3M	37.75	254.7	
ViT-b_16	81.072	95.318	86.6M	17.56	330.3	2020
ViT-b_32	75.912	92.466	88.3M	4.41	336.6	
ViT-l_16	79.662	94.638	304.4M	61.55	1161.0	
ViT-l_32	76.972	93.070	306.6M	15.38	1169.4	
ViT-h_14	88.552	98.694	633.5M	1016.72	2416.6	

Table 2.1: Comparison of vision models for image classification on ImageNet

dependencies in data sequences through specific internal gates. A further improvement was introduced with Gated Recurrent Units (GRUs) [9], that are a simplified version of LSTM with fewer gates and parameters.

However, with the advent of image classification in computer vision, the interest in video analysis rose. Considering the previous work on RNNs, it was natural to extend the concept of time-series analysis to video time frames combining RNNs with CNNs. This led to the introduction of 3D CNNs that are capable to extract spatio-temporal features from video sequences through 3D convolutional kernels.

As for image classification, recent developments in the field focused on transformer-based models. In particular, the recent ViViT model [1] utilizes a similar concept of 3D CNNs, but applied to the vision transformer. The model stacks images of the video sequence and then applies the transformer architecture to capture spatio-temporal relations between them. While this approach outperforms 3D CNNs in terms of effectiveness, it comes with increased computational costs for training. Additionally, it is required to tune numerous hyperparameters.

2.5 Video Anomaly Detection

VAD is a challenging task that aims to identify patterns in data that do not conform to expected results. It is a more general task, not only focused in driving scenarios, but also used in medical imaging, surveillance systems, and industrial quality control. However, many of these techniques are cross-domain, meaning they can be applied and adapted to various fields beyond their initial area of application. Generally, the two main aspects to focus on are the availability of datasets and the development of effective models. In the following subsections we will list some of the most common datasets and models used in the field of VAD. We also distinguish training methods, supervised, unsupervised and semi-supervised, to accomplish different tasks.

However, VAD is usually performed in an unsupervised mode due to the scarcity of data. Therefore, when necessary, we will also include some works related to Video Action Recognition (VAR) that can be adapted to VAD.

2.5.1 Existing VAD Datasets

Considering the common techniques used in VAD, and the initial focus on surveillance systems, many more datasets have been released in this field compared to driving scenarios. Therefore, we divide the datasets into two categories: surveillance and driving-related datasets.

Surveillance Datasets

Li et al. proposed UCSD Ped1/Ped2 [28], a dataset of densely crowded pedestrian walkways to detect abnormal movements and prohibited objects. In particular, "Ped1" is based on 70 scenes where peple are walking toward and away from the camera. "Ped2" is composed of 28 scenes depicting pedestrians walking almost horizontally to the camera. Another dataset is Avenue [33], a dataset of 15 sequences with 14 unusual events, such as throwing objects, running, and loitering, with a total of 35,240 frames. Liu et al. proposed ShanghaiTech [31], a dataset of 437 videos (330 for training and 107 for testing) with 130 abnormal events and a total of 13 different scenes related to walking pedestrians. Finally, the UCF-Crime dataset [44] is composed of 1900 videos with 13 anomalies including robberies, accidents, and thefts.

Driving Datasets

Chan et al. introduced a new dataset with 678 dashcam accident videos taken from the web [6]; in each video, the last 10 frames are labelled as anomalous. Yao et al. proposed AnAn Accident Detection (A3D)[55], which contains 1,500 anomalous driving videos with the start and end frames' annotations. In total the dataset contains, respctively, 79,991 and 128,175 training and testing frames. There are 1,500 anomaly events that include targets like cars, trucks, bikes, pedestrians and animals.

Considering the interaction of the driver's gaze towards the environment, there is Driver Attention Prediction in Driving Accident scenarios (DADA) [13], [12], a dataset of 2,000 videos focused on the prediction of the driver's gaze in accident scenarios. In total they collected 658,476 frames, each labelled as normal or anomalous. A similar work was proposed by Palazzi et al. with DR(eye)VE [36], a dataset based on 74 videos,

8 drivers, over 500,000 frames, 6 total hours of driving in downtowns, countrysides and highways. However, DR(eye)VE is focused on prediction of the driver's gaze in safe scenarios, not in accident ones.

Xia et al. created the BDD-A [53] dataset, a subset of the Berkeley DeepDrive dataset [57] with 1,232 critical scenes and a total of 3.5 hours of driving. To create the dataset, they selected video clips that both included braking events and took place in busy areas, and then included 6.5 seconds before and 3.5 seconds after the braking event. They also collected gaze data of 45 partecipants, who were asked to watch the videos and press a button when they noticed an unsafe maneuver.

Finally, a recent dataset focused on detection of traffic accidents is DoTA [56]. It contains 4,677 videos with temporal, spatial, and categorical annotations. In total there are 9 different categories of accidents, including collision with other vehicles in different directions and out-of-control situations. Moreover, they double the classes, splitting each category to ego-involved and non-ego-involved accidents, resulting in 18 categories total.

2.5.2 State-of-the-Art VAD and VAR Models

Video Anomaly Detection models for driving scenarios can be divide into three main categories, depending on the available data and learning approach: unsupervised, semi-supervised, and supervised mode. Typically, it is easy to collect unlabelled data through ego-vehicle-mounted cameras or dashcams, while labelled data is much more difficult and expensive to obtain. Moreover, the quality of the labels can be affected by human subjectivity, age, and experience. Therefore, most of the research in this field is focused on unsupervised and semi-supervised learning.

Another imoprtant division is related to what features to extract: some models consider a single frame to analyze, while others take a sequence through a sliding time window.

Unsupervised Learning

Hasan et al. [19] proposed a Convolutional Autoencoder (ConvAE) model for reconstructing some stacked frames. Then anomalies were identified by comparing the reconstruction error. They fonud out that anomalies were better detected with longer sequences. To process the sequence with a Convolutional Neural Network (CNN), they input the data in a temporal cuboid format, converting RGB images to grayscale.

Medel et al. [8] and Luo et al. [35] used a spatio-temporal architecture, a combination of a CNN to extract spatial features for each frame and a Long Short-Term Memory (LSTM) to capture temporal dependencies between all the features. The model is also called Convolutional LSTM Autoencoder (ConvLSTM-AE). The end goal is also to reconstruct a sequence of frames, and anomalies are detected by comparing the reconstruction error. However, they tested the model on surveillance datasets, including Avenue [55] and UCSD [28].

Regarding driving scenarios, Yao et al. [56] introduced a *when-where-what* pipeline to detect, localize and recognize anomalous events from their custom dataset (DoTA). For the reconstruction task in unsupervised mode, they compare five different models described before: ConvAE [19], ConvLSTM-AE [35], AnoPred [31], TAD [55],

Supervised Learning

VAD problems can be addressed through supervised learning in different ways. On one hand, the model can be trained to make a binary classification (anomaly or not), or to detect the specific type of anomaly. The approach depends on the availability of data and the specific problem to solve.

A 3D CNN architecture was proposed by [50, 5, 10], for capturing spatio-temporal features from video sequences. For the action recognition task, they added a linear layer on top of the network.

Gao et al. [15] used a LSTM encoder-decoder architecture to predict future frames in a video sequence, like in the unsupervised approach. However, they also added a classification layer to predict the future action, depending on the ouptut predicted in the decoder stage.

Finally, Arnab et al. [1] proposed a pure-transformer based model for video classification. The model is called ViViT and it is able to capture spatio-temporal features without sequential stages. Also Huang et al. [1] used the same architecture, better analyzing how quality of data source, data preprocessing and augmentation affect the results.

Semi-Supervised Learning

Semi-supervised learning uses a small amount of labelled data and a large quanitty of unlabelled samples. However, there are different approaches to manipulate unsupervised data to improve the model's performance.

Shou et al. [42] trained a 3D CNN to detect starting windows of specific actions. Moreover, they used a Generative Adversarial Network (GAN) to generate new frames that are difficult to distinguish and they do not correspond to starting actions. Those frames are called *hard negatives* by the authors.

Xu et al. [54] used a pseudo-labelling method to improve video action recognition with a 3D CNN. They introduced an auxiliary lightweight model in addition to the backbone, to predict pseudo-labels for each other.

Zeng et al. [60] used an unique encoder-decoder architecture to reconstruct both labelled and unlabelled video sequences. Moreover, for labelled data, there is a classification layer to predict the action label.

Chapter 3

Background

3.1 Meta Pseudo-Labels

Large models usually tend to overfit on small labeled datasets, and semi-supervised learning can help to mitigate this issue by leveraging the information contained in the unlabeled data. However, to apply semi-supervised learning methods in practice, unlabeled data needs to carry useful information that is not already contained in the labeled dataset. This hypothesis can be formulated through three different assumptions: *smoothness*, *clusters*, and *manifold* [7].

In a more formal formulation, given $p(y|\boldsymbol{x}, D_L)$ as the probability distribution over the possible classes, given the input x and the labeled dataset D_L, the goal is to estimate $p(y|\boldsymbol{x}, D_L, D_U)$, where D_U is the unlabeled dataset. We can say that semi-supervised learning is effective only if $p(y|\boldsymbol{x}, D_L, D_U)$ is more accurate than $p(y|\boldsymbol{x}, D_L)$. If it is not the case, semi-supervised learning does not provide any additional benefit to the inference of the model, and it could even deteriorate the performance [7].

3.1.1 Smoothness Assumption

Assumption: If two examples are close in the input space, then they have similar outputs (labels).

The smoothness assumption implies that slightly changing the input data should not drastically change the output. Therefore, also the model is supposed to predict similar outputs for similar inputs. This assumption is fundamental in the propagation of the labels on the unlabelled data from the prior knowledge.

In Figure 3.1, on the left, the smoothness assumption is illustrated with an example on the moon dataset. The smoothness assumption is satisfied because similar points, in terms of the distance, have the same output.

3.1.2 Cluster Assumption

Assumption: Data points are orgnaized in clusters, and points that belong to the same cluster are more likely to have the same label.

This assumption supports methods like clustering followed by label propagation, where each cluster receives a common label, potentially inferred from a few labeled examples within or near the cluster. It does not imply that each class is related to

a unique cluster. On the other hand, it means that it is not possible to have two different classes in the same cluster. From this hypothesis, it is also derived the *low-density separation* assumption, which imposes that the decision boundary should lie in low-density regions between clusters.

In Figure 3.1, on the right, the cluster assumption is illustrated with 2D data points organized in clusters. Each cluster's point is more likely to have the same label.

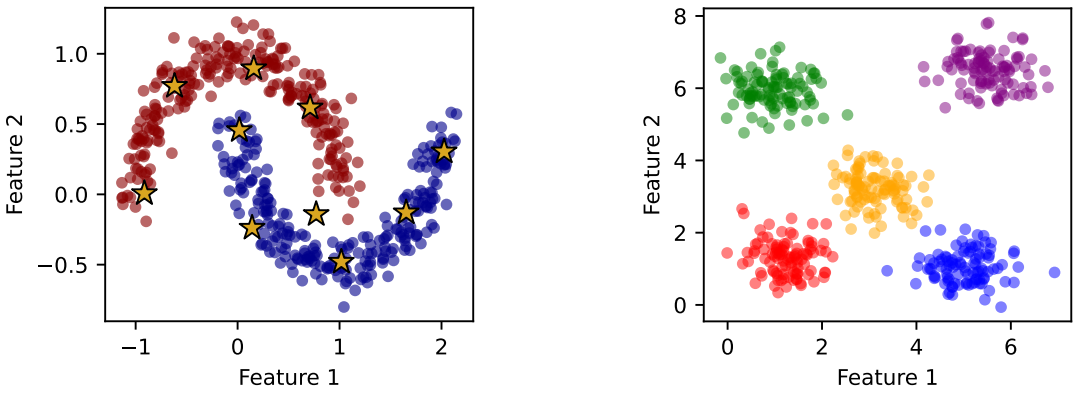


Figure 3.1: Smoothness and cluster assumptions. **Left**: The moon dataset representing smoothness between data points of each class. **Right**: The cluster assumption illustrated with a toy example.

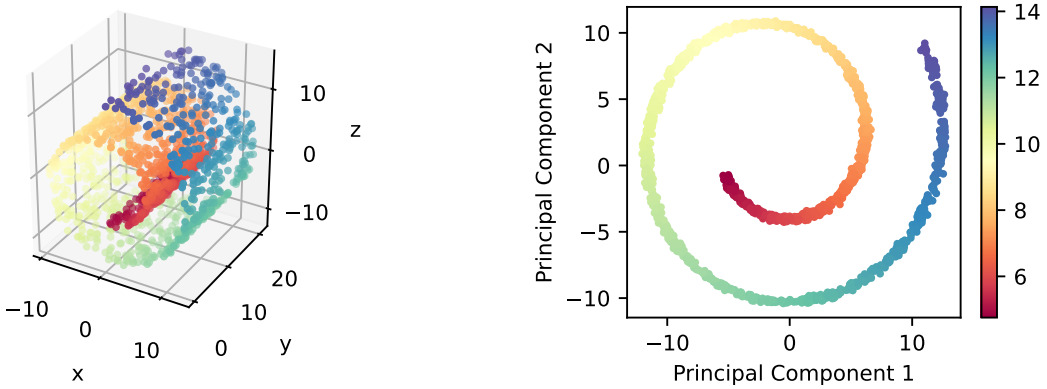


Figure 3.2: Illustration of the manifold assumption. **Left**: Data points are distributed in a high-dimensional space. **Right**: The data points lie on a lower-dimensional manifold.

3.1.3 Manifold Assumption

Assumption: The data lies on a low-dimensional manifold embedded in a high-dimensional space.

This assumption is based on the idea that the data is not distributed uniformly in the input space, but it is concentrated on a lower-dimensional manifold. It is fundamental to avoid problems related to the curse of dimensionality, which can lead to overfitting and poor generalization because of sparse data.

In Figure 3.2, the manifold assumption is illustrated with the Swiss roll dataset. The data points are distributed in a three-dimensional space, but they lie on a two-dimensional manifold. In general, the principal components analysis (PCA) can be used to reduce the dimensionality of the data, and in this specific case principal components are respectivlely the x and z axes.

3.1.4 The Training Algorithm

Classical pseudo-labeling methods usually train the teacher model on the labeled data and then they keep it fixed to predict the labels on the unlabeled data. The pseudo-labels are then used to train the student model, which is a copy of the teacher model. The student model is finally trained on the pseudo-labelled data.

On the other hand, on Meta Pseudo-Labels [38], the teacher is trained along with the student model. In particular the student model is trained on the pseudo labels generated by the teacher model, and the teacher is trained on the performance of the student model on the labeled data. This process is repeated until the student model converges reaching a better performance with respect to the teacher. Therefore, the teacher model is updated to maximize the performance of the student.

In a more formal way, Meta Pseudo-Labels aims to optimize the student's loss function $CE(T(x_u, \theta_T), S(x_u, \theta_S))$, with the pseudo-labels generated from the teacher on the unlabeled data:

$$\begin{aligned} \theta_S^{PL} &= \arg\min_{\theta_S} \mathbb{E}_{x_u} \left[CE(T(x_u, \theta_T), S(x_u, \theta_S)) \right] \\ &:= \arg\min_{\theta_S} \mathcal{L}_u(\theta_T, \theta_S) \end{aligned} \tag{3.1}$$

where $T(x_u, \theta_T)$ is the pseudo target generated by the trained teacher model with its parameters θ_T already optimized and fixed. Therefore, the final goal of Pseudo Labels is to have a lower student loss with respect to the teacher on the labelled data:

$$\begin{aligned} \mathbb{E}_{x_l, y_l} \left[CE(y_l, S(x_l, \theta_S^{PL})) \right] \leq \mathbb{E}_{x_l, y_l} \left[CE(y_l, T(x_l, \theta_T)) \right] \quad &\iff \\ \mathcal{L}_l(\theta_S^{PL}) \leq \ \mathcal{L}_l(\theta_T) \end{aligned} \tag{3.2}$$

Combining Equations (3.1) and (3.2) it is possible to notice that the student loss $\mathcal{L}_l(\theta_S^{PL})$ depends on the teacher parameters θ_T. Therefore, it could be possible get a better optimization of the teacher model to reduce the student loss. This is the main idea behind Meta Pseudo-Labels. The final student loss $\mathcal{L}_l^*(\theta_S^{PL})$ will be the following:

$$\begin{aligned} \mathcal{L}_l^*(\theta_S^{PL*}) &= \min_{\theta_T} \mathcal{L}_l(\theta_S^{PL}(\theta_T)) \\ \text{where} \qquad \theta_S^{PL}(\theta_T) &= \arg\min_{\theta_S} \mathcal{L}_u(\theta_T, \theta_S) \end{aligned} \tag{3.3}$$

However, calculating the gradient $\nabla_{\theta_T} \theta_S^{PL}(\theta_T)$ is computationally expensive and complicated to get because it requires to unroll all the previous training of the student, given that θ_S^{PL} was updated multiple times with a combination of the fixed θ_T. To overcome this issue, Meta Pseudo-Labels uses a meta-learning approach to approximate the computation of $\theta_S^{PL}(\theta_T)$ to a single step:

$$\theta_S^{PL}(\theta_T) \approx \theta_S - \eta_S \nabla_{\theta_S} \mathcal{L}_u(\theta_T, \theta_S) \tag{3.4}$$

where η_S is the learning rate used for training the student model. In this way, the objective function to optimize the student introduced in Equation (3.3) can be approximated as follows:

$$\mathcal{L}_l^*(\theta_S^{PL}) \approx \min_{\theta_T} \mathcal{L}_l(\theta_S - \eta_S \nabla_{\theta_S} \mathcal{L}_u(\theta_T, \theta_S)) \tag{3.5}$$

The final step that distinguishes Meta Pseudo-Labels from other pseudo-labeling algorithms is to not keep the teacher fixed during the training of the student. With the approximation introduced in Equation (3.4), the objective function can be updated depending on the information provided only at the previous iteration. Therefore, the teacher is trained along with the student. The combined training can be summarized in the following steps:

- **Student**: Sample a batch from unlabeled data x_u, get the corresponding pseudo-labels $T(x_u, \theta_T)$, and update the parameters: $\theta_S \leftarrow \theta_S - \eta_S \nabla_{\theta_S} \mathcal{L}_u(\theta_T, \theta_S)$.
- **Teacher**: Sample a batch from labeled data (x_l, y_l) and use the new student's parameters to optimize the objective function: $\theta_T \leftarrow \theta_T - \eta_T \nabla_{\theta_T} \mathcal{L}_l(\theta_S)$.

3.2 The Vision Transformer

Transformers, originally developed for natural language processing tasks, have revolutionized the way machines understand and generate text. This model architecture, introduced by Vaswani et al. in the paper "Attention is All You Need" [52] relies on a mechanism known as self-attention to process data sequences in parallel, significantly improving efficiency and scalability compared to prior methods that used recurrent layers.

Building on the success in NLP, the concept of transformers has been adapted for image recognition tasks, marking a significant departure from the conventional convolutional neural networks (CNNs) that have dominated the field for years. This adaptation was most notably realized in the Vision Transformer (ViT) model introduced by Dosovitskiy et al. [11]. Their research demonstrates how a pure transformer applied directly to sequences of image patches can perform at or above current state-of-the-art levels on major image recognition benchmarks like ImageNet.

This approach challenges the prevailing reliance on CNNs for image tasks, suggesting that transformers can equal or exceed CNN performance on benchmarks with sufficient training data and computational power. The implications of this finding are fundamental, indicating a potential shift towards more flexible and scalable models for not only image classification but potentially other computer vision tasks as well. Moreover, it proves that the transformer's ability to handle spatial data can be as effective in visual tasks as it has been in processing text.

In their paper, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" [11] the authors propose a straightforward yet highly effective approach: an image is divided into fixed-size patches, these patches are linearly embedded, and positional embeddings are added to retain positional information. The resulting sequence of vectors is then fed into a standard transformer architecture, similar to that used for NLP tasks. The model is trained end-to-end for image classification tasks, leveraging extensive pre-training on large datasets followed by fine-tuning on targeted smaller datasets.

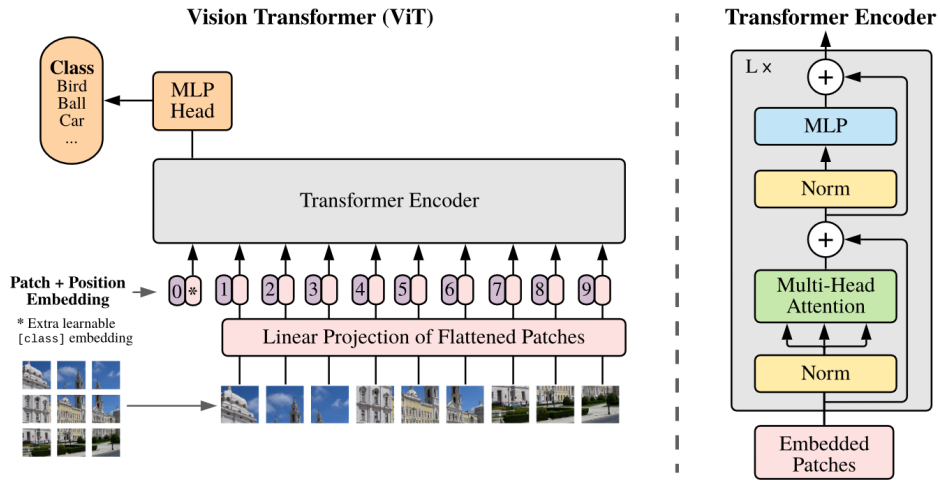


Figure 3.3: The Vision Transformer architecture [11]. The input image is divided into fixed-size patches, linearly embedded, and positional embeddings are added. The resulting sequence of vectors is fed into the transformer encoder architecture. On the top a feed-forward layer performs the classification.

3.2.1 Patch Embeddings

Patch embedding is a key process that transforms raw image data into a format suitable for the transformer architecture. An image is first divided into small patches, which can be considered as sub-areas of the input image.

Each patch is then flattened and projected into a higher-dimensional space through a linear transformation. This creates dense vector representations of the patches, like the word embedding process in NLP. Positional embeddings are added to these patch embeddings to preserve their spatial relationships, allowing the transformer to understand the layout of the image as it processes the sequence of embedded patches.

This method adapts the transformer's powerful self-attention mechanisms to handle and interpret spatial relations of visual data.

3.2.2 Self-Attention Mechanism

The core of the transformer architecture is the self-attention mechanism, which allows the model to weigh the importance of different parts of the input data. It is worth noting that the fundamental concept of self-attention does not require any trainable parameters. It is actually computed only through the embedded patches.

Considering a sequence of embedded patches $\{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_t}\}$ as input, the self-attention is a sequence-to-sequence operation that outputs $\{\boldsymbol{y_1}, \boldsymbol{y_2}, ..., \boldsymbol{y_t}\}$. The goal is to have each output vector $\boldsymbol{y_i}$ that represents a similarity between the i-th patch and all the input sequence of patches. In particular it can be described by a weighted sum of the input sequence

$$\boldsymbol{y_i} = \sum_{j=1}^{t} w_{ij} \boldsymbol{x_j} \tag{3.6}$$

where $\boldsymbol{w_i} := \{w_{ij}\}_{j=1}^{t}$ is the set of weights that describes the importance of each input vector for the i-th one. Therefore each i-th vector has a different set of weights with

respect to the others.

The simplest way to compute the similarity is to use the dot-product between the input vectors $w'_{ij} = \boldsymbol{x_i^T x_j}$. Moreover, to guarantee them to have a limited range in $(0, 1)$ and to sum up to 1, the softmax function is applied:

$$w_{ij} = \boldsymbol{\sigma}(\boldsymbol{w'_i})_j = \frac{e^{w_{ij}}}{\sum_j e^{w_{ij}}} \tag{3.7}$$

From Equation (3.7) it is possible to notice that the weights depend only on the input vectors, and they are computed independently. That means that the self-attention mechanism is a parallel operation that does not consider the order of the input sequence. If the input sequence order is changed then the output sequence will change the order as well, keeping the same similarity results.

3.2.3 Queries, Keys and Values

In the self-attention task, each input vector $\boldsymbol{x_i}$ plays three different roles:

- **Query**: $\boldsymbol{x_i}$ is used to compute the similarity to all the other vectors in order to calculate y_i.
- **Key**: $\boldsymbol{x_i}$ is used to compute the similarity of all the other vectors with respect to itself, in order to calculate all the other outputs.
- **Value**: $\boldsymbol{x_i}$ is used to compute the weighted sum of the input vectors, given the weights already computed.

This method introduces some controllable parameters that can be learned during the training to accomplish the respective tasks. In particular, the matrices are $\boldsymbol{W_q}, \boldsymbol{W_k}, \boldsymbol{W_v}$ and they all have dimesion (q_k, q_k), given q_k the embedding dimension of patches. Equations (3.6) and (3.7) can be rewritten as follows:

$$\begin{cases} w_{ij} = \boldsymbol{\sigma}(\boldsymbol{K^T q_i})_j \\ \boldsymbol{y_i} = \sum_j w_{ij} \boldsymbol{v_j} \end{cases} \qquad \text{where} \quad \begin{cases} \boldsymbol{q_i} = \boldsymbol{W_q x_i} \\ \boldsymbol{K} = \boldsymbol{W_k X} \\ \boldsymbol{v_j} = \boldsymbol{W_v x_j} \\ \boldsymbol{X} = (\boldsymbol{x_1}, ..., \boldsymbol{x_t}) \end{cases} \tag{3.8}$$

It is worthwile ntoicing that it is easier to consider the queries and values vectors, and the keys matrix. That is because the query is referred a specific input vector that is used to compute the similarity with all the others. The value is still directly related to the input vector used to compute the weighted sum. On the other hand, the keys are related to all the input vectors. Therefore, multiplying a query with the keys matrix will give all the similarities for the selected vector.

Adding trainable parameters to the self-attention mechanism allows to figure out the importance of all the sub-areas of the input image, depending on the bias we use to label the dataset. Then it is possible to use the attention mask, to highlight the importance of some specific areas of the image.

Finally, dot products between rows of $\boldsymbol{K^T}$ and $\boldsymbol{q_i}$ typically grow with the dimension of the embedding, and it can lead to instabilities in terms of gradients during the

learning process. Therefore it is common to normalize the dot product by the square root of the embedding dimension $\sqrt{d_k}$, considering that it is also the grow rate of the Euclidean distance of the input vectors.

In a more general matrix form, Equation (3.8) with the normalization can be described as follows:

$$Y = \boldsymbol{\sigma}\left(\frac{\boldsymbol{K^T Q}}{\sqrt{\boldsymbol{d_k}}}\right) V \qquad \text{where} \quad \begin{cases} \boldsymbol{Q} = \boldsymbol{W_q X} \\ \boldsymbol{K} = \boldsymbol{W_k X} \\ \boldsymbol{V} = \boldsymbol{W_v X} \\ \boldsymbol{X} = (\boldsymbol{x_1}, ..., \boldsymbol{x_t}) \end{cases} \tag{3.9}$$

In Equation (3.9) the softmax function $\boldsymbol{\sigma}$ is applied to the rows of its argument.

3.2.4 Multi-Head Self-Attention Mechanism

In an image each patch can have different relations with the others, depending on the context. Therefore, using only one self-attention mechanism would sum all the information together, limiting the ability to capture the scene context. To overcome this issue, the multi-head self-attention mechanism is introduced. It is a parallel operation based on the single self-attention mechanism, which introduces multiple weight matrices $\boldsymbol{W_q^r}, \boldsymbol{W_k^r}, \boldsymbol{W_v^r}$, also called *attention heads*.

It is also possible to keep the same number of parameters of the single self-attention mechanism in an efficient way. The main idea is to project queries, keys and values to a lower-dimensional space according to the number of heads. In particular, given h the number of attention heads, d_k the embedding dimension of the patches, we can use a transformation matrix $\boldsymbol{T^r}$ of dimension $(d_k, d_k/h)$. Therefore, the total number of parameters for the single-head and multi-head self-attentions will be:

$$\begin{aligned} \#\text{params SHA} &= 3 \cdot d_k^2 \\ \#\text{params MHA} &= 3 \cdot h \cdot d_k \cdot \frac{d_k}{h} = 3 \cdot d_k^2 \end{aligned}$$

In Figure 3.4 the multi-head self-attention mechanism is schematically represented, and data flow is from the bottom to the top. The input patches are first embedded and projected to a lower-dimensional space. Therefore we get queries, keys and values for each attention head. The self-attention mechanism is applied to each head, and the outputs are concatenated together. In this way we augment the information of the spatial relation of each patch. Finally, a linear transformation is applied to get the final output of the multi-head self-attention mechanism, which describe the attention of each patch with respect to all the others.

To avoid any misunderstanding on the generation of the queries, keys, values and on the concatenation, data is represented through channels, where each channel is related to a specific attention head and it is independent from the others.

It is also worth noticing that the transformer architecture leverages on residual connections and layer normalization to stabilize the training process, avoiding the vanishing gradient problem. The residual connections are used to sum the output of the multi-head self-attention mechanism with the embedded patches, and the layer normalization is applied to the sum.

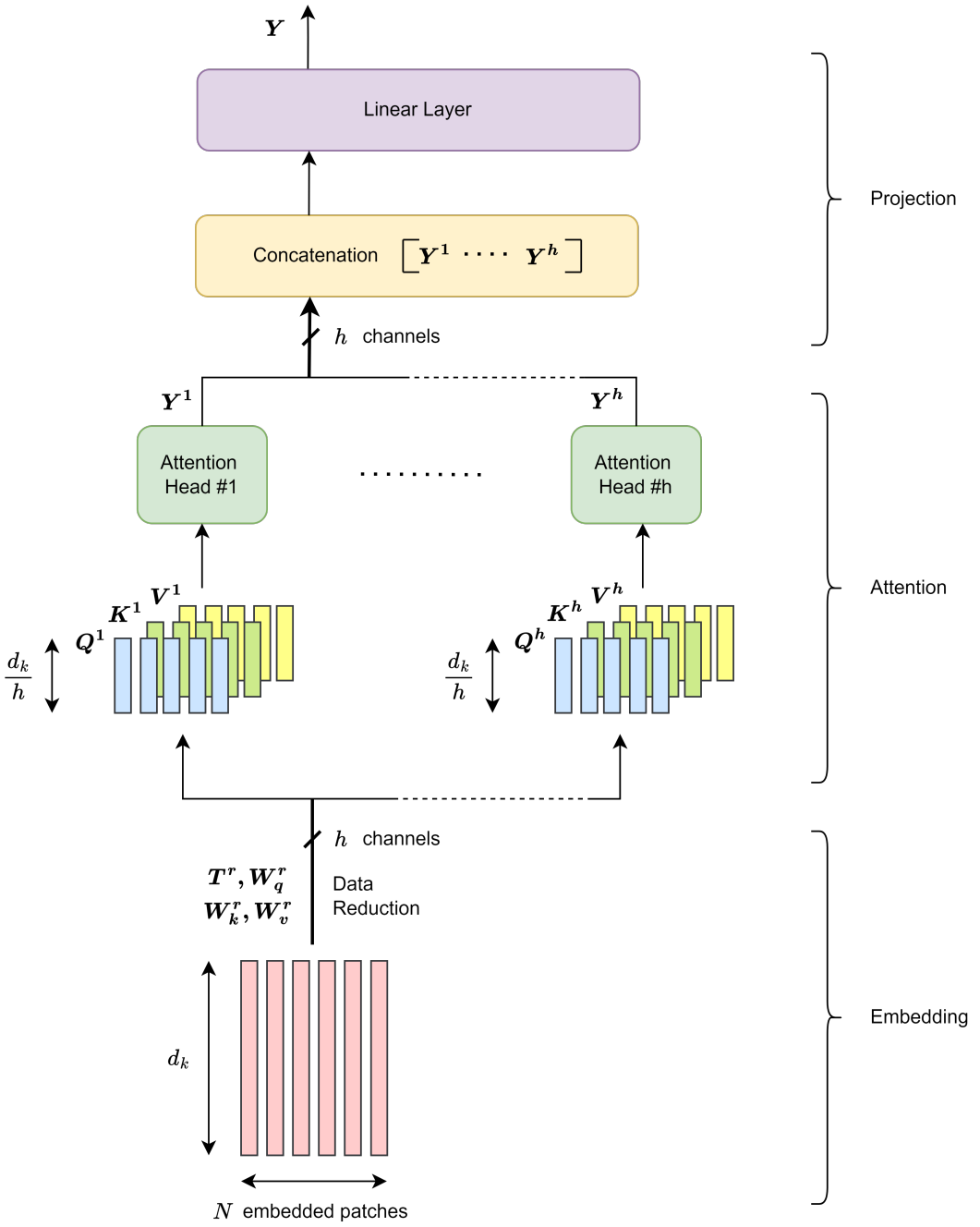


Figure 3.4: Multi-head self-attention mechanism schematically represented with the main blocks: patch embeddings, dimension reduction, attention heads, data concatenation, and final linear transformation.

3.2.5 Classification Transformer

If the final task is a classification of the input image, it is necessary to have a feed-forward layer on the top of the transformer encoder, as shown in Figure 3.3. Therefore, the final output will depend on the relative relation of the patches and their absolute location in the image.

3.3 Traditional Computer Vision Techniques

In this section we refer to some methods that have been used for a long time in computer vision tasks, even before the deep learning era. These methods are still used in some specific tasks, usually related to multiple view geometry. In particular, the following methods are described: the Scale-Invariant Feature Transform (SIFT) [32], the Random Sample Consensus (RANSAC) [14], and the optimization algorithm to estimate the homography between two images.

3.3.1 Scale Invariant Feature Transform (SIFT)

The SIFT algorithm is a method to detect and describe local features in images. It was introduced by David Lowe in 2004 [32], and it is still used in computer vision tasks thanks to its robustness to changes in illumination, rotation, and scale. However, being computationally expensive, it is difficult to integrate it in real-time applications. Therefore other versions of the algorithm have been developed, like Speeded-Up Robust Features (SURF) [2] and Affine Scale-Invariant Feature Transform (ASIFT) [58]. These algorithms have also been implemented in computer vision libraries, like OpenCV, to leverage the power of parallel computing of GPUs.

The SIFT algorithm is based on the following steps: scale-space peak selection, keypoints detection, and assignment of descriptors to the keypoints.

Scale-Space Peak Selection

To make the algorithm robust and scale-invariant, the first step is to analyze the image at different scales, applying a Gaussian filter with different standard deviations. Given an image $I(x, y)$, the scale-space representation is defined as:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{3.10}$$

where $G(x, y, \sigma)$ is the Gaussian kernel with standard deviation σ. The scale-space representation is computed for different values of σ, and the process is repeated for different octaves of the image. An octave is a set of images, each one with a resolution half of the previous one. The number of images for each octave is not standard, and it depends on the resolution of the original image.

The scale-space representation is used to detect the local maxima and minima of the difference of Gaussian (DoG) function. The DoG function is computed as the difference between two consecutive images of the same octave (let's say they have standard deviations σ and $k\sigma$):

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{3.11}$$

The local maxima and minima of the DoG function are detected by comparing each pixel with its eight neighbors in the same image and the nine neighbors in the previous and next images (26 in total). If the pixel is a local maxima or minima, it is considered a keypoint. To be a maxima or minima, the candidate pixel must be greater or smaller than all the neighbors, respectively.

Keypoints Localization

The scale-space peak selection produces many keypoints, but not all of them are stable under changes in illumination, rotation, and scale. Therefore, the keypoints are filtered by applying the Taylor expansion to the DoG function for each candidate keypoint. The Taylor expansion is used to estimate the actual maxima or minima of the DoG function, approximated by the second-order derivative of the function with respect to the selected keypoint.

Considering the DoG function D related to a keypoint, and an offset along all the dimensions $\boldsymbol{x} = (dx, dy, d\sigma)^T$, to find accurate location of the selected keypoint a second-order Taylor expansion is computed as follows:

$$D(\boldsymbol{x}) = D + \frac{\partial D}{\partial \boldsymbol{x}}^T \boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^T \frac{\partial^2 D}{\partial \boldsymbol{x}^2}\boldsymbol{x} \tag{3.12}$$

Minima, or maxima, $\hat{\boldsymbol{x}}$ is detected by solving the equation:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} D(\boldsymbol{x}) = \arg\{\nabla_{\boldsymbol{x}} D(\boldsymbol{x}) = \boldsymbol{0}\} \tag{3.13}$$

Keypoints with low intensity are filtered out, depending on a threshold. Therefore, the more accurate keypoint $\hat{\boldsymbol{x}}$ is discarded if:

$$D(\hat{\boldsymbol{x}}) < \text{threshold} \tag{3.14}$$

To remove the keypoints that are located on edges, on the other hand, the ratio between the principal curvatures of the DoG function is computed. If the ration is greater than a threshold, it means that there is a high variation of intensity on one direction with respect to the other (typical of an edge). Otherwise the keypoint corresponds to a corner and it is kept. Principal curvatures are given by the eigenvalues λ_1 and λ_2 of the Hessian matrix of the DoG function, and the ratio is computed as follows:

$$\frac{\text{Tr}(H)^2}{\text{Det}(H)} = \frac{(r+1)^2}{r} < \text{threshold} \qquad \text{where:} \quad r = \frac{\lambda_1}{\lambda_2} \tag{3.15}$$

It is possible to notice that the evaluation function in Equation (3.15) has a minimum when the two eigenvalues are equal, and it grows when the ratio between the two eigenvalues increases.

Keypoint Orientation

To make the keypoints invariant to rotation of the image, the orientation with respect to the neighbor pixels is computed. The orientation is computed through two terms,

the magnitude and the orientation of the gradient of the correspondent DoG function:

$$m(x,y) = \sqrt{[D(x+1,y) - D(x-1,y)]^2 + [D(x,y+1) - D(x,y-1)]^2}$$
$$\theta(x,y) = \arctan\left(\frac{D(x,y+1) - D(x,y-1)}{D(x+1,y) - D(x-1,y)}\right)$$

The angle θ is computed in the range $[0, 2\pi]$, and it is quantized in a histogram with 36 bins. The assigned orientation consists of the peak of the histogram, and all the other peaks that are greater than 80% of the maximum.

Keypoint Descriptor

After having assigned the orientation to the keypoints, the final step is to assign a unique descriptor to each keypoint. A grid of 16x16 pixels is considered around the keypoint, and it goruped into 4x4 macro-areas. For each macro-area, the gradient magnitude and orientation are computed, and they are grouped into 8 bins. Therefore, the descriptor is a 128-dimensional vector (4x4x8).

To make the descriptor invariant to rotation, the orientation of the keypoint is subtracted from each gradient orientation. Moreover, to make the descriptor invariant to illumination changes, each magnitude is truncated to a maximum thrashold, and then normalized.

Matching Keypoints

To match similar keypoints in two different images, the Euclidean distance between the descriptors can be computed. However, considering a descriptor, there could be more than one similar match on the other image (i.e. the second best match could be close to the best). To overcome this issue, the ratio between the best and the second best match is computed. If the ratio is less than a threshold, the match is considered valid.

3.3.2 Projective Transformation

A homography is the transformation between two planes under perspective projection. This means that a planar object in transformed by a homogeraphy into its image sensor. However, the transformation can also be seen between two cameras, if all points in the Euclidean space lie on the same plane.

In Figure 3.5 the homography transformation between two camera planes is represented. Two cameras, with their own system of coordinates, are pointing to the same plane. All points in the Euclidean space lie on the same plane, and they are projected on the two cameras' planes. The homography matrix $\boldsymbol{H}$ is the transformation of the projected points between the two camera planes.

Considering a general point $P_i \in \mathbb{R}^3$ in the Euclidean space, lying on a plane, and its correspondent projections on the two cameras' planes $\boldsymbol{p_i}, \boldsymbol{p'_i} \in \mathbb{R}^2$. Without focusing on a specific point, and without loss of generality, projection can be described by the following equation:

$$\tilde{\boldsymbol{p}} \sim \boldsymbol{H}\tilde{\boldsymbol{p}}' \qquad \boldsymbol{H} \in \mathbb{R}^{3\times3}, \quad \tilde{\boldsymbol{p}}, \tilde{\boldsymbol{p}}' \in \mathbb{P}^2 \tag{3.16}$$

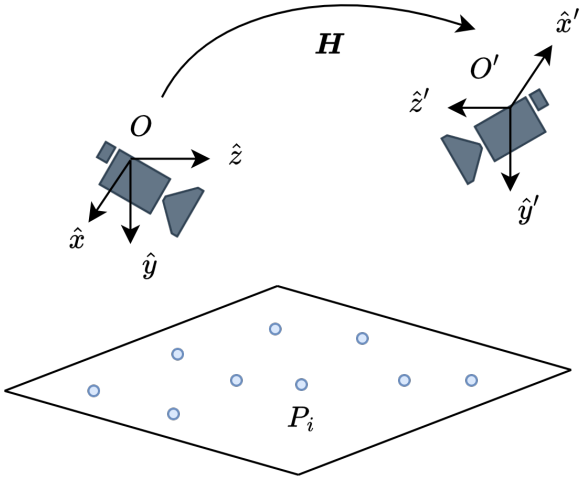


Figure 3.5: Homography between two planes. The transformation matrix defines the projection between the camera planes. All points in the Euclidean space lie on the same plane.

where $\tilde{\boldsymbol{p}}$ and $\tilde{\boldsymbol{p}}'$ are homogeneous coordinates of the points $\boldsymbol{p}$ and $\boldsymbol{p}'$ in the 2D projective space $\mathbb{P}^2$. It is worth to notice that $\boldsymbol{H}$ is the homography matrix, and it is defined up to a scale factor. Therefore, to fully define it, at least four correspondent points are required. Considering homogeneous coordinates of the points, Equation (3.16) can be rewritten as follows:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{3.17}$$

Homography Estimation

Given the i-th correspondance between two images, defined by the set $\{(x_i, y_i), (x'_i, y'_i)\}$, constraints described in Equation 3.17 are:

$$\begin{cases} x'_i = h_{11}x_i + h_{12}y_i + h_{13} \\ y'_i = h_{21}x_i + h_{22}y_i + h_{23} \\ 1 = h_{31}x_i + h_{32}y_i + h_{33} \end{cases} \iff$$

$$\begin{cases} x'_i \left(h_{31}x_i + h_{32}y_i + h_{33}\right) = h_{11}x_i + h_{12}y_i + h_{13} \\ y'_i \left(h_{31}x_i + h_{32}y_i + h_{33}\right) = h_{21}x_i + h_{22}y_i + h_{23} \end{cases} \tag{3.18}$$

In a matrix form, rearranging the terms to explicit terms of the homography matrix, Equation (3.18) can be written as:

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i \end{bmatrix} \begin{bmatrix} h_{11} \\ \vdots \\ h_{33} \end{bmatrix} = \begin{bmatrix} x'_i \\ y'_i \end{bmatrix} \tag{3.19}$$

Considering now a general case with n correspondences the problem to solve becomes in the form:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{11} \\ \vdots \\ h_{33} \end{bmatrix} = \mathbf{0} \tag{3.20}$$

Equation (3.20) is in the form of $\boldsymbol{Ah} = \mathbf{0}$, where $\boldsymbol{A} \in \mathbb{R}^{3x3}$ and $\boldsymbol{h} \in \mathbb{R}^9$. When the number of correspondences is greater than 4, the system is overdetermined, and the homography matrix can be estimated by solving the least squares problem:

$$\boldsymbol{h}^* = \arg\min_{\boldsymbol{h}} \|\boldsymbol{Ah}\| \quad \text{such that } \|\boldsymbol{h}\| = 1 \tag{3.21}$$

Chapter 4

Methods

4.1 Traditional Computer Vision-based Approach

This section is related to all the experiments done on the traditional computer vision-based approach. The general scheme of the driver's attention model is shown in Figure 4.1. The model is divided into four main stages: data synchronization, homography projection of the gaze, scene perception, and the driver's behavioral model.

The data synchronization stage is responsible for aligning the data from the different sensors. In particular it is important that gaze data and images from the eye-tracking glasses are well synchronized each other and with video frames from the camera installed on the roof top of the car.

The homography projection of the gaze stage is responsible for projecting the gaze of the driver from the ETG camera plane to the roof top camera plane. In this way we have a wider, more stable and accurate representation of the outside environment. It is possible to estimate the homography transformation making the approximation that the matched keypoints are far away enough from the vehicle. This is a reasonable approximation since the driver is usually looking at the road and the objects far away from the vehicle. Moreover, the baseline of the stereo vision system is small compared to the depth of the keypoints. The optimal way to estimate the projection would be through the epipolar geometry, estimating the fundamental matrix and the essential matrix. However, we have an uncalibrated stereo setup, and the baseline between the two cameras is not fixed. Even though it could be possible to integrate GPS data to estimate the two matrices, there is a consistent noise error that affects accuracy of the estimation. The homography estimation is then divided in three steps: detection of keypoints in the two images through the SIFT algorithm, matching of the keypoints through RANSAC, and estimation of the homography matrix through the least squares method.

The scene perception stage is responsible for detecting and tracking the vulnerable users in the scene, such as pedestrians and cyclists. The detection is done through the YOLOv8 algorithm [40] and the tracking through ByteTrack [61]. In this way it is possible to compare the gaze of the driver with the state of the targets, including their position.

Finally, on the top, there is the driver's attention model. The responsibility of this block is to classify dangerous scenarios given the data from the driver and the scene. However, this block is sensitive to the quality of the signals of the previous stages.

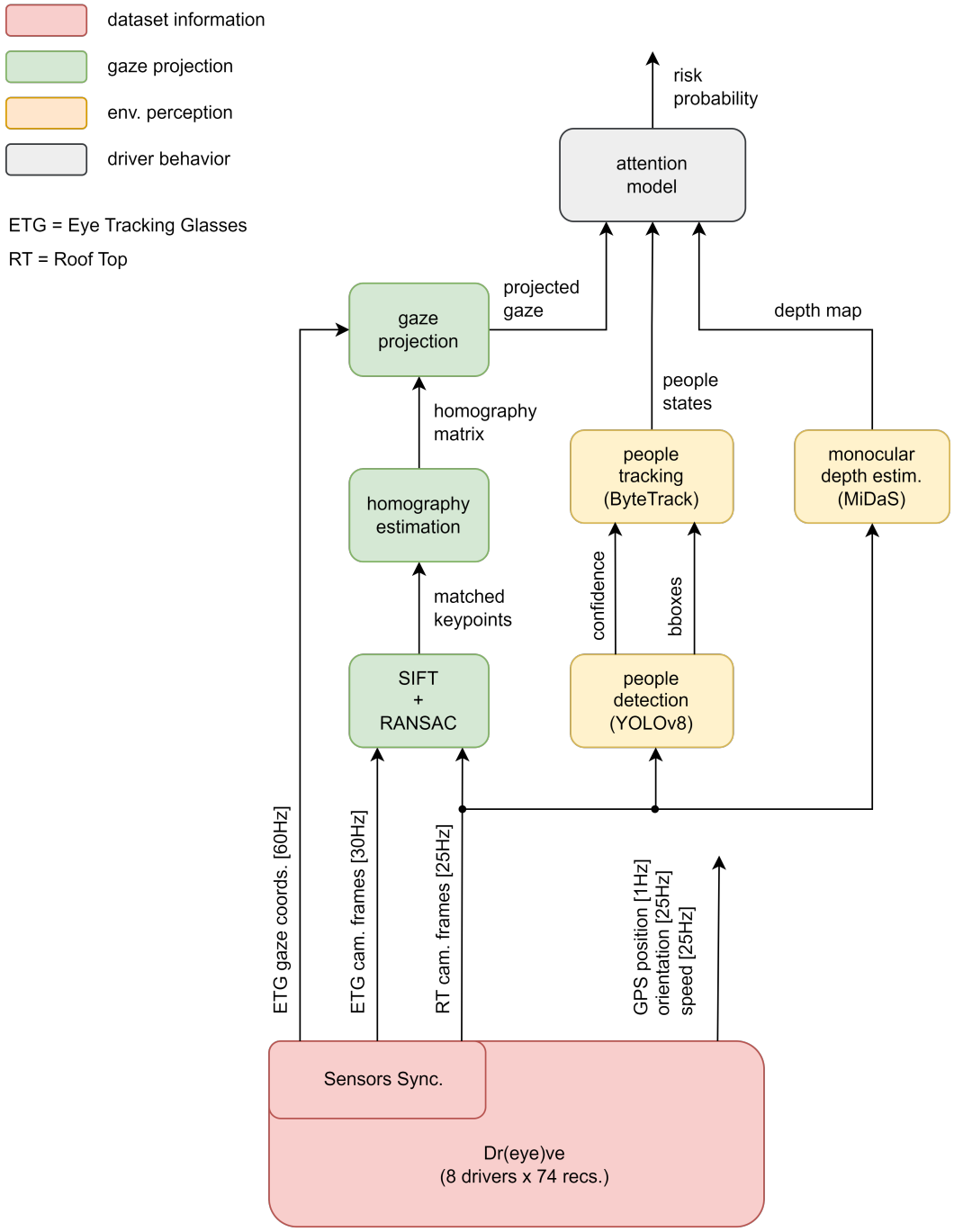


Figure 4.1: The overall driver's attention scheme. It is divided into four main stages: data synchronization, homography projection of the gaze, scene perception and the driver's behavioral model.

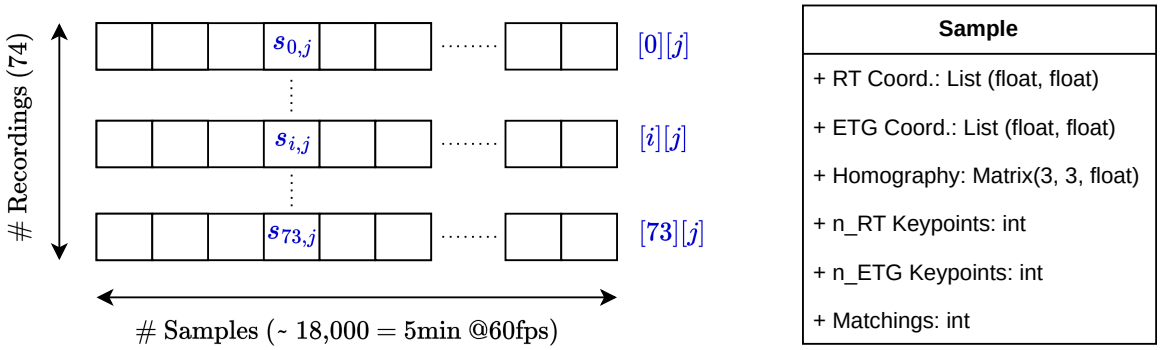


Figure 4.2: The data structure used to store the homographies. **Left**: Each element is a specific sample in the i-th video, at the j-th frame. **Right**: The attributes of each sample.

4.1.1 Homography Data Structure

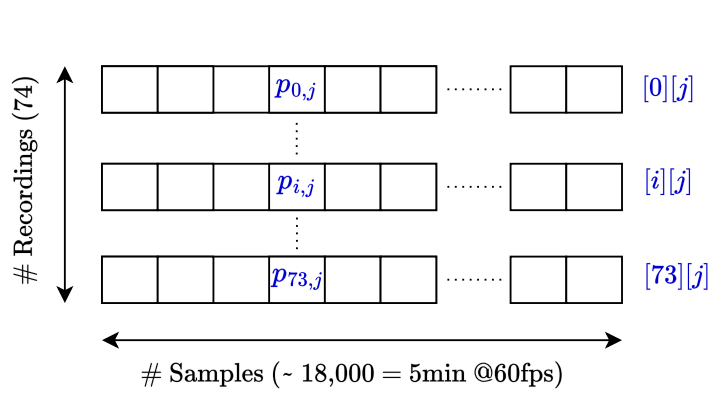
The DR(eye)VE dataset is composed of 74 sequences of five minutes each. Moreover, the two cameras are not synchronized. The ETG camera has a frame rate of 30 fps, while the roof top camera has a frame rate of 25 fps. It is also important to notice that there are some videos that was recorded at slightly different frame rates. All the synchronization data are provided in the dataset.

Therefore, we decided to implement an interface to preprocess all the frames and synchronize them. After the synchronization, we compute all the homographies and store them in a file. This file is then used to project the gaze of the driver in the roof top camera plane. Furthermore, we chose to store other informations related to the quality of the estimation. The data structure is described in Figure 4.2: it is a list of lists where each element is a unique sample that has the following fields: the gaze coordinates in the ETG and RT camera planes, the homography matrix, detected keypoints on the two planes, and the number of matchings between them.

4.1.2 Stereo Camera System Setup

The cameras recording system can be reconducted to an uncalibrated stereo vision setup. In particular, the two cameras are not aligned and the baseline is not fixed. However, this is a problem for the projection of 3D world points between the two cameras. In fact, homography is a planar transformation, and it is possible to estimate the projection matrix if keypoints lie on the same plane. For this reason we decided to make the approximation that the keypoints are far away enough from the vehicle. This is a reasonable approximation since the driver is usually looking at the road and the objects far away from the vehicle. Moreover, the baseline of the stereo vision system is small compared to the average depth of the keypoints.

In this way, it is not necessary to know intrinsic and extrinsic parameters of the cameras, such as focal length, resolution, and distortion coefficients. This approach is also proposed in the DR(eye)VE paper [36].



Person
+ ID: int
+ Detected: List(float)
+ Observed: List(float)
+ Boxes: List(BoxCoord)
+ Score: List(float)
+ RT Gaze: List(2DCoord)
- BoxCoord: List(4, float)
- 2DCoord: List(2, float)

Figure 4.3: The data structure used to store targets' states. **Left**: Each element is a specific person, which contains the set of states during all its tracking. **Right**: The attributes of the tracked person during time.

4.1.3 Targets Data Structure

From the DR(eye)VE dataset, we extracted the bounding boxes of the vulnerable road users, such as pedestrians and cyclists. Moreover, through ByteTrack [61], we tracked the targets to have a spatio-temporal representation of the scene. Then, we compare the projected gaze of the driver with the location of targets.

However, it is necessary to consider the quality of the tracking. Even though ByteTrack was specifically designed to track also overlapping objects, in driving scenarios it is not uncommon to have tracking failures. To partially mitigate the problem, we set two different states for each target: a *detection* state and an *observation* state.

- **Detection**: the target is being detected and tracked by the algorithms.
- **Observation**: the target is being observed by the driver.

Through this approach, we can make sure if a person is both detected by the algorithm and observed by the driver. Without storing the detection data, there can be some unclear situations where the driver is looking at a target that is partially or completely occluded. That can happen because ByteTrack keeps the tracking of occluded targets through a Kalman filter. The data structure is represented in details in Figure 4.3: it is a list of lists where each element is a unique person that has been tracked in the respective video.

Each person has the following list of states: a unique tracking ID, the detection states, the observation states, the bounding boxes' coordinates with the confidence score, and the coordinates of the projected driver's gaze.

4.1.4 Adding Depth Information

Targets' depth information can be very informative for the driver's attention model. In fact, the driver is usually more interested in the objects that are closer to the vehicle. Considering the cameras' setup there are three main ways to include depth information of the scene: using a neural network to estimate the depth of the detected objects from the bounding box dimensions, leveraging the stereo vision data and car's

location information (GPS and speed) to estimate the depth of similar keypoints in consecutive timeframes, or to compute a monocular depth map through a pretrained model.

The first approach is the simplest and the fastest, but it is also the less accurate. In fact the depth of an object is not only related to its dimensions, especially for people. Bounding boxes can vary depending on the target age, pose, occlusions and if they are riding a vehicle, like a bicycle. Moreover, a custom calibration should be made at least for each camera model, with its dedicated lens and image sensor. Image distortion could heavily affect the quality of predictions. However, this is not feasible in our case because we do not have any ground truth information to fine-tune the model. Some recent works on targets' depth estimation with stereo vision systems were proposed in [27], [37]. There are also studies on retrieving the depth of objects from monocular images [59].

The second approach is the most accurate and robust, but it requires an accurate calibration of the stereo vision system to perform well. Moreover, data depth is only related to correspondent keypoints. This means that if no matching keypoints related to a target are found, it is not possible to estimate its depth. Finally, the stereo vision system is not always reliable, especially when it is uncalibrated and the baseline is not fixed and relatively small compared to the average depth of the keypoints. Recently some self-calibration methods have been proposed [49], [17]. The roof top camera has a wide field of view because it uses a wide angle lens, and then an anti-distortion algorithm is computed. This affects the quality of the depth estimation.

The third approach is the most versatile and suitable to the specific application. In fact, the dense depth map computed by the pretrained model is not related to the ETG camera and allows to estimate the depth of all the objects in the scene. The model is trained on a large dataset and is able to generalize well to different scenarios. The only drawback is that the depth map is not absolute. This means that it is not possible to estimate the distance of objects in meters, but through a relative scale depending on the maximum and minimum depths of the scene. However, this should not be able to compromise the quality of the driver's attention model because that is the same approach we use as humans when driving. However, we actually compute a hybrid approach where we use a sort of stereo vision system through our eyes and our knowledge and past experience to make an approximate estimation of the depth of the objects. In the experiments section we will show the performance of pretrained MiDaS [39].

4.1.5 Adding Spatial Information

Spatial information can be fundamental to analyze the driver's attention towards targets in the scene. In fact, the driver usually pays more attention to some locations of the field of view, depending on the context (e.g. type of road, traffic, weather conditions, crowdness, etc.). Moreover, the double camera setup allows to have a wider field of view on the rooftop camera. This is particularly useful to track the gaze also when the driver is moving the head.

Therefore, an initial approach is to include the spatial information of the gaze in the driver's attention model. In particular, we divided the roof top camera view in a 3x7 grid, as shown in Figure 4.5. This division is made to have a more detailed

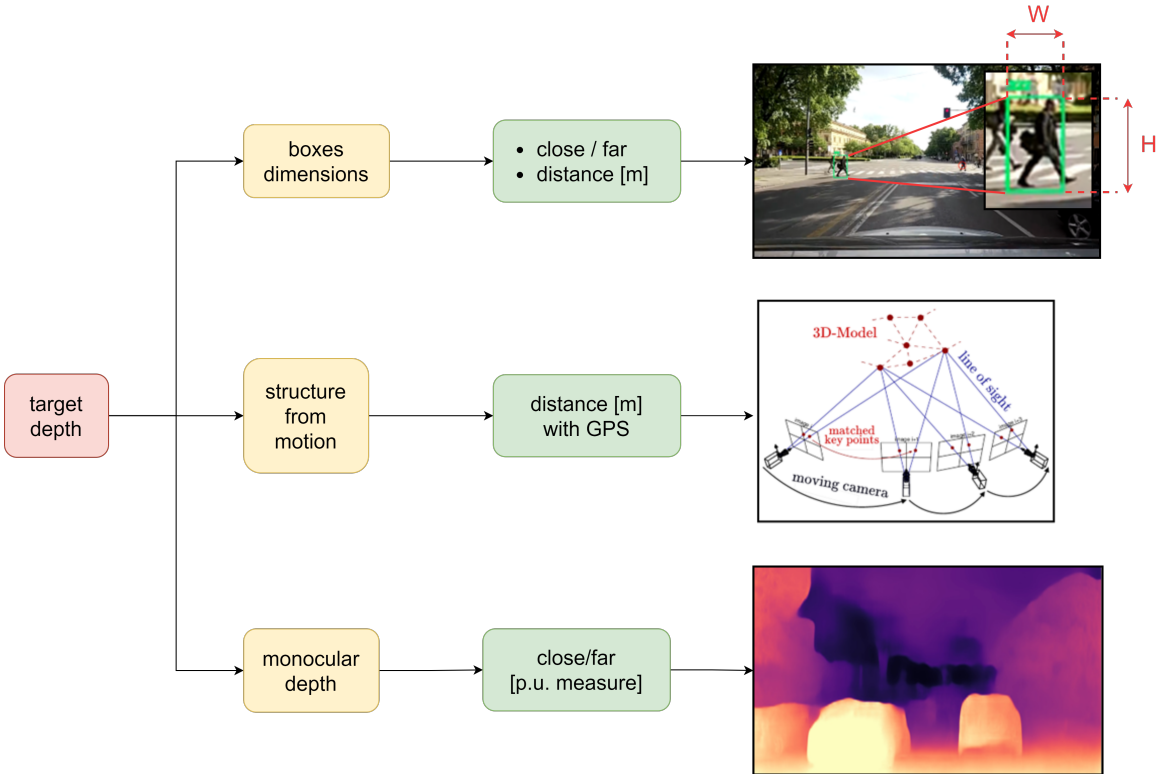


Figure 4.4: Summary of the possible three methods to estimate the depth of targets.

representation of the scene, in fact it is possible to notice that cells from 1 to 7 are out of interest when the driver is looking straight ahead. These area are located on the top of the image, where the driver is usually not looking at, except for some specific situations (e.g. traffic lights, road signs, etc.).

Cells 8, 9, 13, 14 represent out of field of view areas, where there could be potential threats (e.g. a pedestrian crossing the road from the left side, a car crossing the road).

Cells 10, 11, 12 are the center of the field of view, where the driver is usually looking at. In particular, considering the city center shown in Figure 4.5, area 11 captures front vehicles. Cells 10, 12, show possible targets interacting in the near future with the ego-vehicle from the road sides (e.g. the ego-vehicle approaching a crosswalk with a pedestrian waiting to cross the road).

Cells 15, 16, 20, 21 represent the left and sie corners out of the field of view of the ETG camera. These areas are similar to cells 8, 9, 13, 14, but on with closer targets. Therefore they are more dangerous because targets on these locations are more challenging to spot and the driver has less time to react.

Cell 17, 18, 19 are at the bottom-center of the field of view. These areas cover in part the dashboard of the car, and the driver is usually looking at them when checking the speed, the fuel, the GPS, etc. Depending on the context, these areas could also cover close vehicles on the front of the ego-vehicle (e.g. a line of vehicles at a traffic light). Therefore, these areas can identify a warning when the driver is not looking at the road and there is a potential threat.

In general, we identified different areas of interest in the field of view of the driver, depending on the specific context. This is an initial equal division that still helps characterize the scene. However, considering the wide view of the roof top camera,

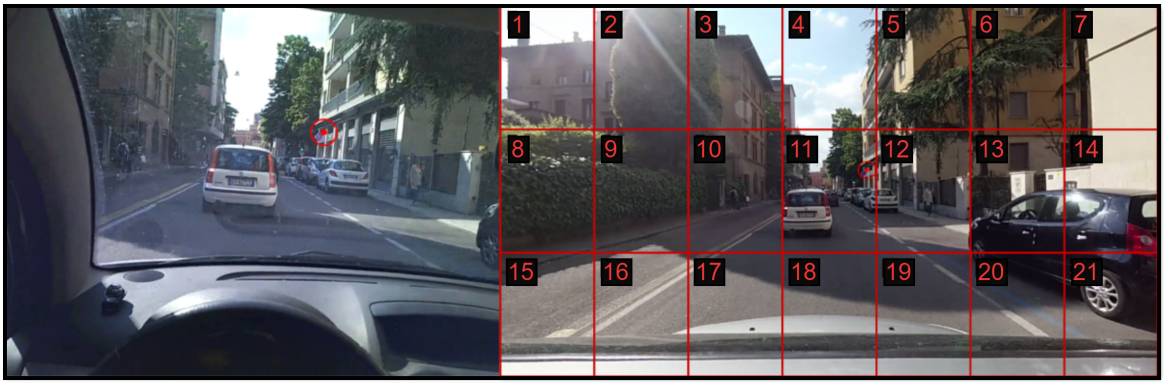


Figure 4.5: Division of the roof top camera view in a 3x7 grid.

and the distortion applied to the image, there are better solutions to divide the scene. For example, it is possible to make a non-equal division, grouping together areas that are more similar to each other. In summary, spatial information is both informative and complicated to manage because it is highly dependent on the context. Therefore, it could be helpful to let a model learn the spatial information by itself, through a deep learning approach. This mainly motivates the move from the traditional computer vision-based approach to the deep learning-based approach.

4.2 Deep Learning-based Approach

In this section we describe the deep learning-based approach to the driver's attention model. Even though in the previous section we identified the main stages of the traditional computer vision-based approach, and many positive aspects were highlighted, there are also some drawbacks. In particular, for each stage there are some simplification hypothesis that affect quality of inputs for the driver's attention model.

For example, the homography projection of the gaze is not perfect, and the approximation that the matched keypoints are far away enough from the vehicle is not always true. Moreover, the quality of homography estimation heavily depends on the quality of the keypoints' detection and matching. This can be compromised by the quality of the images, presence of occlusions, light and weather conditions, etc.

The scene perception stage is also affected by the quality of the detection and tracking algorithms. In particular, ByteTrack is not always able to track overlapping objects, and the quality of the tracking is affected by the quality of the detection. Moreover, the detection is not always perfect, and there are some false positives and false negatives.

Depth estimation stage is also affected by the quality of the pretrained model of MiDaS. In particular, the model is trained on a large datasets in completely different contexts, and it is not always able to generalize well to different scenarios. Moreover, the depth map is not absolute, and it is not possible to estimate the absoulte distance in meters.

That is why it is better to let a model learn important features automatically, through some classification biases used to label a dataset. We decided to use a vision transformer model to learn the self-attention map of the scene to detect potential threats. Four main experiments were made: supervised and semi-supervised training

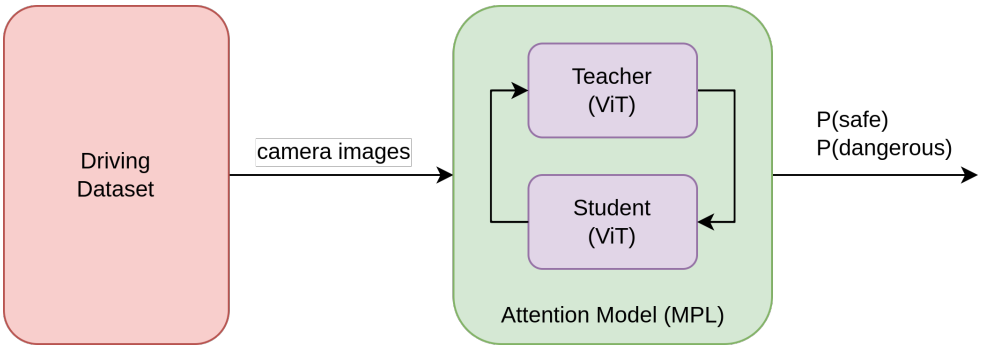


Figure 4.6: General scheme of the deep learning-based approach for detecting dangerous situations in driving scenarios.

on DR(eye)VE, supervised and semi-supervised training on BDD-100K. The general scheme is shown in Figure 4.6; as it is possible to notice, the scheme does not consists of some blocks to extract indirect features from the scene, but it is mainly focused on the deep learning model to learn the features by itself by camera images. In particular, the attention model is represented by two vision transformers: one is the teacher and the other is the student. The teacher is in charge of generating pseudo labels for the student from the unlabelled dataset. The trained student is then used to make the final predictions, and it is supposed to perform better than the teacher.

4.2.1 Attention Map for Anomaly Detection

The vision transformer model is able to learn an attention map of the scene for each head. This feature is useful to let the model learn many spatial relations between patches. The idea is to have many images labeled by humans to detect dangerous situations in a driving dataset. Then, we can use the attention map to spot critical areas of the scene that are more likely to be related to dangerous situations. It is also possible to check if attention maps are consistent with the human decision bias with a test set.

An example of attention maps for a pre-trained multi-head vision transformer is shown in Figure 4.7. It is possible to notice that each head is focusing on some areas of the scene, learning different relations. This is particularly useful to detect anomalies in the scene, such as a pedestrian crossing the road, a car approaching the ego-vehicle, etc. In this particular case, the model is pre-trained on ImageNet, therefore many highlighted areas correspond to the presence of people and cars. However, in the fifht head the model extracts more general features to combine with the desired targets.

Artificial Bias

Labelling an entire dataset with dangerous situations is a very challenging task. On one hand, it is not always possible to have a clear definition of what is a dangerous situation. In fact, it is highly dependent on the context, the person who is labelling and the driving experience. On the other hand, it is very time consuming and expensive to label a large dataset.

Therefore, we decided to start using an *artificial bias* to classify dangerous situations. In particular, we used two different criterias to label DR(eye)VE and BDD-100K datasets. In DR(eye)VE, we used the following criteria: a driving scene is considered dangerous if there is at least one vulnerable road user (e.g. pedestrian, cyclist) or one car; the scene is condidered safe otherwise.

In BDD-100K, on the other hand, we used the following criteria: a driving scene is considered dangerous if there is at least one vulnerable user (e.g. pedestrian, cyclist) or a bicycle; the scene is considered safe otherwise.

The main motivation for the change from DR(eye)VE to BDD-100K is that the former dataset does not have any human annotations regarding location of targets. Therefore an object detector was used to accomplish the task. However, the quality of the detection is not always perfect, and there are some false positives and false negatives. The latter, on the other hand, has high-quality human annotations for many classes. Moreover, it is the largest dataset available for driving scenarios, because each labelled frame corresponds to the frame at the tenth second of a driving video. This is particularly useful to leverage semi-supervised learning techniques, like Meta Pseudo Labels [38].

Finally, we used two different labelling rules and two different datasets because DR(eye)VE has many more frames with no other vehicles in the scene (recordings during night, in the countryside, etc.), while BDD-100K has a high percentage of frames containing other vehicles. Therefore, the second labelling method is to have a less skewed dataset. However, the two methods should not affect the model training, since we are grouping frames according to the presence of some objects, that the model should find out from the training set.

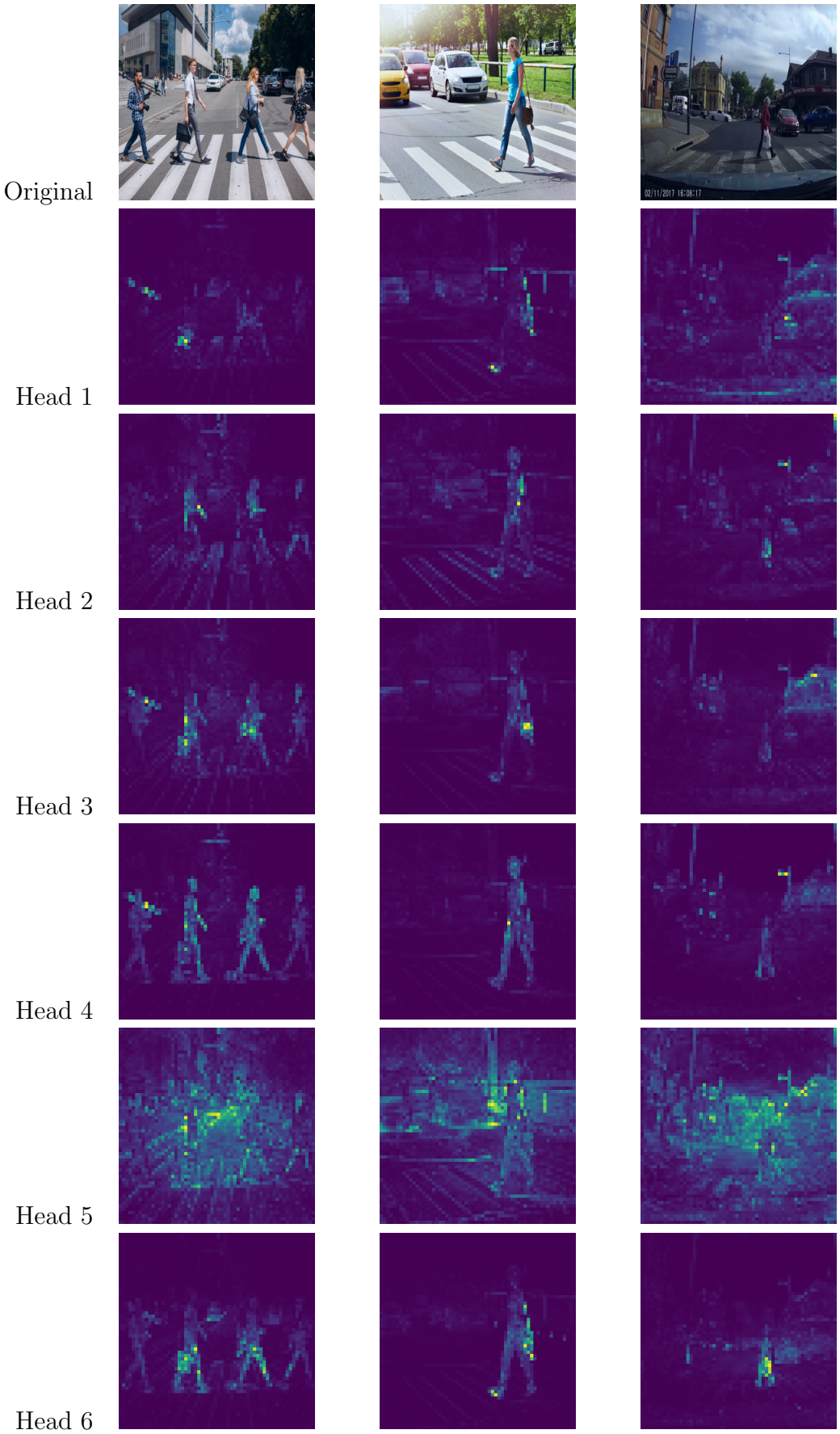
4.2.2 Data Preprocessing of DR(eye)VE

The DR(eye)VE dataset is composed of 74 video sequences of five minutes each. To reduce redundancy of images it was downsampled at 4 fps and resized and stretched to 300x300 pixels. Then RetinaNet [29] was used to detect the desired targets, and the label was assigned to the frame if at least one of the targets was detected.

Then we divided the dataset in training, validation and test sets. In particular, the validation set is based on 1000 images, equally balanced between dangerous and safe situations. The test set is based on 200 images, also equally balanced between dangerous and safe cases. Both the validation and test sets are always kept fixed, with the same ratio.

On the other hand, the training set has different sizes, because we made some experiments in both supervised and semi-supervised learning, changing the labelled training size to figure out how it affected the predictions. Therefore, we started with 500 labels, then 1000, 2500, 5000 and 10000. The initial training set is not equally distributed between dangerous and safe cases (around 63% of safe cases and 37% of dangerous cases). A further training set with 40000 labels was also made to better understand the model's behavior with a large amount of labelled data.

Figure 4.7: Attention masks for a pre-trained multi-head ViT [4].



4.2.3 Training Pipeline on DR(eye)VE

Starting from the smallest training set, when increasing labels after each experiment, we still use RetinaNet to classify new images. However, the criteria to augment the training set is based on the model's predictions. In particular, we used the object detector as a *validator* to correct wrong predictions. In summary, the training pipeline can be described in the following way: In particular, when increasing the training set,

Algorithm 1: Iterative training on the DR(eye)VE dataset

```
Initialize training, validation and test sets (Algorithm 2)
while True do
    Train model in supervised or semi-supervised mode (Algorithm 3)
    Update training set with N wrong predictions (Algorithm 4)
end
```

we compensate the unbalance between dangerous and safe cases by picking the same number of wrong predictions between the two classes. This is particularly useful to avoid the model to overfit on the majority class. However, we also weight the loss function to give more importance to the minority class.

The same pipeline is used for training models with Meta Pseudo Labels, but in this case the unlabeled set is both used to train the model and to pick new samples to label. Moreover, we stopped the training when reaching the maximum validation accuracy of the student.

4.2.4 Data Preprocessing of BDD-100K

The BDD-100K dataset is a large and diverse driving video database designed for the development and evaluation of automated driving technologies. It includes 100,000 videos captured from various geographic, environmental, and weather conditions. Each video in the BDD-100K dataset is typically about 40 seconds long, recorded at 30 frames per second, resulting in approximately 1,200 frames per video. However, only 100,000 frames are completely labelled by humans, and each frame corresponds to the tenth second of a video.

Therefore, we used all the labelled frames and downsampled the unlabeled set (the videos) at 3fps. This is to reduce redundancy of data in semi-supervised learning and speeding-up the training process. We also resized and stretched all frames to 600x600 pixels. We chose this resolution because it is a good trade-off between details of the scene and computational cost.

In this case it not required to use any validator because we only used human-labelled frames for the labelled training set. In particular, we labelled a frame as dangerous if there is at least one vulnerable target, including pedestrians, riders, bicycles. The scene is considered safe otherwise. However, considering the trade-off between resolution and details described above, we choose to set a minimum threshold of the area occupied by the bounding box of the target to consider the scene as dangerous. This is particularly useful to have more reliable data, to make sure that the model is able to extract enough useful feature from. The minimum threshold area is set to 5000 pixels for each target in the original image resolution of 1280x720 pixels.

4.2.5 Handling Unbalanced Data

Managing unbalanced datasets in binary classification poses significant challenges due to the disproportionate distribution of the classes. In such scenarios, models trained on these datasets might exhibit a bias toward the majority class, often at the expense of the minority class. This can lead to a situation where the model performs well statistically in terms of overall accuracy but fails to correctly identify instances of the less frequent class.

The complexity arises because the usual evaluation metric, accuracy, does not reflect the model's performance on the minority class effectively. This misrepresentation can lead to misleading conclusions about the model's true effectiveness, especially in the BDD-100K dataset, where there is a proportion of 90% of safe cases and 10% of dangerous cases.

To address these challenges, it's crucial to employ a suite of metrics that provide a more comprehensive view of the model's performance across both classes. Metrics such as precision, recall, F1-score, and others allow for a more detailed assessment of how well the model identifies and classifies instances from both the majority and minority classes. Each metric highlights different aspects of the model's behavior, such as its ability to correctly predict positive cases, avoid false positives, or balance these factors through a combined score. By considering these metrics, we can better understand and mitigate the biases inherent in models trained on unbalanced datasets.

Evaluation Metrics

In the context of unbalanced datasets like DR(eye)VE and BDD-100K, traditional accuracy, which simply measures the proportion of total correct predictions relative to the total dataset size, can be misleading. For instance, in BDD-100K dataset, where 90% of the data are of one class, a naive model predicting only that class would achieve 90% accuracy, despite not having learned to identify the rarer class effectively.

Instead, more nuanced metrics such as precision, recall, and F1-score are used. Precision is the ratio of correctly predicted positive observations to the total predicted positives. It is defined as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

where TP is the number of true positives and FP is the number of false positives. This metric helps to understand the accuracy of the positive predictions.

Recall (or sensitivity) measures the ability of a model to find all the relevant cases within a dataset. It is calculated by:

$$\text{Recall} = \text{Sensitivity} = \frac{TP}{TP + FN}$$

where FN is the number of false negatives. This metric is crucial for cases where missing a positive instance is significantly worse than falsely labeling negative instances as positive.

The F1-score is the harmonic mean of precision and recall, and is calculated by:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

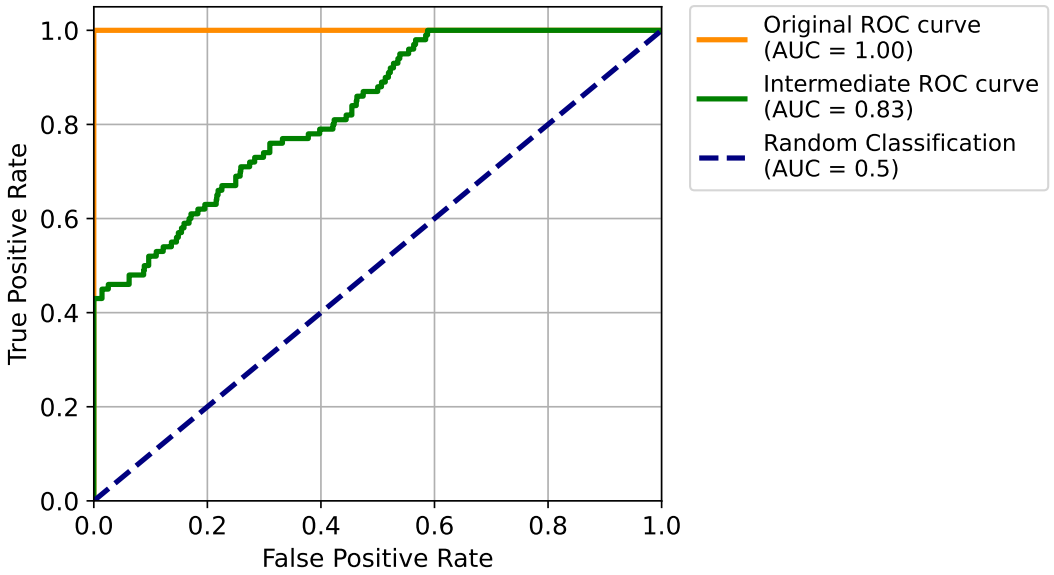


Figure 4.8: Example of ROC curve for a binary classification model.

This score is particularly useful for unbalanced datasets because it takes both false positives and false negatives into account, providing a more realistic measure of a model's performance, especially when the classes are unevenly distributed.

In case of DR(eye)VE this problem does not affect the validation set because it is forced to be balanced. However, the training set is unbalanced, and the model could overfit on the majority class, even though we also weighted the loss function. In this case, we used the F1-score as the main metric to evaluate the model's performance.

In case of BDD-100K, on the other hand, the problem affects both the training and validation sets. In this case, we used the F1-score as the main metric to evaluate the model's performance. However, we also considered recall as an important metric to evaluate the model's ability to find all the relevant cases within the dataset, even though it could lead to more false positives.

ROC Curve

The Receiver Operating Characteristic (ROC) curve is a crucial tool for evaluating binary classification models, particularly in scenarios like detecting dangerous scenes in driving datasets, where the data is highly unbalanced. The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold settings. Sensitivity, as described above is also called recall, and specificity is defined as the true negative rate, or the proportion of negative instances correctly identified as such:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

This curve helps to visualize the trade-off between sensitivity and specificity from the two extreme operation points (with a threshold of 0.0 and 1.0). In fact, it is possible to notice that setting a threshold of 0.0, the model will output all positive cases, then the sensitivity will be 1.0, but the specificity will be 0.0. On the other

hand, setting a threshold of 1.0, the model will output all negative cases, then the specificity will be 1.0, but the sensitivity will be 0.0. An example of ROC curve is shown in Figure 4.8.

Setting the operation point on the ROC curve involves choosing a specific threshold that defines how the binary classifier will categorize positive and negative classes. This threshold setting is crucial in unbalanced datasets because it helps balance the sensitivity and specificity according to the specific needs of the application. For instance, in detecting dangerous driving scenes, missing a dangerous scene (low sensitivity) might be more critical than incorrectly labeling a safe scene as dangerous (high specificity). Therefore, we choose a threshold that prioritizes sensitivity.

The optimal solution often depends on the specific costs associated with false positives and false negatives. These can be explicitly defined through a cost function, which quantifies the impact of these errors. For instance, a cost function could be constructed such that the cost of missing a dangerous scene (false negative) is set higher than incorrectly identifying a scene as dangerous (false positive). By minimizing this cost function across possible thresholds, we can determine the most cost-effective operation point on the ROC curve.

Matthews Correlation Coefficient (MCC)

The Matthews Correlation Coefficient (MCC) is a metric used to evaluate the quality of binary classifications. It is especially useful in scenarios where the dataset is unbalanced. The MCC takes into account true and false positives and negatives and is generally regarded as a balanced measure that can be used even if the classes are of very different sizes. The coefficient returns a value between -1 and +1. An MCC of +1 indicates a perfect prediction, 0 indicates a random prediction, and -1 indicates total disagreement between prediction and observation. The formula for MCC is:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

When any of the sums in the denominator is zero, the denominator can be arbitrarily set to 1, leading to a MCC of zero if both classes are absent from the dataset.

In unbalanced datasets, where the number of instances in each class is not equal, common metrics like accuracy can be misleading. For example, in a dataset where 95% of instances belong to one class, a model that always predicts the majority class will have high accuracy but will perform poorly on the minority class. The F1 score, which considers both precision and recall, improves on this by focusing on the performance related to the minority class but still can be skewed depending on the ratio of the classes. MCC, on the other hand, provides a more comprehensive evaluation as it considers all four confusion matrix categories, making it more reliable in assessing the performance of classifiers on unbalanced datasets.

Therefore, when evaluating the performance of supervised and semi-supervised learning on BDD-100K, we combine multiple metrics, including ROC curve, AUC, F1-score and MCC, to provide a comprehensive view of the model's effectiveness in detecting dangerous driving scenarios.

Chapter 5

Experiments

5.1 Traditional Computer Vision-based Approach

5.1.1 Glances to Vulnerable Users

The projection of the gaze from the ETG camera to the RT camera is shown in Figure 5.1. We manually set some gaze points on the ETG camera such that they overlap with the vulnerable users. In this way, we are setting some operation points that we can use to evaluate the quality of the homography estimation. From the figure it is possible to see that the projection is not perfect, but it is a good approximation of the gaze in the RT camera plane. The small errors are due to the fact that the scene is not flat and the homography is a planar transformation. However, most pixels with high contrast correspond to objects that are far away from the vehicle, therefore the approximation is reasonable.

In Figure 5.2 we show keypoints' matchings between the two images. The keypoints are extracted using the SIFT algorithm. In a first instance, matchings are computed according to the Lowe's ratio test, with a threshold of 0.7. This is to make sure that there is enough Euclidean distance from the best matching to the second best matching. Then, we apply the RANSAC algorithm, with a threshold of 5 pixels for the reprojection error. Finally, we compute the homography matrix using the inliers from the RANSAC algorithm. To optimize the estimation problem, we set the maximum number of iterations to 2000 and a confidence of 99.5%.

As it is possible to see from Figure 5.2, the matchings are accurate enough to compute homography. The dark points are all the keypoints extracted with SIFT, and the colored lines are the correspondent matchings obtained through RANSAC. As expected, in a scenario wehre there is a high contrast between the road evironment and the sky, the matchings are related to those loactions. However, many other keypoints where there is high contrast are detected, for example on crosswalks, pedestrians and buildings. They are probably not matched beacuse they are not unique enough to satisfy the Lowe's ratio test.

This is a fundamental aspect to consider when designing a system that should be used in many different scenarios, during the day and night, in different weather conditions, etc. Moreover, in Figure 5.1 and 5.2 we show the results of an especially favorable scenario, where there are good conditions of light and contrast, and the driver is looking straight ahead.

Figure 5.1: Projection of the gaze from the ETG camera to the RT camera. All the gaze points are manually set. **Left**: Roof top camera view. **Right**: ETG camera view.

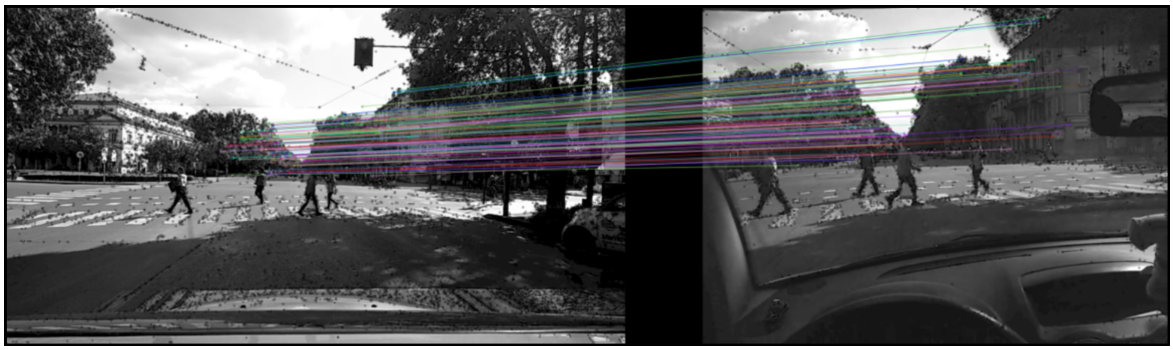


Figure 5.2: Projection of the gaze from the ETG camera to the RT camera. All the gaze points are manually set. **Left**: Roof top camera view. **Right**: ETG camera view.

5.1.2 Data Distribution of DR(eye)VE

After computing all the projected gaze points, we can focus on the interaction of the driver with the vulnerable users. Therefore, it is important to have a general overview of the distribution of people in the DR(eye)VE dataset. However, considering that the focus is on the interaction *during time*, it is also important to embed poeple's tracking information in the analysis.

Therefore, in Figure 5.3 we count the number of different people that are tracked by the driver. In particular, on the lower x-axis there is the total observation time of the person in seconds. On the upper x-axis there is the number of frames where the person is tracked, this is just to have a different representation of the same data that considers preprocessing. On the y-axis there is the number of people observed by the driver for the specific amount of time. The distribution is shown in a logarithmic scale to better compare the values. The plots also compare the downtown with all other scenarios. Green bars represent the downtown scenarios, while red bars represent the sum of all the scenarios in the dataset (therefore including the downtown).

The distribution is right-skewed, with a long tail of people that are observed for a short amount of time. On one hand, this is expected, considering that the driver is usually looking straight ahead when driving. On the other hand, it is important to consider that some missing data could be due to tracking losses or occlusions with ByteTrack, or some non-accurate gaze projections that do not overlap with the bounding boxes of the people.

In Figure 5.4 we show the cumulative distribution of the tracking counts. This is a useful representation to understand the number of people that are observed, and so tracked, for at least a certain amount of time. In this case, the y-axis is linear to better understand the variation of counts with respect to the time.

From the cumulative distribution it is possible to see that the total number of different observed people is around 110. However, considering a minimum observation threshold of 0.3s, which consists of 9 frames, the number of people is reduced to 40. This is a relative small number to consider such a complex interaction betweeen humans, in many different scenarios. Moreover, as described in one of the next sections, ByteTrack suffers from some tracking losses and mismatching, compromising the quality of the data.

It is also important to consider the distribution of recordings with respect to time, weather and areas for each driver. In fact the DR(eye)VE dataset is composed of 8 different drivers, each one with a different driving style and preferences. In Figure 5.5 we show the mentioned plots. Data is enough equally distributed among the drivers, with a slight preference for some of them. This means that the behavioral information embedded in the dataset can generalize well.

5.1.3 Gaze Interaction with Targets

In Figure 5.6 there are six different people that the driver observed for at least 0.5 seconds. This minimum threshold is considered a good compromise between the time needed to understand the context and the quantity of data available, as shown in Figure 5.4. Each plot is referred to a unique person, tracked during the whole sequence. In particular, it is possible to notice that two different functions overlapped for each plot:

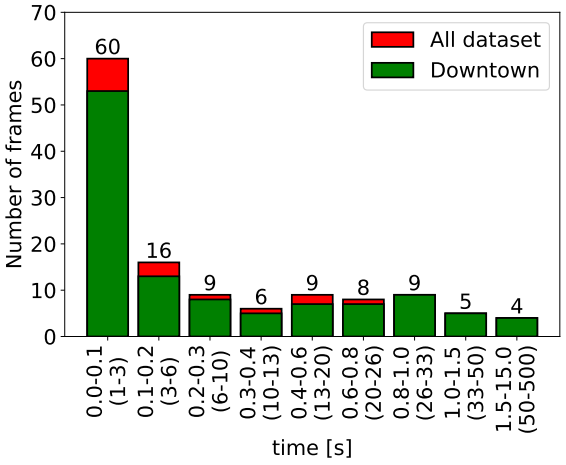


Figure 5.3: Observation time counts.

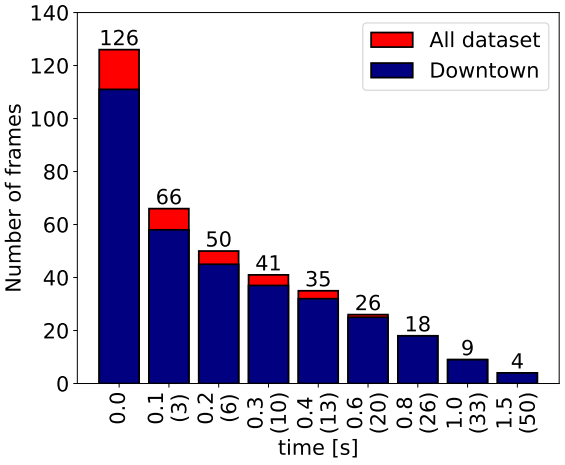


Figure 5.4: Cumulative of the counts.

the observation signal (blue) and the detection signal (red). Both functions can assume two different states: True or False.

The observation signal is set to True when the driver is looking at the person, and False otherwise. The detection signal is set to True when the person is correctly detected and tracked by ByteTrack; otherwise it is set to False. Therefore we can have three different states:

- **D=True, O=True**:
  Driver is looking at a visible and detected person.
- **D=True, O=False**:
  The person is visible and detected, but the driver is not looking at it.
- **D=False, O=False**:
  Tracking of person is lost (possible overlapping, occlusion or missed detection).

It is important to emphasize that it is not possible to also have the fourth state with D=False and O=True, even though the driver could be looking at a person that is not detected by ByteTrack at the moment. Therefore, when there are high variations of both the observation and detection signals at the same states, it is possible that the person is occluded or overlapped, and the driver is looking at it. If the frequency is very high, probably the detection algorithm is missing some detections between time frames (e.g. there is a variation of light conditions, some quick movements, etc.). An example of this case is shown in Figure 5.6.d), between 11s and 13s.

Another case is when the person is continuously detected and the driver is looking at it only for a short period of time. There can be oscillations of the observation during this period, probably due to a small projection error when the case is close to some corners of the bounding boxes. This can be amplified in quick movements' scenarios. An example of this case is shown in all the figures in different periods of time.

If the person is detected continuously and the observation signal varies slowly and periodically, like in Figure 5.6.b-e), it is possible that the driver is looking at different parts of the scene keeping updating the attention towards them.

Finally, in Figure 5.6.c-f) there are some cases where the person is not glanced at by the driver, and the detection signal varies at high frequency. This is the case when the person is visible and tracked, but the driver is not paying attention to it.

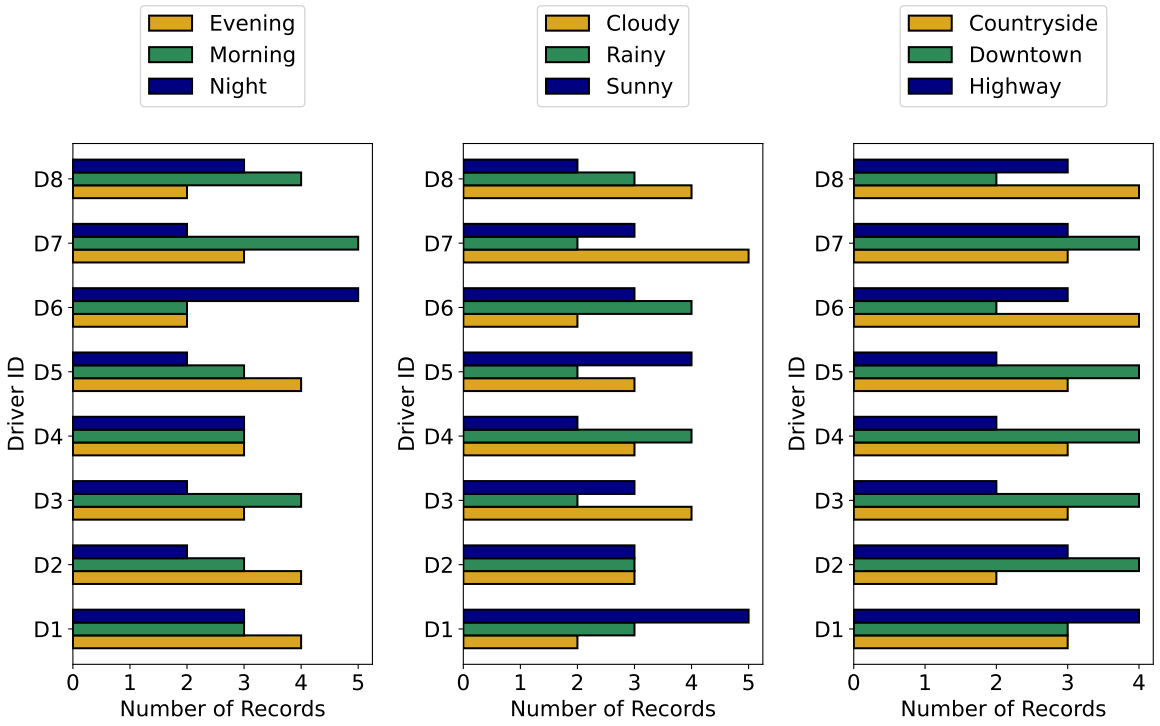


Figure 5.5: Distribution of recordings in DR(eye)VE dataset. **Left**: Distribution of recordings with respect to time. **Center**: Distribution of recordings with respect to weather. **Right**: Distribution of recordings with respect to driving areas.

This is a good example of the importance of contextualizing the scene through perception and driver's data. However, there are many complex cases that are difficult to explain with just these two signals, especially when it is fundamental to consider possible tracking losses or mismatchings.

5.1.4 Tracking Performance

As mentioned before, analyzing driving interatctions between the driver and vulnerable users is a complex task that can be compromised by tracking errors. Even though ByteTrack is a state-of-the-art tracking algorithm, especially designed for scenarios with occlusions and overlapping, it can still suffer from some mismatches and tracking losses. It is complicated to find out when the tracking is lost, from the output of the algorithm. Even worse, if there is a tracking mismatch, driver's gaze information can be compromised, leading to wrong conclusions about the interaction between the driver and the other targets.

In Figure 5.7 we show some examples of tracking mismatches in a driving scenario of DR(eye)VE. In particular, four different frames from the roof top camera are shown. We decided to show the frames where there are overlapping or occluded people, to emphasize the complexity of the tracking task.

From Figure 5.7.a) to Figure 5.7.b) the tracked person (with ID 342) overlaps with another person that is walking on the crosswalks. In this case the tracking is not lost, and the person is correctly tracked.

From Figure 5.7.b) to Figure 5.7.c) the same tracked person overlaps with another

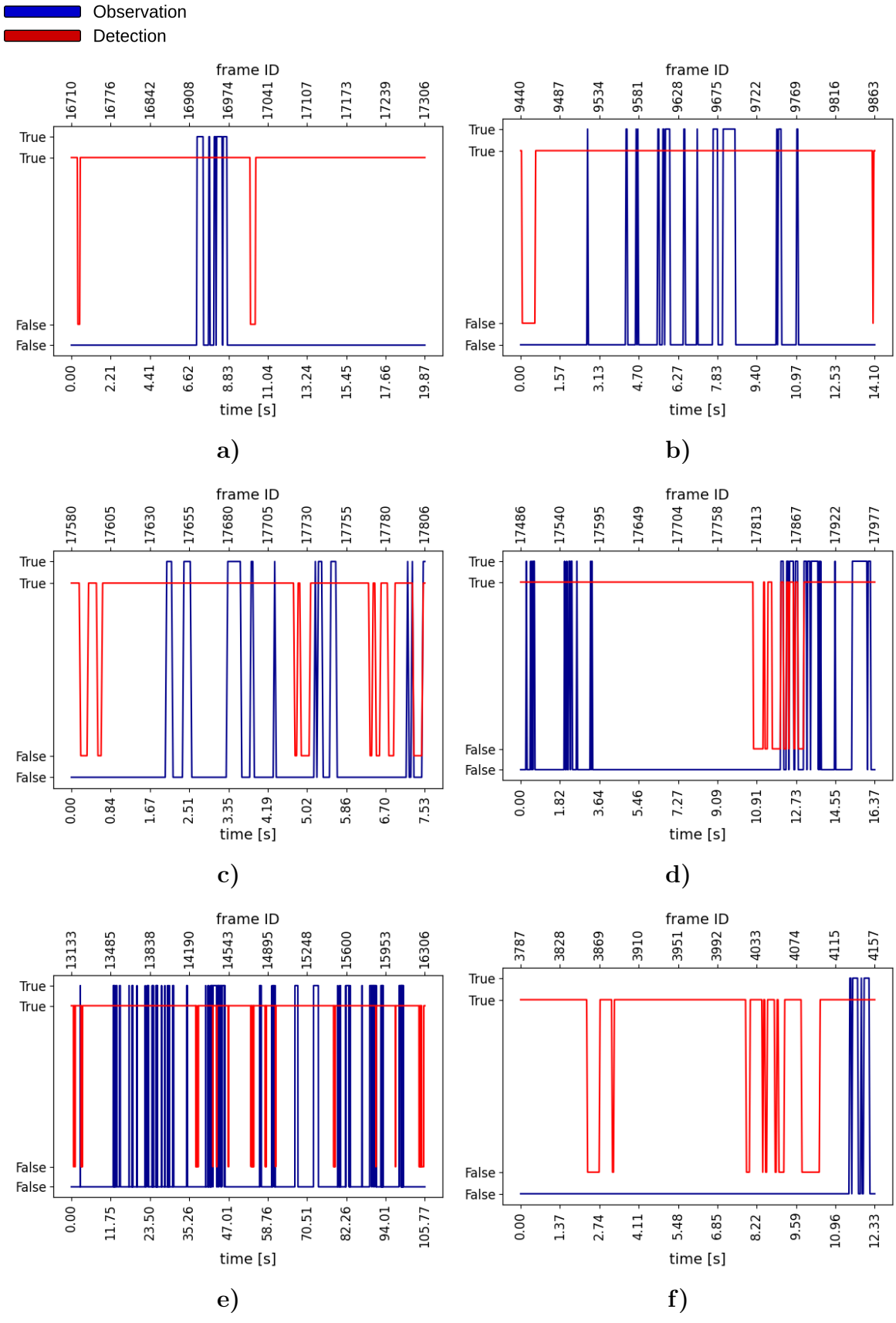


Figure 5.6: Driver glances towards people in the DR(eye)VE dataset (minimum glance threshold is set to 0.5s).

target, but this time there is a tracking mismatch. The bounding box is now assigned to the incoming person, and the tracked person is lost.

A similar case is shown from Figure 5.7.c) to Figure 5.7.d). The tracked person is mismatched with another incoming person.

Even though lighting conditions are optimal, and people are not occluded by other objects, tracking mismatches can still happen. This is a fundamental aspect to consider when designing a feature for an ADAS system. Another aspect is that, despite the presence of other objects in the background like cars, the tracking algorithm is able to correctly keep the right tracking, considering that the backbone model is not specifically trained on detecting people in driving scenarios. This is a good sign of the robustness of the algorithm with respect to the background environment, and could be enhanced through an ad-hoc fine-tuning of the detection stage on driving scenarios.

5.1.5 Monocular Depth Estimation with MiDaS

In this section we show and evaluate the results of the monocular depth estimation computed by MiDaS [39]. To evaluate the quality of the estimation it is necessary to have a ground-truth information to compare with. In this case, DR(eye)VE dataset does not provide any depth information by default, because no depth sensors are used. Therefore, the idea is to validate the model on another dataset that provides depth information and a good variety of scenarios, from downtowns to highways, with different weather conditions, etc. We decided to use the NuScenes dataset [3], that provides both LiDAR and camera data.

Evaluation on NuScenes

In Figure 5.8 we show the results of the monocular depth estimation for an entire scene of the NuScenes dataset. The left image is an heatmap of the relative inverse depth estimated by MiDaS, and compared with the ground-truth pointclouds taken from the front LiDAR. In particular, it is necessary to mention that MiDaS provides a *relative* inverse depth that depends on the specific image, depending on the distance of closer and farther objects. Considering that the comparison is not straightforward for the entire dataset, we decided to show the results for one entire scene, where at least the same environment is present in all the frames. Then the relative inverse depth is normalized to be in the range $[0, 1]$, and compared with the absolute depth of correspondent points in the LiDAR pointcloud. This means that non-correspondent pixels in the dense depth map are masked out. Finally, we decided to use an heatmap to not only show the distribution of points, but also find how frequently the model makes some errors in the estimation. Considering the preprocessing of the data, a reference curve is also shown. It spans from 4m to 60m on the x-axis and from the minimum to the maximum of the relative inverse depth on the y-axis. This is to have a unique function that is able to map all values in the two domains. To be precise, the reference curve should not span only up to the maximum value of the pointcloud, but it should consider the maximum distance in the entire scene. However, this is a good approximation to have a general idea of the quality of the estimation, leveraging ground-truth data. In fact, from the experiments we realized that the model is more accurate in estimating high relative distances between objects in the scene, and this is

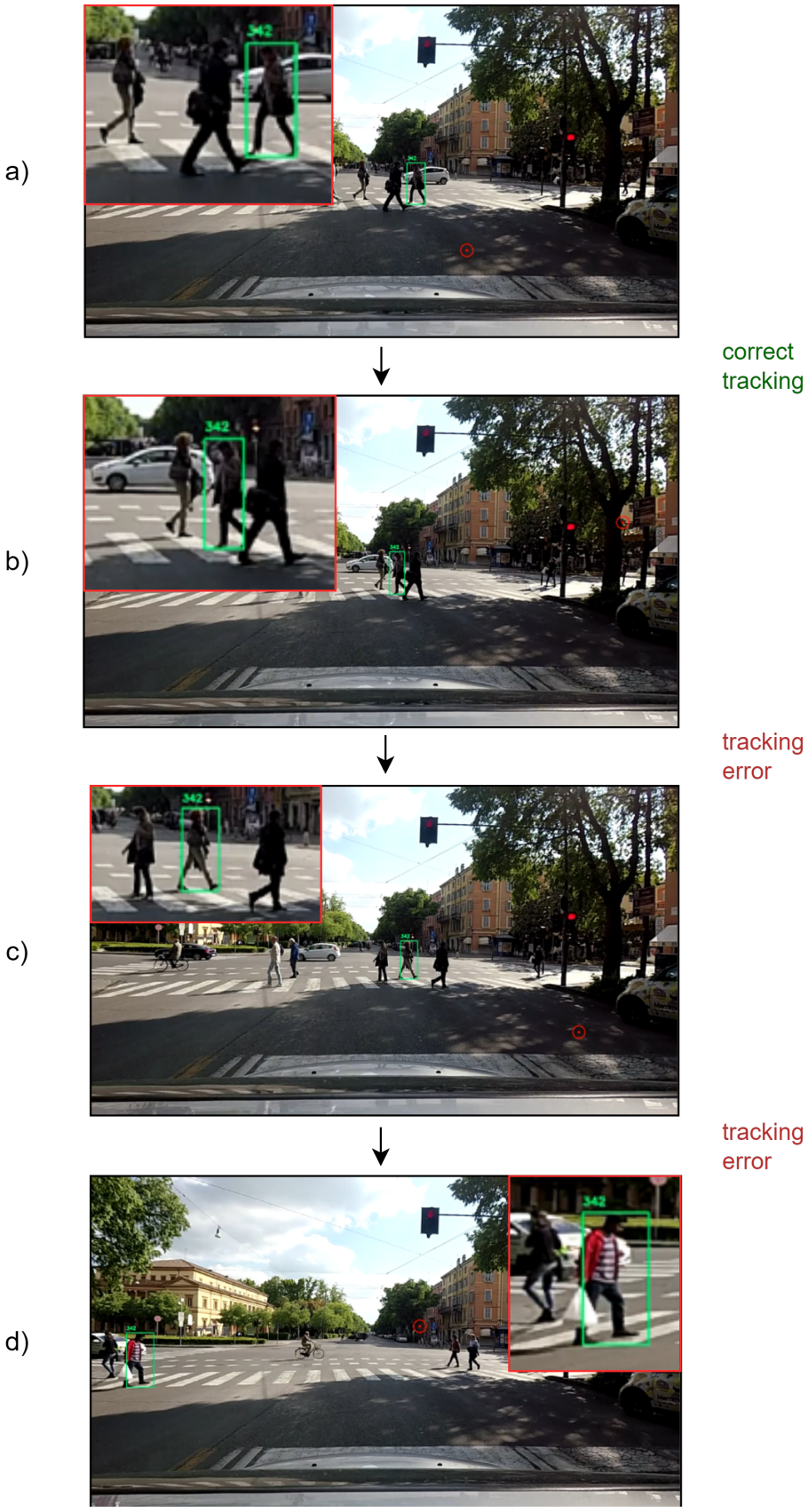


Figure 5.7: ByteTrack's tracking mismatches in a driving scenario.

usually related to closer-to-the-ego-vehicle objects. Moreover, LiDAR pointclouds can reach up to 100m, but the further points are, the more sparcely they are distributed in the 3D space.

Another important aspect to consider is the output of MiDaS, that is the relative inverse depth. This means that the closer the object is, the higher the output of the model is. This is the opposite of the LiDAR pointcloud, where the closer the object is, the lower the depth value is. However, we decided not to invert one of the two signals to have a better comparison at closer distances, but also including farther estimations, with the output tending to zero for the farther objects.

From the results in the heatmap it is possible to notice that the model is able to estimate the depth of closer objects with a good approximation, but it is much less accurate for farther objects. Regarding closer objects, there are many point clouds related to the road that, considering its shapes, it is probably easier to estimate. On the other hand, farther objects are usually less visible in the image, and the model is not able to estimate them correctly. Moreover, there are some close pointclouds that are estimated far from the ego-vehicle, and it adds a consistent error for the evaluation.

Relative Evaluation Metrics

Considering the problems to evaluate the accuracy of the model with respect to the ground-truth data, we decided to report two error metrics: the root mean squared error (RMSE) and the relative root mean squared error (RRMSE). They are calculated as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(d_i - \hat{d}_i\right)^2} \qquad \text{RRMSE} = \sqrt{\frac{1}{n}\frac{\sum_{i=1}^{n}\left(d_i - \hat{d}_i\right)^2}{\sum_{i=1}^{n} d_i^2}}$$

where d_i is the normalized depth of the i-th point in the LiDAR pointcloud, and $\hat{d}_i$ is the relative depth estimated by MiDaS. In total there are n point clouds visible in the scene. As it possible to notice from the definition of the metrics, the RRMSE is a normalized version of the RMSE with respect to the average depth of the scene. This is useful to understand the quality of the estimation in all the dataset. However, we have inverted the relative depth obtained from MiDaS, therefore small values in the dense depth map heavily affect the RRMSE.

The right image, on the other hand, shows the projection of the LiDAR pointcloud on the image plane, through the camera calibration matrix. Each point is colored according to the depth value in the LiDAR pointcloud. This is to have a visual understanding of the distribution of pointclouds in one frame of the scene. As shown in the picture, the scenario corresponds to a downtown environment, with many buildings and cars. Closer point clouds can be related to the road, parked vehicles on the sides, road obstacles, etc. Farther point clouds are usually related to buildings, trees, and other objects that are not directly on the road.

After having introduced the relative metric, the RRMSE, we evaluated the model on other five scenes of the dataset. Results for camparing the metrics are reported in Table 5.1.

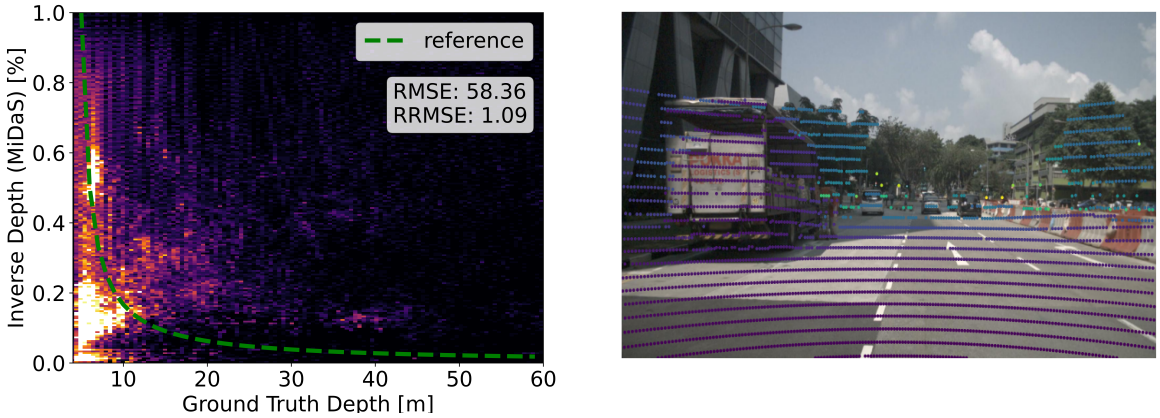


Figure 5.8: Monocular depth estimation with MiDaS on NuScenes dataset [3]. **Left**: Relative inverse depth estimated from the model compared to the ground-truth information taken from the front LiDAR for an entire scene. **Right**: Projection of the LiDAR point cloud on the image plane for one sample of the same scene.

Scene	RMSE	RRMSE
Scene 1	58.36	1.09
Scene 2	29.09	0.52
Scene 3	26.50	0.53
Scene 4	21.96	0.40
Scene 5	24.54	0.99

Table 5.1: Evaluation of MiDaS on five scenes.

5.2 Deep Learning-based Approach

5.2.1 Training on DR(eye)VE

To compare supervised and semi-supervised training of ViTs on DR(eye)VE, we trained different models with different number of labels. In particular, we used the method with the validator explained in the methods section. The reason behind using two different vision transformer models is to also understand if and when more parameters are necessary to better generalize the validation set.

In Table 5.2 we show the results obtained with the training method for both supervised and semi-supervised training. Relevant information to track is reported in the columns, and they are:

- **Mode**: the training method used, supervised or semi-supervised (MPL).
- **Model**: the vision transformer model used, MViT or LViT.
- **Labels**: the number of labels used in the training.
- **Safe** [%]: the percentage of safe frames in the training set.
- **Dang.** [%]: the percentage of dangerous frames in the training set.
- **top-acc.**: the best accuracy of the model on the validation set.

- **Biters @top-acc.**: the number of batches seen by the model when it reaches the top accuracy.
- **max. Biters**: the maximum number of batches seen by the model during the entire training.

In Table 5.3 the two models, MViT and LViT, are described in details through their fundamental hyperparameters. In order, they are:

- **Patch size**: the dimension of each patch in pixels.
- **Emb. size**: the size of the embedding layer.
- **Depth**: the number of layers in the model.
- **Heads**: the number of heads in the multi-head attention mechanism.
- **MLP size**: the size of the multi-layer perceptron of the model.

Training Setups

Regarding the trainings, we used a learning rate of 0.001, a seed of 42, and two GPUs' setups: one with two NVIDIA Tesla A100 of 40GB each, and the other with one NVIDIA RTX 3080 of 10GB. The semi-supervised learning was performed in the first setup, in distributed training. The supervised learning, on the other hand, was performed in the second setup.

Considering different models, different setups and different training methods, we introduce a custom metric, *Biter*, that allows to compare the results in all the experiments. The metric defines the number of batches that the model has seen during training, therefore it is calculated as the batch size multiplied by the number of iterations. This is a good metric for comparing supervised and semi-supervised training because it is independent from the number of labels used. This is important, considering that while Meta Pseudo Labels iterates through the labelled dataset it also add new information from the unlabelled dataset. Therefore, the number of epochs has a different meaning in the two training methods.

Training Results

From the results in Table 5.2 it is possible to notice that in supervised learning mode the model asymptotically increases the best accuracy with the number of labels. This is expected, considering that the model is able to generalize better with more information. However, the model does not always increase the accuracy with the number of labels, as shown in the case of MViT-1k and MViT-5k. This is probably due to the fact that adding more labels does not always mean better generalization. In fact, some new labels could just be already seen samples, contributing to unbalancing the dataset with respect to the validation set.

For the supervised training, from 500 to 5,000 labels, Biters to reach the top accuracy increase exponentially, until 100,000 Biters for the last case, due to the increasing size of the labelled set. However, with many more labels, the case with 40,000 labels, just 36,000 Biters are required to reach the optimal training. This is probably due to the redundancy of the labels that comes when substantially enlarging the dataset.

The experiment with 40,000 labels was specifically design to understand the model behavior with a large amount of labels. What we noticed is that the model overfits the training set in all the cases except for the last one. This is why we also decided to train the LViT model with 500 and 10,000 labels. In the former case, it is possible to notice that with more parameters the model reaches better generalization with respect to the MViT model with the same number of labels. In the latter case, the model still increases the best accuracy with respect to both the MViT and LViT models with less labels. This implies that increasing the model is beneficial also with few labels.

Regarding the semi-supervised training with 500 labels, the model is able to outperform all the supervised epxeriments up to 5,000 labels. This is a good result, considering that the smaller model in semi-supervised learning performs similarly to the larger model with 20 times more labels. This means that unlabelled data carries a lot of information that can be used to improve the generalization of the model. However, when training in semi-supervised learning mode, it is complicated to understand when the model is overfitting the labelled set. This is because training is done in two stages for each iteration, through the teacher and the student. Therefore, for a range of Biters, the teacher is learning the training set to better supervise the student. During the second phase, the student starts learning from the "right" pseudo-labels, better performing the teacher on validation data. This process takes many more Biters than the supervised learning, as it is possible to notice in the last column of Table 5.2. However, a validation accuracy of 0.77 is reached with just 180,000 Biters.

Description of Models

The patch size is kept at 6 pixels, considering that the dataset is based on images with a resolution of 300x300 pixels. This means that each image is divided in 50x50 patches; this seems a good compromise to both have a general overview of the scene and to have enough information to understand the details.

On the other hand, embedding size is increased from 64 to 1024, considering the quantity of information to keep to represent the complexity of the dataset. The depth of the model is increased from 6 to 14, and the number of attention heads is doubled. This is to let the model learn more complex and deep features, with many relations between the patches. Finally, the size of the multi-layer perceptron is increased to 2048, to better match extracted features with dangerous scenarios.

Possible Problems

We also made one last experiment in supervised learning, substantially reducing the transformer's size. In this case, the model overfits quickly the training set, in both cases with few and many labels, without reaching the same generalization of the other cases. This strange behavior is probably due to the fact that the model is not able to learn the complexities of the dataset with a smaller number of parameters. The quick overfitting can be justified by easier, but wrong, features extracted by the model.

However, there is a potential problem in the training pipeline; the training and validation sets are not related to completely different scenes. This means that some frames of the same video can be in the training set and in the validation set. Even though the frames are not the same, and videos were downsampled at 4Hz, there still could be some information leakage between the two sets.

These are a fundamental aspects to consider, and that is also why we decided to move to BDD-100K for the next experiments. Moreover, BDD-100K is definetly larger, with much unlabelled data that can be used for semi-supervised training with Meta Pseudo Labels.

5.2.2 Data Distribution of BDD-100K

The BDD-100K dataset contains a large variety of driving scenarios and objects. In Figure 5.9 we show the distribution of classes in the training set. Because manual labels are only available for the frame at the tenth second of each video, statistics regards only a smaller part of all data. However, the distribution is still representative, considering that it covers all the scenes. As it is possible to see from the bar plots, the dataset is unbalanced, with some classes that are more frequent than others. In particular, the most frequent classes are *car*, *traffic sign*, *traffic light* and *pedestrian*. This is expected, considering that the dataset is mainly focused on downtown scenarios, where these objects are usually present. However, the classification bias used for the DR(eye)VE dataset is not valid anymore: using the presence of cars or people in the scene to classify a scene as dangerous would mean that at least 98% of the dataset is dangerous. In fact the number of frames containing at least one car is 68,943; but there are also some frames containing one person (pedestrian, rider, etc.), without cars. The total number of frames in the training set is 70,000.

This observation explains why we changed the bias towards only the presence of people. Considering only pedestrians, riders and bicycles in the scene, we still have an unbalanced dataset, but in the oppsite direction. In fact the number of dangerous frames is 7,411, while the safe frames are 62,589. The dangerous and safe ratios are, respectively 10.5% and 89.5% over the entire training set. This is a common distribution in anomaly detection tasks, and it is actually an intrinsic characteristic of the problem to solve. In fact, a similar distribution of classes is also present in the test set. That is why, instead of augmenting the training set, we change the final classification threshold according to the ROC curve of the trained model.

5.2.3 Training on BDD-100K

In this section we show the results after training two standard vision transformers on the BDD-100K dataset. The models, ViT-B/32 and ViT-L/32, are standard and also used for classification tasks in common benchmark datasets. Hyperparameters' details are reported in Table 5.4. The comparison was made to understand how models with different complexities perform on the BDD-100K dataset in supervised and semi-supervised learning mode. After each epoch, the model is evaluated on the validation set, and the model is saved if the validation loss is the lowest reached until that epoch.

Due to the complexity of the dataset, we reduced the learning rate to 1E−6, with a cosine scheduler that reduces the learning rate to 1E−7 in 100 epochs. We noticed that using a larger learning rate, as in the DR(eye)VE dataset, the model tends to find a local minimum without reducing the training loss to zero.

Table 5.2: Results of supervised and semi-supervised training of ViTs on DR(eye)VE.

Mode	Model	Labels	Safe [%]	Dang.[%]	top-acc.	Biters.@ top-acc.	max. Biters
Supervised	MViT-0.5k	5.0E2	63	37	**0.729**	8.8E3	2.2E5
	MViT-1k	1.0E3	56	44	0.665	2.6E4	2.2E5
	MViT-2.5k	2.5E3	53	47	0.756	3.5E4	2.2E5
	MViT-5k	5.0E3	51	49	0.697	1.0E5	2.2E5
	MViT-40k	4.0E4	83	17	**0.819**	3.6E4	2.0E6
	LViT-0.5k	5.0E2	63	37	**0.756**	6.4E4	8.0E4
	LViT-10k	1.0E3	51	49	0.787	6.2E4	4.4E5
MPL	MViT-0.5k	5.0E2	63	37	**0.781**	1.3E6	3.0E6

Table 5.3: ViTs details for the training on DR(eye)VE.

Hyperpar.	MViT	LViT
Patch size	6	6
Emb. size	64	1024
Depth	6	14
Heads	8	16
MLP size	128	2048

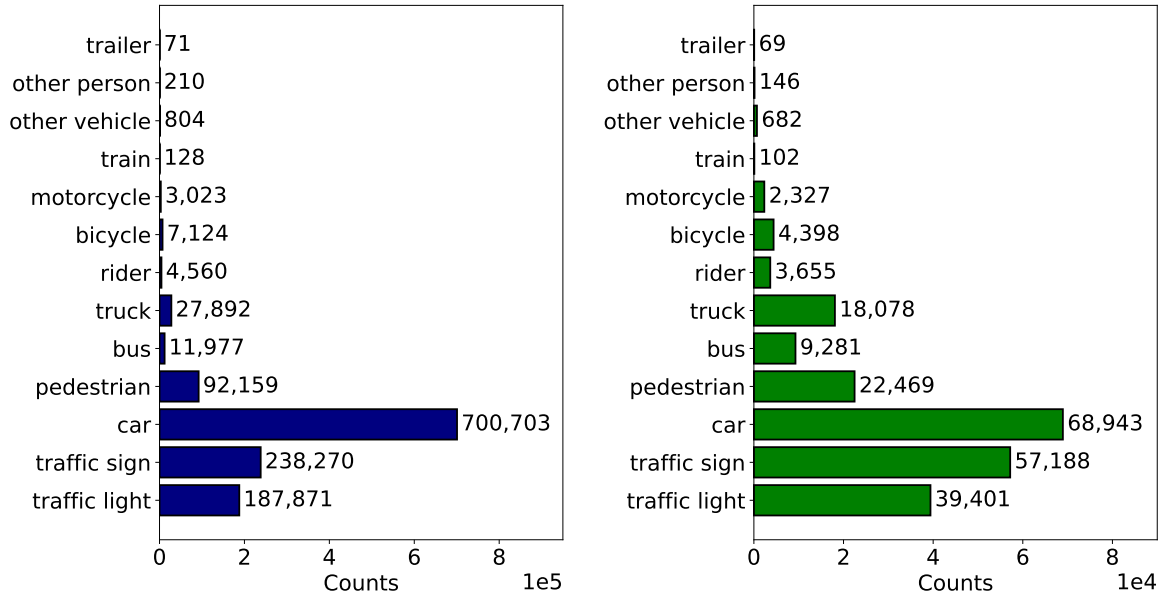


Figure 5.9: Distribution of classes in the training set of the BDD100k dataset. **Left**: Number of instances per class. **Right**: Number of frames containing each class.

Hyperpar.	**ViT-B/32**	**ViT-L/32**
Patch size	32	32
Emb.size	768	1024
Depth	12	24
Heads	12	16
MLP size	3072	4096

Table 5.4: ViTs details for the training on BDD-100K.

Training Setups

The training was performed on two different setups, as for the DR(eye)VE dataset. In the first case, we used two NVIDIA Tesla A100 of 40GB each, and in the second case we used one NVIDIA RTX 3080 of 10GB. The first setup was used for the larger model, ViT-L/32, in semi-supervised and supervised learning. On the other hand, the second setup was used for the smaller model, ViT-B/32, also in semi-supervised and supervised learning mode.

This decision helps to maximize the usage of the resources, considering that a slower CPU with many cores is used for the first setup, and a faster CPU with the second setup. In this way, there is no bottleneck for loading and unloading data from the disk to the GPU, and training is faster.

Training Results

After training the models in supervised and semi-supervised mode, we evaluated quality of predictions through the ROC curve. In Figure 5.10 and Figure 5.11 we show, respectively, the ROC curve of ViT-B/32 and ViT-L/32 on the BDD-100K dataset. For a concise explanation, the semi-supervised training with Meta Pseudo-Labels is referred to as MPL, and the supervised training is denoted as SL. These acronyms are also used to refer to the models trained in the respective modes. Moreover, abbreviations of TPR and FPR are used to refer to the true positive rate and false positive rate, respectively.

In the case of ViT-B/32, in Figure 5.10, the model trained in semi-supervised mode demonstrates comparable performance to supervised learning. However, for FPR values close to 0.2, MPL slightly outperforms SL. Considering that an empirical upper limit for FPR is set to 0.4, to avoid having too many false positives, the working point is constrained to the yellow area in the plot. Therefore, it can be considered one of the possible candidates. For most of the other FPR values, the two models have similar performance. It is also possible to notice how, for same classification threshold, the MPL model always has a lower TPR than the SL model. For example, for a threshold of 0.20, the MPL model has a TPR of 0.58, while the SL model has a TPR of 0.85. This is probably due to the fact that the unlabelled set has a similar distribution of classes to the labelled set, with a higher number of safe frames. Considering that the unlabelled and labelled set are almost the same size, it means that the MPL model learns from more safe frames than dangerous ones, considering the pseudo-labels generated by the teacher model. Therefore, a lower threshold is required to get the same probability of detection of dangerous frames.

In the case of ViT-L/32, in Figure 5.11, the SL model surprisingly underperforms the MPL model for all thresholds. Even though it is complicated to understand the reason behind this behavior, it is possible to notice that the SL model notably adjusts decision thresholds in favor of safe scenarios. In fact, if the SL ViT-B/32 has a TPR of more than 0.9 for a threshold of 0.10, the SL ViT-L/32 has a TPR of 0.7. This could happen because, considering the larger complexity of the model, it is more prone to capture features of the larger class, the safe frames, that do not provide useful information for dangerous predictions.

Comparing the best ROC curves of the two models, which are ViT-B/32 in MPL and SL mode, and ViT-L/32 in MPL mode, we noticed that they have similar per-

formance for all FPR values. However, this means that the ViT-L/32 model benefits much more from MPL, helping mitigating the overfitting on the safe cases. However, it is still difficult to compare the methods because there is not a unique working point in the ROC curve. ROC and AUC are useful to understand the general performance of the model on the positive predictions. Therefore we also decided to analyze the correlation of training and validation sets through the Matthews Correlation Coefficient (MCC) metric and adding a custom cost function for the specific problem to solve.

Correlation Analysis

Through Matthews Correlation Coefficient (MCC) it is possible to understand the correlation between the training set, on which the model is trained, and the validation set. Ranging from -1 to 1, the MCC helps to understand if, and how much, there is a positive or negative correlation between the two sets. In Figure 5.12 we compare the MCC of the two models, ViT-B/32 and ViT-L/32, in supervised and semi-supervised learning mode. The MCC is calculated for different threshold values ranging from 0 to 1. From the plot it is possible to notice how the maximum correlation is reached for a threshold of around 0.18 for the semi-supervised models, 0.48 for ViT-B/32-SL and 0.27 for ViT-L/32-SL.

Considering maximum correlations for the ViT-B/32 model, and comparing them with the ROC curve in Figure 5.10, it is possible to notice that they correspond to the maximum gap between the two curves, for FPR values of around 0.2. On the other hand, for the ViT-L/32 model, it is more complicated to find the correspondence between the two plots, due to the large gap between the two ROC curves for all FPR values. In general, from Figure 5.12 it is possible to notice that the ViT-B/32 in semi-supervised learning mode has the highest correlation.

In Figure 5.13 we show the confusion matrices on the validation set of the models at the maximum MCC value. This means that the working point is set without considering the specific problem to solve. In this case, it is possible to notice that, for ViT-B/32, semi-supervised learning helps to reduce false negatives, while also increasing false positives. In particular, considering the anomaly-detection problem, this model performs best than all the others with 698 true positives. On the other hand, for the ViT-L/32 model, semi-supervised learning helps both to reduce false positives from 1,552 to 1,246 and false negatives from 558 to 537. These results imply that, without considering the specific task, Meta Pseudo-Labels slightly helps to improve the generalization of the models.

Determining the Optimal Working Point

In the previous subsections, we analyzed the models' performance with respect to some points of view agnostic to the specific problem to solve. Therefore, to find the optimal working point, it is necessary to define a trade-off between false positives and false negatives. In this case, we decided to find a unique working point that minimizes a custom cost function defined as in Equation 5.1. In this case, the cost function weights hundred times more false negatives than false positives, and then it is normalized by the total number of frames in the validation set.

$$\mathcal{C} = \frac{FP + 100\,FN}{TP + TN + FP + FN} \tag{5.1}$$

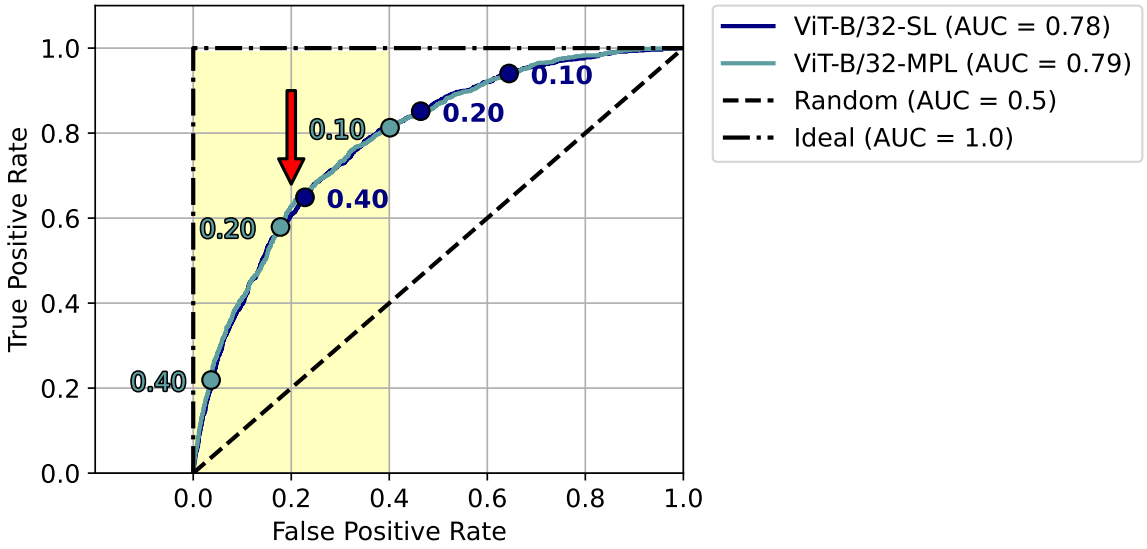


Figure 5.10: ROC curve of ViT-B/32 on the BDD-100K dataset. There is a slight improvement for FPR values lower than 0.2. Yellow area represents constraints on the working points, considering FPR values lower than 0.4, to avoid having too many false positives.

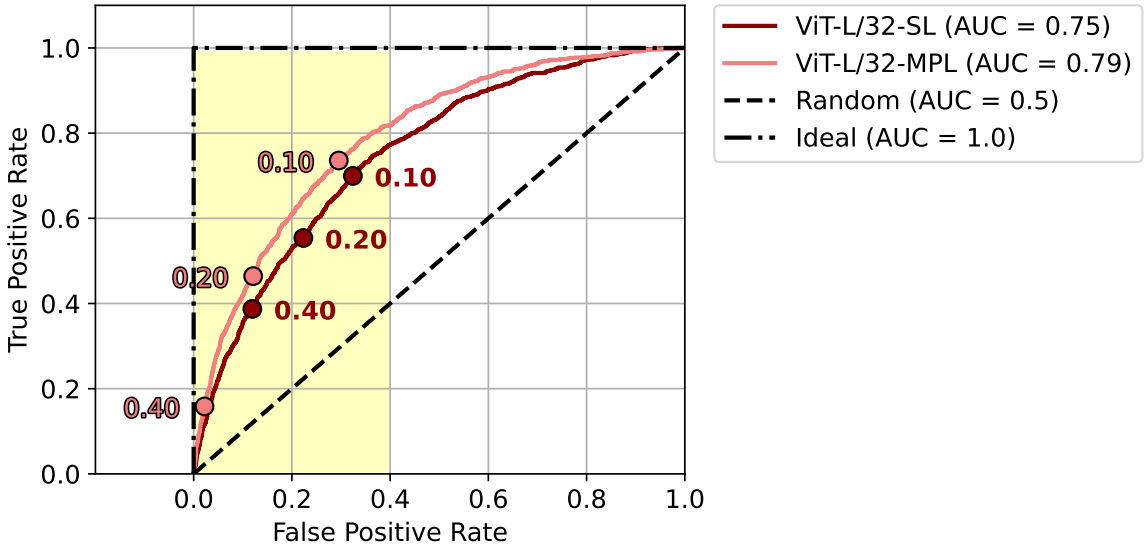


Figure 5.11: ROC curve of ViT-L/32 on BDD-100K dataset. The model trained in supervised mode always underperforms the semi-supervised model. The yellow area represents constraints on the working points, considering FPR values lower than 0.4, to avoid having too many false positives.

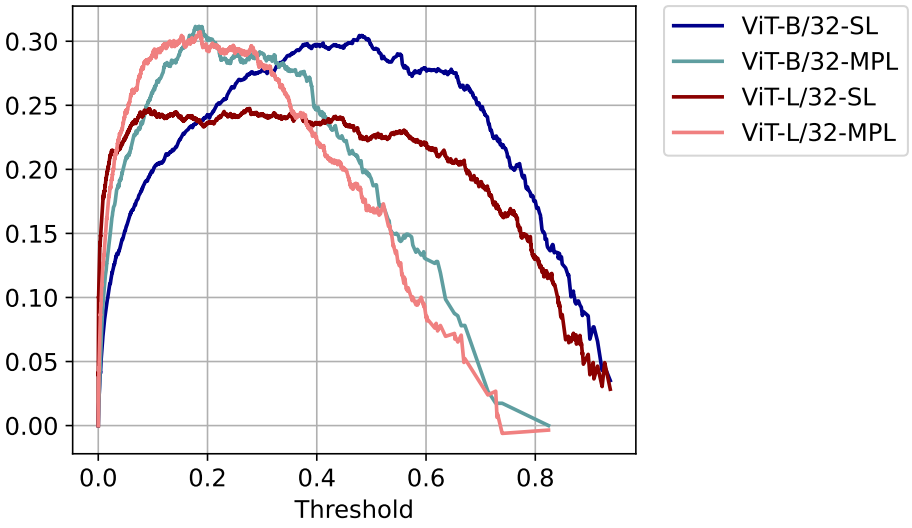


Figure 5.12: MCC comparison between ViT-B/32 and ViT-L/32 in supervised (SL) and semi-supervised (MPL) learning, with different thresholds.

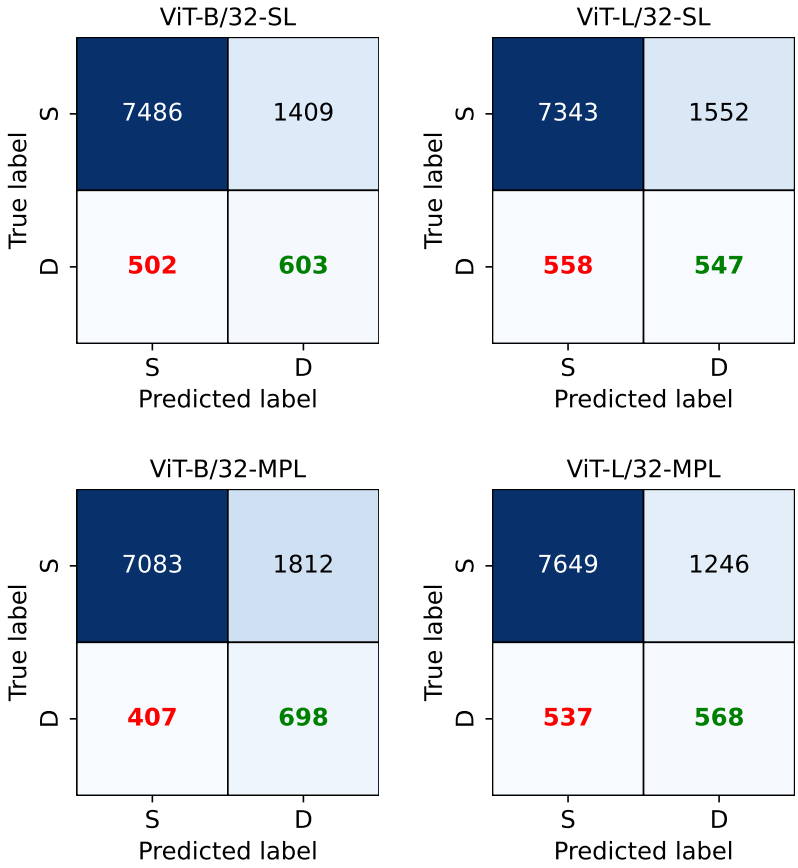


Figure 5.13: Confusion matrices at maximum MCC for ViT-B/32 and ViT-L/32. False negatives are marked in red, meaning that it is important to not miss them. True negatives are marked in green, meaning that we mainly want to increase them.

It is worth noting that it is not possible to only weight false negatives, because this would lead to a working point with a threshold of 0, where all the frames are classified as dangerous. This, of course, is not a good solution, because does not involve any training of the model.

In Figure 5.14 we show the cost function in all the cases, and it is possible to notice that in this case the lowest cost is reached by the ViT-L/32 model in semi-supervised learning mode. Even though it is a slight improvement, it is an interesting result because with the agnostic analysis the ViT-B/32 model in semi-supervised learning mode was the best. This means that in a more general perspective, ViT-B/32-MPL works better, but considering the specific problem to solve, ViT-L/32-MPL can capture more information from the training set.

In Figure 5.15 we show the confusion matrices at the minimum cost for the two models. In this case, it is possible to notice that for the ViT-B/32 model, the semi-supervised learning helps to reduce false positives from 3,410 to 2,875 but also increases false negatives from 212 to 266. On the other hand, for the ViT-L/32 model, the semi-supervised learning helps both to reduce false positives from 3,232 to 3,060 and false negatives from 281 to 238. Therefore, despite the slight improvement, it is interesting to notice how Meta Pseudo-Labels helps to increase the generalization of the larger model without negative impacts, when considering the specific problem to solve.

Possible Problems

In the training pipeline there are various assumptions that were made. In particular, the first regards data pre-processing: for the experiments we used a resolution of 600x600 pixels, that is a good compromise between the quality of the images and the computational cost. However, considering that we chose to use the BDD-100K dataset to have human annotations, it is possible that the resizing could have lost some important information for the model. To partially mitigare this problem, we used a minimum threshold area of 7,000 pixels to consider the objects in the scene. This means that small objects are discarded from the analysis, but it is complicated to find the right trade-off between these two aspects.

Another problem is related to the unbalanced dataset. In fact, even though the problem is partially solved by adjusting the classification threshold, there is much variation of informaiton in the labelled dataset. This is because human-labelled frames consist of the tenth second of each video, and each video is recorded in different areas and conditions. Therefore, even though Meta Pseudo-Labels can slightly help to reduce the overfitting, it is still complicated to satify all the hypothesis to utilize semi-supervised learning techniques.

A further consideration regards the definition of the models. In fact, especially for the vision transformer, there are many hyperparameters to tune. In this case, to simplify this step we chose two standard models, with the largest patch size, already used in other benchmark datasets. However, the task to accomplish is different than the common classification tasks, and it is complicated to understand how the model learns the features of the dataset. therefore, it is possible that different configurations of the models could lead to better results.

Finally, to simplify the data preparation, we did not use any human bias to classify the images. Using the artificial bias, on one hand, helps to obtain many dangerous

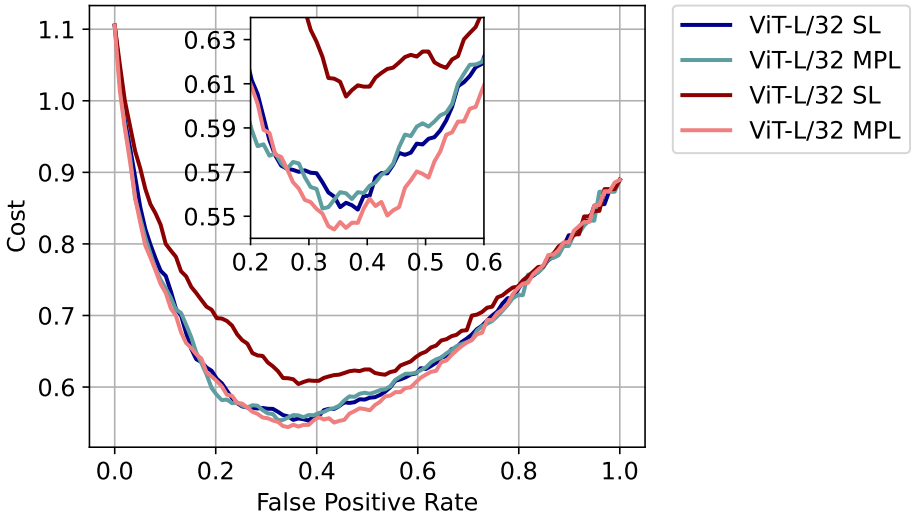


Figure 5.14: Cost function of ViT-B/32 and ViT-L/32 in supervised (SL) and semi-supervised (MPL) learning, with different thresholds.

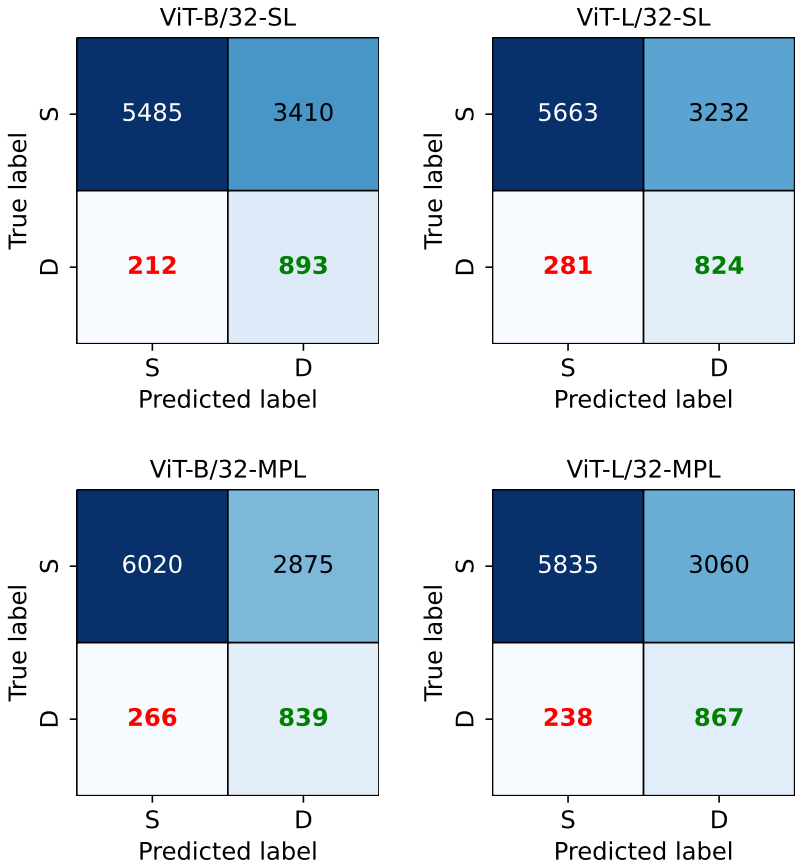


Figure 5.15: Confusion matrices at minimum cost for ViT-B/32 and ViT-L/32. False negatives are marked in red, meaning that it is important to not miss them. True negatives are marked in green, meaning that we mainly want to increase them.

frames without any particula effort. On the other hand, the problem is reduced to the presence of small targets in the scene, and there is no spatial relation between them and sorrounding sub-areas. Thi means that information characterizing the problem is drastically reduced.

5.2.4 Few-Shots Inference of GPT4-o

In May 2024, OpenAI released GPT4-o, their multilingual, multimodal generative pre-trained transformer. Considering its powerful capability to describe general contexts in wide scenes, we decided to test the architecture to solve a problem similar to ours. Experiments were conducted on BDD-100K dataset.

Problem Definition

We defined two groups of six images each. The first group consists of general driving scenarios that contain at least one person in each image. The second group, on the other hand, is still related to driving scenarios but does not contain any person. However, in most images of both groups there are some common objects, including cars, buildings, etc. Then the model will be asked to evaluate a new, never seen, image and answer in which group it belongs.

Other experiments in the two groups were done, including asking the model which common targets are present in each group. The first group of images is shown in Figure 5.16. It consists of six images that do not contain any person. In particular, in the first image on the top left there is the ego-vehicle in a highway with some other vehicles on the front. The second on the top-center represents a downtown scenario with some vehicles on the front. In the third image on top-right there is a vehicle on the front and many vehicles parked on both sides of the road. The fourth image on bottom-left represents the ego-vehicle driving on a road with a traffic divider and some cars parked on the right side. In the fifth image on the bottom-center there is a straight road with some vehicles on the front and tall buildings on the sides. The last image on the bottom-right represents a downtown scenario with some vehicles, including cars and a bus. There is actually an occluded person in the image, but it is almost not visible.

The second group of images is shown in Figure 5.17. In the first image on the top-left there are two cars in fron of the ego-vehicle, some cars parked on both sides and a pedestrian standing on the right side. There are also some other pedestrians far away on sidewalks, not much visible in the image. The second image on the top-center represents a downtown scenario where some pedestrians are crossing the road. There is also a car partially visible on the right of the ego-vehicle. The environment is characterized by tall buildings. The third image on the top-right represents a downtown scenario with some empty crosswalks but many pedestrians on sidewalks. There are also some cars driving on the main road. The fourth image on the bottom-left represents a night downtown scenario with a taxi in front of the ego-vehicle and group of pedestrians on the left sidewalk and far away, where there are crosswalks and red traffic lights. In the fifth image, bottom-center, there is a line of cars in front of the ego-vehicle, some cars parked on the left side, and a pedestrian standing on the left sidewalk. The last image on the bottom-right represents a main road with some vehicles parked on both sides, some others driving, and a person standing on the right side of the road.

Figure 5.16: Group of safe images for the features extraction task. There are no people in all the images.

Considering the dangerous scenarios, some specific cases were carefully selected, especially with people partially visible, in the corners of the images, and on the background. On the other hand, in both groups there are many vehicles, both in foreground and background, that contribute to add noise to the data to classify. This decision represents a further challenge for the model to isolate the right group of pixels.

Common Features Extraction

The first experiment consists of showing the model the two groups of images, without explicitly explaining the difference between them. The model is asked to describe the common targets in each group. The question-answer pairs are shown below:

Q: What common targets do all the images have? Do not list a target if it is not inside all the images. (*showing the dangerous images*)

A: Cars, roads, people, sky.

Q: What common targets do all the images have? Do not list a target if it is not inside all the images. (*showing the safe images*)

A: Road, other vehicles, sorrounding environment.

It is possible to notice that for the dangerous images the model correctly detects also people as common targets. Moreover, vehicles are shown in both groups, and the model is able to distinguish them as common targets correctly. This means that the model is able to extract the common noise in both groups (the presence of cars, that usually occupy many pixels with respect to people), but at the same time it is able to extract features of people related to the dangerous group.

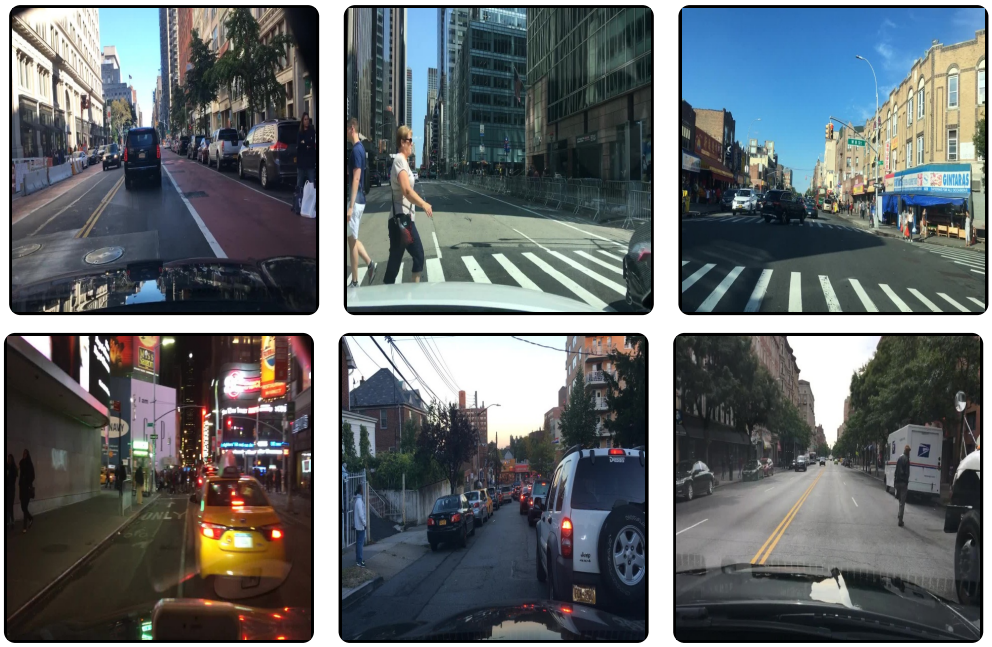


Figure 5.17: Group of dangerous images for the features extraction task. There is at least one person in each image.

This is a promising result, considering that GPT4 is a pretrained model on general multimodal data, and not specifically on driving scenarios. However, it has over 174 billion parameters, and that is why it is capable to generalize in many different contexts. Using a vision transformer architecture for specific tasks, like driving scenarios, with much less parameters, could still lead to good results.

Grouping and Classification

In this experiment we only showed the model the two groups of images, without asking to extract common targets. Then we show a new image and ask the model to classify it in the right group. The question-answer pairs are shown below:

Q: I will show you two groups of images. Then I will show you a test image and you have to tell me what group it is related to. You have to identify common features for each group to find the correct matching. Do not output anything before I have finished, when I will tell you "predict this image".

A: Understood! I'm ready when you are to proceed with the image groups and the test image.

Q: This is the first group. (*showing the first group of images*)

A: I've received the first group of images. Please show me the next group.

Q: This is the second group. (*showing the second group of images*)

A: I've received the second group of images. Please proceed with showing the test image when you are ready.

Figure 5.18: Group of safe images for the classification task. There are no people in all the images.

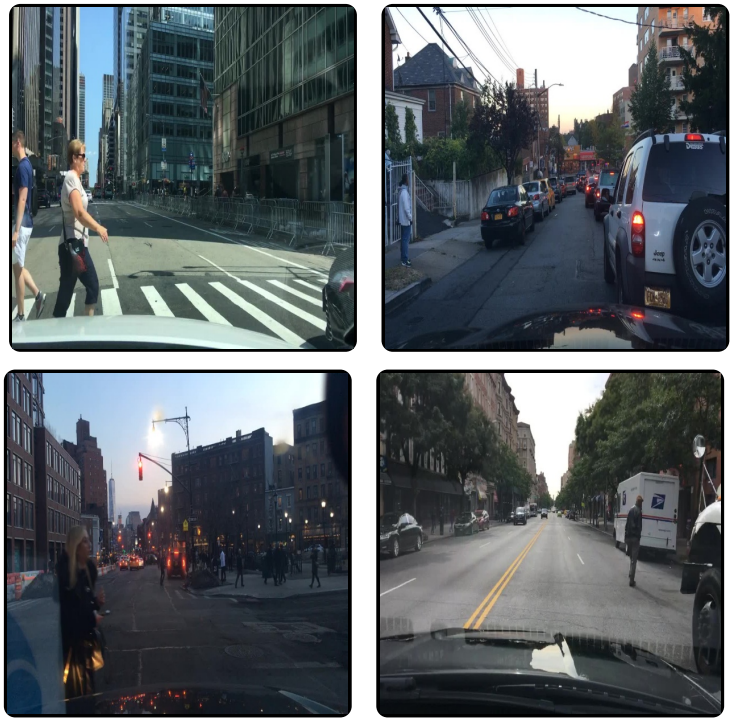


Figure 5.19: Group of dangerous images for the classification task. There is at least one person in each image.

Figure 5.20: Test image for the classification task.

Q: This is the test image. (*showing the test image*)

A: Based on the test image and comparing it to the two groups shown earlier: The test image features an urban setting with buildings lining the streets and multiple vehicles on the road. It closely resembles the environment and context of the images in the second group, which also depict urban settings with buildings, streets, and a visible presence of pedestrians or traffic. This environment is distinct from the first group, which primarily showed roads in more open, less densely populated areas or showing less urban characteristics. Therefore, the test image belongs to the second group.

Also in this experiment the model is able to correctly classify the test image in the right group. The model is able to give a detailed explanation of the features that are common in the test image and in the second group of images. It is impressive that the model is capable to give such detailed explanations of contexts in images, considering that it is a general model and not specifically trained on driving scenarios.

Chapter 6

Conclusions

In this thesis we studied and discussed different approaches that are suitable for the scene understanding task in driving scenarios. We started by discussing how traditional computer vision-based approaches can be used, with a driver-attention model to combine main perception algorithms, but also how they poorly perform in complex driving scenarios. We then discussed how deep learning-based approaches can be considered a more robust solution, and how they can be used to fit human-decision biases. However, the solution we developed is not optimal; there are many improvements that can be made, considering all the problems we faced during the development.

Therefore, we conclude the thesis with three sections: the first one summarizes the main points of experiments based on the traditional computer vision-based approach, the second one highlights important aspects for the deep learning-based approach. Finally, the last section discusses possible future work and improvements that can be made to the solution developed in this thesis. It also considers how the two methods can be combined, integrating the driver-attention state with the outside-world attention map.

6.1 Traditional Computer Vision-based Approach

Within the traditional computer vision-based approach, we have seen that homography is a robust method to project driver's gaze on the roof top camera to have a better view of the scene. However, estimating the homography matrix requires high computational power, and it is not always accurate when environmental conditions are not optimal.

We showed that using specific states of observation and detection can help to analyze the interaction between the driver and vulnerable road users. However, many parts of the proposed model generate some noise that affects the results. In fact, we found that the tracking algorithm based on ByteTrack [61] is not always accurate, leading to tracking mismatches and loss of information.

Finally, we compared monocular depth estimation provided by the MiDaS model [39] with the ground truth depth map provided by LiDAR sensors on the NuScenes dataset [3]. We found that the model is not accurate, especially for mid-long range distances, and it is not suitable for the proposed driver-attention model. This is probably because the model is trained on large general datasets, and it is not fine-tuned for specific tasks. The same problem is found in the object detection model, YOLOv8 [40], that could be fine-tuned for detecting vulnerable users on driving datasets.

6.2 Deep Learning-based Approach

Through the deep learning-based approach, we used a vision transformer model [11] to extract information related to human-decision biases. To simplify the problem at the beginning, we used an artificial bias to classify dangerous situations depending on the presence of some targets in the image. First experiments on the DR(eye)VE dataset [36] showed advantages in using Meta Pseudo-Labels [38], but we found that data was not completely independent on the training and validation sets. Moreover, when using RetinaNet [29] as the object detection model, we found difficulties in configuring the detector without an appropriate fine-tuning. Also the quantity of data was not enough to evaluate semi-supervised learning techniques.

That is why we decided to move to the BDD-100K dataset [57], a large dataset with 80,000 manually annotated images and 100,000 unlabelled videos suitable for semi-supervised learning. We used standard vision transformer architectures with large patch sizes to avoid problems related to the choice of the right hyperparameters. We found that the model trained with Meta Pseudo-Labels slightly outperformed the model trained with supervised learning, but the difference was not significant except for the working point with highest correlation. After making more experiments increasing complexity of the model, we found that it leads to a significant deterioration of correlation in the predictions in supervised learning. Meta Pseudo-Labels helped to improve the metric, reaching similar results to the smaller model. However, defining the optimal working point depending on the optimization of a custom cost function, we found that the best performance was achieved by the larger model in semi-supervised learning.

Finally, we analyzed the attention mechanism of a recent pre-trained multi-modal transformer, GPT-4o. In particular we found that it has exceptional capabilities of contextualizing complex driving scenarios and it is able to extract common features from few images. Therefore, we used the model in few-shots inference mode, showing it some dangerous and safe samples, without explicitly providing information about the artificial bias. We found that the model is able to capture the relations and generalize the bias to new test samples. This promising result suggests that large pre-trained models can be used to transfer task-specific knowledge to smaller models through knowledge distillation [22] in few-shots inference mode.

6.3 Future Work

In this thesis, we focused both on the model architecture and the right data to use for accomplishing the scene understanding task in driving scenarios. However, data is a crucial aspect in the development of deep learning models, and it is fundamental that it carries the right information to explain the problem to solve. For simplicity we used an artificial bias, but it drastically simplifies the problem, casusing an important loss of information for the model. Moreover, reducing the task to the presence of some targets, we are not considering spatial relations of patches in the image.

Therefore, as future improvements, we suggest to use some annotations related to actual driving dangerous situations to train the model. There are recent works [53] that provide this type of annotations for the BDD-100K dataset. In this way, it is still possible to leverage the large amount of unlabelled data to train the model in a

semi-supervised learning setting, but with high-quality annotations that can provide more information to the model.

Another possible method that can help with the labelling process is the use of knowledge distillation with large-pretrained models. Showing the large model some dangerous and safe samples, can also help to generate higher-quality pseudo-labels. The mechanism is in the middle between semi-supervised learning and transfer learning: the teacher model is not trained at all, used in inference mode, but from its pretrained knowledge and the attention mechanism, it is able to adapt the output to our task-specific problem. Therefore, it can be used to significantly increase the labelled set. Through this process, it is possible to analyze how the quality of the labels change depending on how many samples are shown to the teacher model. Knowledge distillation is a relatively new field also used in other applictions, and deserves further investigation.

It is also possible to combine the driver's gaze with an attention map extracted from the attention heads of the vision transformer. In fact, the model, through the attention mechanism, is able to extract a score map that weights the importance of each patch in the image. This map can be compared with the gaze of the driver during the time, to understand if the driver is aware of the dangerous situation. However, further considerations are needed to understand the temporal dependencies of these signals.

Finally, we suggest also to investigate the branch of unsupervised learning. In fact, if the problem is only related to anomaly detection in the scene, there are recent works that try to address it in driving scenarios, typically with the use of convolutional autoencoders. However, the use of vision transformers in unsupervised learning is still an open field, and it is possible to investigate how the attention mechanism can help to extract the most important features from the scene.

Appendices

Appendix A

Training Algorithms

A.1 Pre-processing of the DR(eye)VE dataset

Algorithm 2 describes how the DR(eye)VE dataset is pre-processed before training the model. The dataset is split into training, validation, test, and unlabelled sets. The **Dreyeve** data structure accepts the split indexes, resolution, and frames per second as input parameters, where split indexes are used to divide the dataset into the sets.

A validator model is used to label the training, validation, and test sets. The validator is a pre-trained **RetinaNet** model. Labels are generated depending on the objects detected by the validator. Finally, all the sets are saved to disk for further training.

Algorithm 2: Pre-processing of the DR(eye)VE dataset

trainSet, valSet, testSet, ulSet ← **Dreyeve**(*splitIdxs, resolution, fps*)
validator ← **RetinaNet**( )
for *set* ← {*trainSet, valSet, testSet*} **do**
    labels ← validator(*set*)
    set.labels ← *labels*
end
save *trainSet, valSet, testSet, ulSet*

A.2 Pseudo-Labelling

Algorithm 3 describes the semi-supervised training with Meta-Pseudo Labels [38]. Splits of the dataset are loaded, and two models are initialized: a teacher and a student. The models are based on a vision transformer architecture, represented by the **ViT** model. Most important hyperparameters that need to be defined are the patch size, depth of the model, embedding size, dimension of the multi-layer perceptron layer, and number of attention heads.

The training loop is executed until convergence of the validation accuracy of the student model. In each iteration, in order, the teacher generates pseudo-labels on the unlabelled set, and the student is trained with the pseudo-labels on the unlabelled

set. Then, both the student and teacher are trained on the training set. The student model, that is supposed to perform better than the teacher, is saved if the validation accuracy is increased.

Algorithm 3: Semi-supervised training with Meta-Pseudo Labels

```
trainSet, valSet, testSet, ulSet ← load data
teacher ← ViT(patch, depth, embedding, mlp, heads)
student ← ViT(patch, depth, embedding, mlp, heads)
while training do
    pseudoLabels ← teacher(ulSet)
    train(student, ulSet, pseudoLabels)
    train(student, trainSet)
    train(teacher, trainSet)
    S.metrics ← validate(student, valSet)
    T.metrics ← validate(teacher, valSet)
    if S.metrics.accuracy is increased then
        save student
    end
end
```

A.3 Correction with Wrong Predictions

Algorithm 4 describes the process of updating the training set of DR(eye)VE to correct the model on the wrong predictions. The training set and the trained model are loaded, and a validator model **RetinaNet** is initialized.

The algorithm iterates over the unlabelled set, and compares the predictions of the model with the labels provided by the validator. If the prediction is wrong, the sample is added to the training set and the respective counter is increased. The process is repeated until the maximum number of wrong predictions is reached for each class.

This process also helps to rebalance the dataset during training. However, if the there are not enough wrong predictions to update the dataset, the algorithm stops after iterating over the unlabelled set. Therefore, an unbalanced quantity of new labels is added to the training set.

Algorithm 4: Update of training set with wrong predictions

```
trainSet ← load training set
model ← load trained model
validator ← RetinaNet( )
S.count, S.max ← 0, N/2
D.count, D.max ← 0, N/2
for sample ← ulSet do
    if (S.count = S.max) and (D.count = D.max) then
        break
    end
    label ← validator(sample)
    pred ← model(sample)
    if pred = label then
        continue
    end
    if (label=Safe) and (S.count <S.max) then
        trainSet.append(sample, label)
        S.count ← S.count +1
    end
    if (label=Dangerous) and (D.count <D.max) then
        trainSet.append(sample, label)
        D.count ← D.count +1
    end
    if endof ulSet then
        break
    end
end
save trainSet
```

Appendix B

Probabilistic Perspective of AUC

In this appendix, we provide a more formal and detailed explanation of the AUC-ROC from a probabilistic perspective. It is fundamental to better interpret results on unbalanced datasets and to understand the trade-off between TPR and FPR. We took inspiration from the work of David J. Hand [18] and adapted the explanation to our specific case.

B.1 Definitions

A trained model for binary classification can also be considered as a probabilistic classifier that outputs a score $s(\boldsymbol{x}) = p(\text{class} = 1 \,|\, \boldsymbol{x})$, where $\boldsymbol{x}$ is the input data. Considering the set of classes $k = \{0, 1\}$, $f_k(s)$ is the Probability Density Function (p.d.f.) of the score s for class k. Its Cumulative Density Function (CDF) is defined as $F_k(s)$. Therefore FPR and TPR can be defined as:

$$\begin{aligned} \text{FPR} &= 1 - F_0(s) = 1 - P(\boldsymbol{X_0} \leq s) = P(\boldsymbol{X_0} > s) \\ \text{TPR} &= 1 - F_1(s) = 1 - P(\boldsymbol{X_1} \leq s) = P(\boldsymbol{X_1} > s) \end{aligned}$$

where $\boldsymbol{X_0}$ and $\boldsymbol{X_1}$ are random variables representing the score s for a random sample of class 0 and 1, respectively. For simplicity, FPR and TPR are respectively defined as $x(s)$, $y(s)$, and the decision threshold is τ. Moreover, from the definition of ROC curve, we have $y(x(\tau)) = y(\tau)$. Then we have:

$$\begin{aligned} FPR &= x(s) \\ TPR &= y(s) \end{aligned}$$

B.2 Derivation

Considering that the AUC is the area under the ROC curve, we can express it as:

$$\begin{aligned} \text{AUC} &= \int_0^1 y(x)dx \\ &= \int_0^1 y(x(\tau))dx(\tau) \end{aligned}$$

Changing the variable of integration from x to τ, and knowing that $dx(\tau) = x'(\tau)d\tau$:

$$\begin{aligned}
\text{AUC} &= \int_{+\infty}^{-\infty} y(\tau)x'(\tau)d\tau \\
&= \int_{+\infty}^{-\infty} (1 - F_1(\tau))(-f_0(\tau))d\tau \\
&= \int_{-\infty}^{+\infty} (1 - F_1(\tau))f_0(\tau)d\tau \qquad \text{(B.1)} \\
&= \int_{-\infty}^{+\infty} P(\boldsymbol{X_1} > \tau)P(\boldsymbol{X_0} = \tau)d\tau \qquad \text{(B.2)}
\end{aligned}$$

B.3 Interpretation

From Equation B.1, we can interpret the AUC as the probability that a randomly chosen positive sample has a higher score than a randomly chosen negative sample. This is equivalent to multiplying the CDF of the positive class by the p.d.f. of the negative class and integrating over all possible values of the score. A more intuitive notation is also given in Equation B.2.

Considering our problem of detecting dangerous events in driving scenes, the AUC can be interpreted as the probability that a randomly chosen dangerous case has a higher score than a randomly chosen safe case. In other words, the probability that the model does not misclassify a dangerous event as safe. This is a fundamental aspect to consider when evaluating the performance of a model on unbalanced datasets. Moreover, we want to weight more false negatives than false positives, as the consequences of missing a dangerous event are more severe than misclassifying a safe event. Therefore, the AUC is a good metric to evaluate the performance of a model in this context, defining dangerous events as the positive class.

Bibliography

[1] Anurag Arnab et al. *ViViT: A Video Vision Transformer*. 2021. arXiv: 2103.15691 [cs.CV].

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.

[3] Holger Caesar et al. *nuScenes: A multimodal dataset for autonomous driving*. 2020. arXiv: 1903.11027 [cs.LG].

[4] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV].

[5] Joao Carreira and Andrew Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2018. arXiv: 1705.07750 [cs.CV]. URL: https://arxiv.org/abs/1705.07750.

[6] Fu-Hsiang Chan et al. "Anticipating Accidents in Dashcam Videos". In: *Computer Vision – ACCV 2016*. Ed. by Shang-Hong Lai et al. Cham: Springer International Publishing, 2017, pp. 136–153. ISBN: 978-3-319-54190-7.

[7] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-supervised learning*. MIT Press, 2010.

[8] Yong Shean Chong and Yong Haur Tay. "Abnormal Event Detection in Videos using Spatiotemporal Autoencoder". In: *CoRR* abs/1701.01546 (2017). arXiv: 1701.01546. URL: http://arxiv.org/abs/1701.01546.

[9] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].

[10] Ali Diba et al. *Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification*. 2017. arXiv: 1711.08200 [cs.CV]. URL: https://arxiv.org/abs/1711.08200.

[11] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[12] Jianwu Fang et al. "DADA-2000: Can Driving Accident be Predicted by Driver Attention? Analyzed by A Benchmark". In: *CoRR* abs/1904.12634 (2019). arXiv: 1904.12634. URL: http://arxiv.org/abs/1904.12634.

[13] Jianwu Fang et al. "DADA: A Large-scale Benchmark and Model for Driver Attention Prediction in Accidental Scenarios". In: *CoRR* abs/1912.12148 (2019). arXiv: 1912.12148. URL: http://arxiv.org/abs/1912.12148.

[14] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Readings in Computer Vision*. Ed. by Martin A. Fischler and Oscar Firschein. San Francisco (CA): Morgan Kaufmann, 1987, pp. 726–740. ISBN: 978-0-08-051581-6. DOI: https://doi.org/10.1016/B978-0-08-051581-6.50070-2. URL: https://www.sciencedirect.com/science/article/pii/B9780080515816500702.

[15] Jiyang Gao, Zhenheng Yang, and Ram Nevatia. *RED: Reinforced Encoder-Decoder Networks for Action Anticipation*. 2017. arXiv: 1707.04818 [cs.CV]. URL: https://arxiv.org/abs/1707.04818.

[16] A Geiger et al. "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237. DOI: 10.1177/0278364913491297. eprint: https://doi.org/10.1177/0278364913491297. URL: https://doi.org/10.1177/0278364913491297.

[17] Rui Gong et al. *Self-Calibration Supported Robust Projective Structure-from-Motion*. 2020. arXiv: 2007.02045 [cs.CV].

[18] David J. Hand. "Measuring classifier performance: a coherent alternative to the area under the ROC curve". In: *Machine Learning* 77 (2009), pp. 103–123. URL: https://api.semanticscholar.org/CorpusID:207211591.

[19] Mahmudul Hasan et al. "Learning Temporal Regularity in Video Sequences". In: *CoRR* abs/1604.04574 (2016). arXiv: 1604.04574. URL: http://arxiv.org/abs/1604.04574.

[20] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[21] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[22] Cheng-Yu Hsieh et al. *Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes*. 2023. arXiv: 2305.02301 [cs.CL]. URL: https://arxiv.org/abs/2305.02301.

[23] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[24] M I Jordan. "Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986". In: (May 1986). URL: https://www.osti.gov/biblio/6910294.

[25] Nikhil Kaitwade. *Automotive ADAS (Advanced Driver Assistance System) Market Outlook*. Online. Accessed: 2024-06-06. Nov. 2023. URL: https://www.futuremarketinsights.com/reports/adas-market.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[27] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. *Stereo R-CNN based 3D Object Detection for Autonomous Driving*. 2019. arXiv: 1902.09738 [cs.CV].

[28] Weixin Li, Vijay Mahadevan, and Nuno Vasconcelos. "Anomaly Detection and Localization in Crowded Scenes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.1 (2014), pp. 18–32. DOI: 10.1109/TPAMI.2013.111.

[29] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[30] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[31] Wen Liu et al. "Future Frame Prediction for Anomaly Detection - A New Baseline". In: *CoRR* abs/1712.09867 (2017). arXiv: 1712.09867. URL: http://arxiv.org/abs/1712.09867.

[32] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[33] Cewu Lu, Jianping Shi, and Jiaya Jia. "Abnormal Event Detection at 150 FPS in MATLAB". In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 2720–2727. DOI: 10.1109/ICCV.2013.338.

[34] Zhichao Lu et al. *RetinaTrack: Online Single Stage Joint Detection and Tracking*. 2020. arXiv: 2003.13870 [cs.CV].

[35] Weixin Luo, Wen Liu, and Shenghua Gao. "Remembering history with convolutional LSTM for anomaly detection". In: *2017 IEEE International Conference on Multimedia and Expo (ICME)*. 2017, pp. 439–444. DOI: 10.1109/ICME.2017.8019325.

[36] Andrea Palazzi et al. *Predicting the Driver's Focus of Attention: the DR(eye)VE Project*. 2018. arXiv: 1705.03854 [cs.CV].

[37] Wanli Peng et al. "IDA-3D: Instance-Depth-Aware 3D Object Detection From Stereo Vision for Autonomous Driving". In: (2020).

[38] Hieu Pham et al. *Meta Pseudo Labels*. 2021. arXiv: 2003.10580 [cs.LG].

[39] René Ranftl et al. *Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer*. 2020. arXiv: 1907.01341 [cs.CV].

[40] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640.

[41] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[42] Zheng Shou et al. *Online Detection of Action Start in Untrimmed, Streaming Videos*. 2018. arXiv: 1802.06822 [cs.CV]. URL: https://arxiv.org/abs/1802.06822.

[43] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[44] Waqas Sultani, Chen Chen, and Mubarak Shah. "Real-world Anomaly Detection in Surveillance Videos". In: *CoRR* abs/1801.04264 (2018). arXiv: 1801.04264. URL: http://arxiv.org/abs/1801.04264.

[45] Pei Sun et al. *Scalability in Perception for Autonomous Driving: Waymo Open Dataset*. 2020. arXiv: 1912.04838 [cs.CV].

[46] Peize Sun et al. *TransTrack: Multiple Object Tracking with Transformer*. 2021. arXiv: 2012.15460 [cs.CV].

[47] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[48] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[49] Roberto Toldo et al. "Hierarchical structure-and-motion recovery from uncalibrated images". In: *Computer Vision and Image Understanding* 140 (Nov. 2015), pp. 127–143. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2015.05.011. URL: http://dx.doi.org/10.1016/j.cviu.2015.05.011.

[50] Du Tran et al. "Learning Spatiotemporal Features with 3D Convolutional Networks". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4489–4497. DOI: 10.1109/ICCV.2015.510.

[51] Michal Uricar et al. *Challenges in Designing Datasets and Validation for Autonomous Driving*. 2019. arXiv: 1901.09270 [cs.CV].

[52] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[53] Ye Xia et al. "Training a network to attend like human drivers saves it from common but misleading loss functions". In: *CoRR* abs/1711.06406 (2017). arXiv: 1711.06406. URL: http://arxiv.org/abs/1711.06406.

[54] Yinghao Xu et al. *Cross-Model Pseudo-Labeling for Semi-Supervised Action Recognition*. 2022. arXiv: 2112.09690 [cs.CV]. URL: https://arxiv.org/abs/2112.09690.

[55] Yu Yao et al. "Unsupervised Traffic Accident Detection in First-Person Videos". In: *CoRR* abs/1903.00618 (2019). arXiv: 1903.00618. URL: http://arxiv.org/abs/1903.00618.

[56] Yu Yao et al. "When, Where, and What? A New Dataset for Anomaly Detection in Driving Videos". In: *CoRR* abs/2004.03044 (2020). arXiv: 2004.03044. URL: https://arxiv.org/abs/2004.03044.

[57] Fisher Yu et al. *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning*. 2020. arXiv: 1805.04687 [cs.CV].

[58] Guoshen Yu and Jean-Michel Morel. "ASIFT: An Algorithm for Fully Affine Invariant Comparison". In: *Image Processing On Line* 1 (2011). https://doi.org/10.5201/ipol.2011.my-asift, pp. 11–38.

[59] Jongsub Yu and Hyukdoo Choi. "YOLO MDE: Object Detection with Monocular Depth Estimation". In: *Electronics* 11.1 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11010076. URL: https://www.mdpi.com/2079-9292/11/1/76.

[60] Ming Zeng et al. "Semi-supervised convolutional neural networks for human activity recognition". In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 522–529. DOI: 10.1109/BigData.2017.8257967.

[61] Yifu Zhang et al. *ByteTrack: Multi-Object Tracking by Associating Every Detection Box*. 2022. arXiv: 2110.06864 [cs.CV].