Sensors, algorithms, and representations for efficient environment perception

Hehn, T.M.

DOI
10.4233/uuid:5b48167c-80b7-4b78-b2d9-59e192a7bc6f

Publication date
2022

Document Version
Final published version

Citation (APA)
Hehn, T. M. (2022). *Sensors, algorithms, and representations for efficient environment perception*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:5b48167c-80b7-4b78-b2d9-59e192a7bc6f

Sensors, algorithms, and representations for efficient environment perception

Sensors, algorithms, and representations for efficient environment perception

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op donderdag 3 November 2022 om 15:00 uur

door

Thomas Markus HEHN

Master of Science in Physics,
Heidelberg University, Duitsland,
geboren te Stuttgart, Duitsland.

Dit proefschrift is goedgekeurd door de

promotor: Prof. dr. Dariu M. Gavrila
copromotor: Dr. J.F.P. Kooij

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. D.M. Gavrila,	Technische Universiteit Delft
Dr. J.F.P. Kooij,	Technische Universiteit Delft
Independent members:	
Prof. dr. T. Gevers,	Universiteit van Amsterdam
Assist. Prof. dr. A. Valada,	Universität Freiburg, Germany
Prof. dr. ir. M. Wisse,	Technische Universiteit Delft
Prof. dr. R. Babuska,	Technische Universiteit Delft
Dr. H. Caesar,	Technische Universiteit Delft

TUDelft Delft University of Technology

NWO Nederlandse Organisatie voor Wetenschappelijk Onderzoek

Keywords: Autonomous Vehicles, Machine Learning, Perception, Representation

Printed by: Gildeprint

Style: Based on TU Delft House Style with modifications by Moritz Beller
`https://github.com/Inventitech/phd-thesis-template`

The author set this thesis in LaTeX using the Libertinus and Inconsolata fonts.

ISBN 978-94-6384-383-6

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

Contents

Summary

Already today, consumer-grade cars are equipped with advanced driver assistance systems that do not require any action from the drivers for a short period of time. Although these systems are still limited and only reliable in certain situations, it shows the general trend: cars will become more and more autonomous. The reasons why people and companies are eagerly anticipating fully autonomous cars are manifold: self-driving vehicles could provide mobility to people unable to drive themselves, they could reduce the need for parking spaces in inner cities, they could decrease traffic jams, and of course, they let passengers spend their time on something else than actively driving. Self-driving vehicles also have the potential to eliminate human error as a cause of traffic accidents and thereby increase traffic safety. Thus, the presence of driver assistance systems and self-driving vehicles in traffic will inevitably increase, and it is crucial to make the technology as safe as possible.

An essential building block to achieving safe autonomous driving is the efficient perception and representation of the vehicle's environment. The perception and representation need to be as accurate as possible, but at the same time, as efficient as possible, to increase the time in which the vehicle can react to the evolving traffic situation. This thesis discusses various ways to increase the *efficiency* of perception systems of autonomous vehicles by showing: how a novel acoustic sensor detects traffic before it becomes visible, how to combine traditional machine learning algorithms with deep neural networks for faster inference, how a compact representation for images of traffic scenes can be enriched with object instance information, and how different modalities, such as images and point clouds, contribute to deep representation learning.

To detect vehicles ahead of commonly used sensors in autonomous vehicles, this thesis introduces a passive acoustic perception approach. This acoustic perception system can detect approaching vehicles behind blind corners by sound before such vehicles enter in line-of-sight. A research vehicle equipped with a roof-mounted microphone array is used to collect data and serves as a demonstrator platform. The data shows that wall reflections provide information on the presence and direction of occluded approaching vehicles. In test scenarios with a static ego-vehicle, a novel data-driven approach achieves an accuracy of 0.92 on the hidden vehicle classification task. Compared to a state-of-the-art visual detector, Faster R-CNN, the acoustic system achieves the same accuracy more than one second ahead, providing crucial reaction time for the situations studied in this work. While the ego-vehicle is driving, acoustic detection shows encouraging results, still achieving an accuracy of 0.84 within one environment type. Further, failure cases are studied across environments to identify future research directions.

As an improvement of the processing speed of perception algorithms, this thesis presents an approach to combining conventional decision trees with end-to-end learnable neural networks. Conventional decision trees have a number of favorable properties, including a small computational footprint, interpretability, and the ability to learn from little training data. However, they lack a key quality that has helped fuel the deep learning revolution: that of being end-to-end trainable. Other works have addressed this deficit, but at the cost of losing a main attractive trait of decision trees: the fact that each sample is routed along with a small subset of tree nodes only. This thesis presents an end-to-end learning scheme for deterministic decision trees and decision forests. Thanks to a new model and expectation-maximization training scheme, the trees are fully probabilistic at train time, but after an annealing process become deterministic at test time. It is found that the method performs on par or superior to standard learning algorithms for oblique decision trees and forests, and requires fewer computations than its competitors.

A more compact high-level representation of the environment can streamline the communication between software modules. Instance Stixels, presented in this thesis, provide a compact spatial representation by grouping many pixels into superpixels and assigning a position in 3D space. This provides subsequent planning and obstacle evasion algorithms with a compact spatial layout of the scene. State-of-the-art stixel methods fuse dense stereo disparity and semantic class information, e.g. from a Convolutional Neural Network (CNN), into a compact representation of driveable space, obstacles, and background. However, they do not explicitly differentiate instances within the same semantic class. This thesis investigates several ways to augment single-frame stixels with instance information, which can be extracted by a CNN from the RGB image input. As a result, the Instance Stixels algorithm efficiently computes stixels that account for boundaries of individual objects and represents instances as grouped stixels that express connectivity. The approach achieves strong segmentation performance and computational efficiency compared to combining the separate outputs of Semantic Stixels and a state-of-the-art pixel-level CNN. The GPU implementation of the algorithm can process up to 28 frames per second on average for 8 pixels wide stixels on images from the Cityscapes dataset at 1792x784 pixels.

When using multiple sensor modalities for perception, it is important to understand how the data of the different modalities relate in order to find efficient representations for inter-sensor communication. This thesis investigates a common approach in state-of-the-art self-supervised representation learning that takes advantage of multi-view and multi-modal data by aligning the feature representations across views and/or modalities. One chapter of this thesis investigates how aligning representations affects the visual features obtained from cross-view and cross-modal contrastive learning on images and point clouds. On five real-world datasets and on five tasks, 108 models based on four pretraining variations are trained and evaluated. The results show that cross-modal representation alignment discards complementary visual information, such as color and texture, and instead emphasizes redundant depth cues. The depth cues obtained from pretraining improve downstream depth prediction performance. Also overall, cross-modal alignment leads to more robust encoders than pretraining by cross-view alignment, especially on depth prediction, instance segmentation, and object detection.

Finally, the last chapter summarizes how the individual chapters contributed to improved efficiency of environment perception, discusses how different approaches relate and affect each other, and proposes potential research directions.

Samenvatting

Personenauto's zijn nu al uitgerust met geavanceerde rijhulpsystemen die gedurende een korte periode geen actie van de chauffeurs vereisen. Hoewel deze systemen nog beperkt en alleen in bepaalde situaties betrouwbaar zijn, toont het een algemene trend: auto's zullen steeds autonomer worden. De redenen waarom industrie en de bredere samenleving uitkijken naar volledig autonome auto's zijn legio: zelfrijdende voertuigen kunnen mobiliteit bieden aan mensen die niet zelf kunnen rijden, ze kunnen de behoefte aan parkeerplaatsen in binnensteden verminderen, ze kunnen de files verminderen, en natuurlijk laten ze passagiers hun tijd aan iets anders besteden dan actief rijden. Zelfrijdende voertuigen hebben ook het potentieel om menselijke fouten als oorzaak van verkeersongevallen te elimineren en daardoor de verkeersveiligheid te vergroten. Zo zal de aanwezigheid van rijhulpsystemen en zelfrijdende voertuigen in het verkeer onvermijdelijk toenemen en is het cruciaal om de technologie zo veilig mogelijk te maken.

Een essentiële bouwsteen voor veilig autonoom rijden is de efficiënte perceptie en representatie van de omgeving van het voertuig. De waarneming en weergave moeten zo nauwkeurig mogelijk zijn, maar tegelijkertijd zo efficient mogelijk, om de tijd te verlengen waarin het voertuig kan reageren op de veranderende verkeerssituatie. Dit proefschrift bespreekt verschillende manieren om de efficiëntie van waarnemingssystemen van autonome voertuigen te vergroten door te laten zien: hoe een nieuwe akoestische sensor verkeer detecteert voordat het zichtbaar wordt, hoe traditionele machine learning-algoritmen kunnen worden gecombineerd met diepe neurale netwerken voor snellere inferentie, hoe een compacte weergave voor afbeeldingen van verkeers scènes kunnen worden verrijkt met informatie over object instanties en hoe verschillende modaliteiten, zoals afbeeldingen en puntenwolken, bijdragen aan het leren van diepe representaties.

Om voertuigen vroeger dan de meest gebruikte sensoren in autonome auto's te detecteren, introduceert dit proefschrift een benadering van passieve akoestische waarneming. Dit akoestische waarnemingssysteem kan naderende voertuigen achter verdekte hoeken door geluid detecteren voordat deze voertuigen in het gezichtsveld komen. Een onderzoeksvoertuig uitgerust met op het dak gemonteerde microfoons wordt gebruikt om data te verzamelen en dient als demonstratie platform. Uit de data blijkt dat muurreflecties informatie geven over de aanwezigheid en richting van verdekte naderende voertuigen. In testscenario's met een staande test-voertuig bereikt een nieuwe datagestuurde aanpak een nauwkeurigheid van 0.92 op de verborgen voertuig classificatietaak. Vergeleken met een moderne visuele detector, Faster R-CNN, bereikt het akoestische systeem meer dan een seconde voorsprong op dezelfde nauwkeurigheid, wat een cruciale reactietijd oplevert voor de situaties die in dit werk worden bestudeerd. Terwijl het test-voertuig rijdt, laat akoestische detectie bemoedigende resultaten zien, waarbij nog steeds een nauwkeurigheid van 0,84 wordt behaald binnen één omgevingstype. Verder worden faalgevallen in verschillende omgevingen bestudeerd om toekomstige onderzoeksrichtingen te identificeren.

Als een verbetering van de verwerkingssnelheid van waarnemingsalgoritmen, presenteert dit proefschrift een benadering om conventionele beslissingsbomen te combineren met end-to-end leerbare neurale netwerken. Conventionele beslisbomen hebben een aantal gunstige eigenschappen, waaronder een kleine computationele voetafdruk, interpreteerbaarheid en het vermogen om te leren van weinig trainingsdata. Ze missen echter een belangrijke kwaliteit die heeft bijgedragen aan de diepe leerrevolutie: end-to-end trainbaar zijn. Andere werken hebben dit tekort aangepakt, maar ten koste van het verlies van een belangrijk aantrekkelijk kenmerk van beslissingsbomen: het feit dat elk observatie alleen samen met een kleine groep van boomknooppunten wordt geleid. Dit proefschrift presenteert een end-to-end leerschema voor deterministische beslisbomen en beslisbossen. Dankzij een nieuw model en trainingsschema voor expectation-maximization zijn de bomen volledig probabilistisch tijdens het trainen, maar na een uitgloeiproces worden ze deterministisch tijdens het gebruik op test data. Het is gebleken dat de methode op gelijke voet of superieur presteert aan standaard leeralgoritmen voor schuine beslisbomen en bossen, en minder berekeningen vereist dan zijn concurrenten.

Een compactere weergave van de omgeving kan de communicatie tussen softwaremodules stroomlijnen. Instance Stixels, gepresenteerd in dit proefschrift, bieden een compacte ruimtelijke representatie door veel pixels te groeperen tot superpixels en door deze groepen een positie toe te wijzen in de 3D-ruimte. Dit zorgt voor latere planning en algoritmen voor het ontwijken van obstakels met een compacte ruimtelijke lay-out van de scène. State-of-the-art stixel-methoden versmelten dichte stereo-disparity en semantische klasse-informatie, bijvoorbeeld van een convolutioneel neuraal netwerk (CNN), tot een compacte weergave van berijdbare ruimte, obstakels en achtergrond. Ze maken echter geen expliciet onderscheid tussen instanties binnen dezelfde semantische klasse. Dit proefschrift onderzoekt verschillende manieren om single-frame stixels te vergroten met instantie-informatie, die kan worden geëxtraheerd door een CNN uit de RGB-beeldinvoer. Als gevolg hiervan berekent het Instance Stixels-algoritme efficiënt stixels die rekening houden met de grenzen van individuele objecten en vertegenwoordigt instances als gegroepeerde stixels die connectiviteit uitdrukken. De aanpak bereikt sterke segmentatieprestaties en rekenefficiëntie in vergelijking met het combineren van de afzonderlijke outputs van Semantic Stixels en een ultramodern CNN op pixelniveau. De GPU-implementatie van het algoritme kan gemiddeld tot 28 frames per seconde verwerken voor 8 pixel brede stixels op afbeeldingen uit de Cityscapes-dataset van 1792x784 pixels.

Bij het gebruik van meerdere sensormodaliteiten voor perceptie, is het belangrijk om te begrijpen hoe de gegevens van de verschillende modaliteiten zich verhouden om efficiënte representaties voor communicatie tussen sensoren te vinden. Dit proefschrift onderzoekt een veelvoorkomende benadering in state-of-the-art zelf-gesuperviseerd leren van representaties die gebruik maakt van multi-view en multi-modale data door de feature representaties over views en/of modaliteiten uit te lijnen. Een hoofdstuk van dit proefschrift onderzoekt hoe het uitlijnen van representaties de visuele kenmerken beïnvloedt die worden verkregen door cross-view en cross-modaal contrastief leren op afbeeldingen en puntenwolken. Op vijf empirische datasets en op vijf taken worden 108 modellen op basis van vier pretraining variaties getraind en geëvalueerd. De resultaten laten zien dat cross-modale representatie-uitlijning complementaire visuele informatie, zoals kleur en textuur, weggooit en in plaats daarvan de nadruk legt op overbodige diepte-aanwijzingen.

De diepte-aanwijzingen die zijn verkregen uit de pretraining, verbeteren de prestaties van de downstream dieptevoorspelling. Ook leidt cross-modale uitlijning in het algemeen tot robuustere encoders dan pretraining door cross-view uitlijning, met name voor dieptevoorspelling, instantiesegmentatie en objectdetectie.

Ten slotte vat het laatste hoofdstuk samen hoe de afzonderlijke hoofdstukken hebben bijgedragen aan een grotere efficientie van de omgevingswaarneming. Het wordt besproken hoe verschillende benaderingen met elkaar in verband staan en elkaar beïnvloeden, en het worden mogelijke onderzoeksrichtingen voorgesteld.

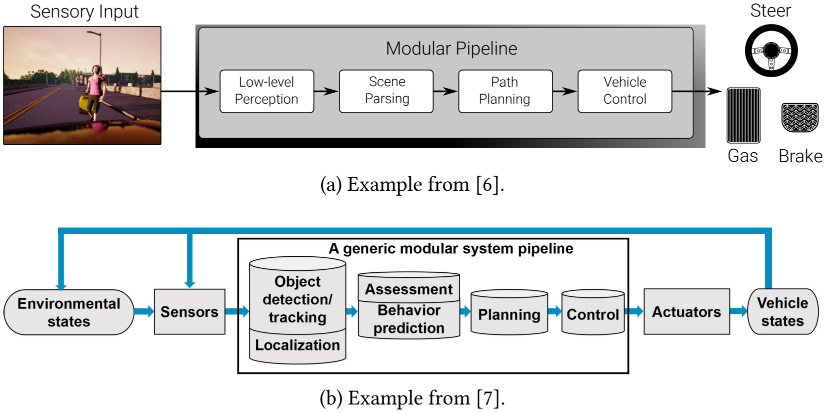
1

Introduction

Many car manufacturers sell vehicles that are equipped with advanced driver assistance systems that allow the driver to hand over control for a short period of time [1]. According to independent market analysts, the share of such vehicles is strongly increasing. As this trend continues, new technology is being developed that increases the autonomy of vehicles. The vehicles will be able to drive autonomously for longer times and in a larger variety of scenarios than currently possible. Such autonomous vehicles have a variety of potential benefits for users and traffic that range from improved mobility for elderly and disabled people, reduced amount of vehicles in inner cities, and of course, increased comfort for the (former) drivers [2]. Self-driving vehicles also have the potential to eliminate human error as a cause of traffic accidents and thereby increase traffic safety.

The World Health Organization (WHO) estimates that about 1.35 million people die each year in traffic-related accidents [3]. A report of the U.S. National Highway Traffic Safety Administration attributes the critical reason to the driver in about 94% of the 5470 crashes that were studied [4]. The advancements in the field of self-driving vehicles raise the hope that soon the risk of human error could be eliminated. Is it reasonable to assume that soon, each year the lives of 1.27 million people will be saved thanks to autonomous vehicles? No. First of all, the WHO also states that the death rate in low-income countries is three times higher than in high-income countries [4]. These countries will likely profit last from a potential increase in road safety due to autonomous vehicles. The traffic composition in low-income countries involves many two-wheelers and important infrastructure is lacking [2]. Second, “critical reason” in the U.S. report means the last failure in a chain of events leading up to a crash, but not the cause of the crash. Still, arguably there are crashes that could have been prevented, e.g. the ones that are related to recognition errors, driving too fast, or sleep. In [5], the authors conclude that automated vehicles may reduce accidents by 48.07% in the U.S. and by 54.24% in India. In any case, over time, the automation of vehicles on public roads will increase, and therefore, it is crucial to make the technology involved as safe as possible and more reliable than human drivers.

The research presented in this thesis aims to improve an essential aspect of the safety of autonomous driving: the perception and representation of the environment. Self-driving vehicles require a detailed understanding of their environment in order to react and avoid



(a) Example from [6].

(b) Example from [7].

Figure 1.1: Two examples of typical modular processing approaches for autonomous vehicles. Sensors and perception algorithms, such as during scene parsing in (a) and object detection in (b), create an internal representation of the environment. This representation is processed by subsequent modules, for instance, for path planning (a) and situation assessment (b).

obstacles as well as to find their path towards their final destination. The perception of the environment is done using a variety of sensors, where arguably cameras are the most well-known. The sensor data needs to be processed by perception algorithms in order to create an internal representation of the environment. Figure 1.1 shows an example of a modular autonomous vehicle processing pipeline, where object detection algorithms process the sensor output and convey the detections and tracks to subsequent modules such as situation assessment and behavior prediction. This position, early on in a modular processing pipeline, leads to its outstanding importance. Any inaccuracies or delays in this stage may deteriorate the overall performance of the system.

The latency of the perception system is of high importance in the prevention of accidents. Any delay in processing the data will reduce the distance the self-driving vehicle has available to avoid a collision. The perception of the environment thus needs to be as efficient as possible, and more efficient perception can be achieved in various ways. *Sensor efficiency* can be improved by capturing more cues from the environment that allow the system to react earlier. This can range, for instance, from higher camera frame rates and denser LiDAR point clouds, to even novel sensing techniques. The time for processing the sensor measurements can be reduced by more *efficient algorithms*. Any speed advantage that is gained here will benefit the entire system as many algorithms depend on the data from the perception system. The interface to those subsequent algorithms is therefore a fundamental link and requires an *efficient representation*. Representations of the environment that compactly store all relevant data in a format that is readily accessible to other algorithms streamline the communication between the different subsystems of a self-driving vehicle. Other ways to improve the efficiency of the perception system which are not discussed in this thesis are, for example, faster processing hardware and collaborative perception approaches via vehicle-to-vehicle communication.

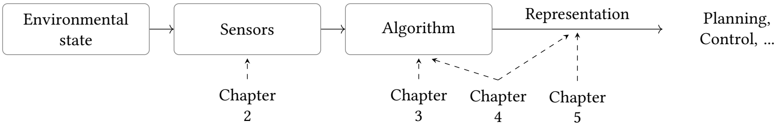


Figure 1.2: An outline of how the chapters in this thesis relate to the perception modules in autonomous driving. Due to its sequential nature, improved efficiency in any of the modules leads to overall efficiency improvements. This eventually leads to faster processing giving the vehicle more time to react.

1.1 Thesis Outline and Contributions

This thesis discusses efficiency improvements for environment perception and representation in autonomous vehicles in three aspects: sensors, algorithms, and representation. Figure 1.2 illustrates their role in environment perception and which chapter discusses each topic. The sensors capture the environment and their low-level data is processed by perception algorithms to create an internal, high-level representation of the scene that can be handled by subsequent modules. In chapter 2, a novel sensor setup based on multiple low-cost microphones demonstrates how the sound of approaching traffic may be used to detect visually occluded vehicles around corners in urban environments. Chapter 3 presents a novel perception algorithm that combines the inference efficiency of decision trees with the feature learning capabilities of deep learning. Instance Stixels, presented in chapter 4, is also a novel algorithm that improves processing speed, but at the same time computes a compact 3D representation of the environment. Chapter 5 discusses how visual representations learned across views and sensing modalities differ and investigates how this affects the performance of visual representations of deep convolutional neural networks on computer vision tasks. The overall conclusions and potential future research directions are presented in chapter 6. The remainder of this chapter will provide an overview of chapters 2-5.

Hearing What You Cannot See: Acoustic Detection Around Corners

Chapter 2 presents how to use multiple cheap vehicle-mounted microphones to capture sound as an auxiliary sensing modality for early detection of approaching vehicles behind blind corners in urban environments. Crucially, the data-driven pattern recognition approach can successfully identify such situations from the acoustic reflection patterns on building walls and provide early warnings before conventional line-of-sight sensing is feasible. While a vehicle should always exit narrow streets cautiously, early warnings would nevertheless reduce the risk of a last-moment emergency brake. Specifically, the contributions of this chapter are threefold:

1. It is demonstrated in real-world outdoor conditions that a vehicle-mounted microphone array can detect the sound of approaching vehicles behind blind corners from reflections on nearby surfaces before line-of-sight detection is feasible.

2. A data-driven detection pipeline is introduced which efficiently addresses this task and outperforms model-driven acoustic signal processing.

3. A new audio-visual dataset in real-world urban environments is presented.

Detecting occluded oncoming traffic is a key advantage for intelligent vehicles, for which passive acoustic sensing is still relatively under-explored. Unlike existing data-driven approaches, visual detectors cannot be used for positional labeling [8] or transfer learning [9], since the targets are visually occluded. Instead, the task is posed as a multi-class classification problem to identify if and from what corner a vehicle is approaching. Direction-of-Arrival estimation provides robust features to classify sound reflection patterns, even without end-to-end feature learning and large amounts of data. The experiments investigate the impact on accuracy and detection time for various conditions, such as different acoustic environments, driving versus static ego-vehicle, and compare to current visual and acoustic baselines. To collect data, a front-facing microphone array was mounted on a research vehicle, which additionally has a front-facing camera. This prototype setup facilitates qualitative and quantitative experimentation of different acoustic perception tasks.

End-to-end Learning of Decision Trees and Forests

Chapter 3 introduces end-to-end learning for deterministic decision trees and forests. Unlike related end-to-end approaches [10], the algorithm presented in this chapter obtains trees with deterministic nodes at test time. This results in efficient inference as each sample is only routed along one unique path of only $\mathcal{O}(\log I)$ out of the I inner nodes in a tree. To reduce variance, multiple trees can also be combined in a decision forest ensemble. Furthermore, an end-to-end trainable tree can provide interpretable classifiers on learned visual features, similar to how decision trees are used in financial or medical expert systems on handcrafted features. In this context, this thesis shows the benefit of regularizing the spatial derivatives of learned features when samples are images or image patches. Differentiable probabilistic nodes at *train time only* enable end-to-end training of a decision tree. This thesis introduces a new probabilistic split criterion that generalizes the long-established information gain [11]. A key aspect of this tree formulation is the introduction of a steepness parameter for the decision [12]. The proposed criterion is asymptotically identical to the information gain in the limit of very steep non-linearities but allows to better model class overlap in the vicinity of a split decision boundary. During training, the probabilistic trees are optimized using the Expectation-Maximization algorithm [13]. Importantly, the steepness parameter is incrementally adjusted in an annealing scheme to make decisions ever more deterministic, and bias the model towards crispness. The proposed procedure also constructs the decision trees level-by-level, hence trees will not grow branches any further than necessary. Compared to initialization with balanced trees, [10] the proposed approach reduces the expected depth of the tree, which further improves efficiency. To summarize, the contributions of this chapter are:

1. An end-to-end learning scheme for deterministic decision trees is proposed that optimizes probabilistic models while training to obtain deterministic trees for testing.

2. A steepness parameter and corresponding annealing scheme is introduced that encourages steeper decisions, and this parameter is used to show that the criterion is asymptotically identical to the established information gain.

3. It is shown how the end-to-end learned features can help interpretability of a model, e.g. by incorporating spatial regularization.

Fast and Compact Image Segmentation using Instance Stixels

Chapter 4 presents Instance Stixels, an efficient stixel-based representation for image segmentation. Stixel algorithms divide an entire image into parallel rectangles of equal width. Such a rectangular group of pixels is called a “stixel”. Instance Stixels, include instance information into the stixels computation, which creates better stixels, and allows grouping the stixels by instance IDs from a single stereo frame. The contributions of this chapter are as follows:

1. A novel approach to include the instance information into stixels is proposed. This approach is compared to two other state-of-the-art approaches. Specifically, it is shown that adding the information into the stixel computation itself results in more accurate instance representations than the alternatives, such as only using it to cluster Semantic Stixels or alternatively assigning Semantic Stixels to instances using pixel-based methods.

2. The trade-off between computation speed and instance segmentation performance is compared for these three variations to showcase the favorable properties of Instance Stixels.

How do cross-view and -modal alignment affect contrastive learning?

Self-supervised training of neural networks on unlabelled data is quickly becoming a key to training robust perception models, especially when limited training data is available. Still, it remains currently unclear how different self-supervised learning strategies on multi-sensor data affect the resulting models. The goal of the work presented in chapter 5 is to understand the effects of cross-view and cross-modal representation alignment on self-supervised contrastive learning in more detail. The question of how redundant and complementary information influence the learned representations, and whether similar improvements as observed for shape-biased over texture-biased networks [14] can be observed as well, are investigated. The empirical study is based on Pri3D [15] which uses both cross-view and cross-modal representation alignment for contrastive learning, as well as their combination. The main contributions of this chapter are two-fold:

1. An assessment is presented on how cross-view and cross-modal representation alignment affect the complementary and redundant information encoded in the learned visual per-pixel representations.

2. An evaluation of the downstream transfer learning performance is conducted to investigate the effects of representations that encode depth rather than texture, and vice versa.

1.2 References

[1] Canalys Newsroom. Canalys: 8% of new cars in Europe sold with level 2 autonomy driving features. https://www.canalys.com/newsroom/canalys-level-2-autonomy-vehicles-europe-q2-2019, September 2019. Accessed on 18-03-2022.

[2] Kareem Othman. Exploring the implications of autonomous vehicles: a comprehensive review. *Innovative Infrastructure Solutions*, 7(2):1–32, 2022.

[3] World Health Organization. *Global Status Report on Road Safety 2018*. Nonserial Publication. World Health Organization, 2019.

[4] S. Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. In *Traffic Safety Facts Crash, Stats. Report No. DOT HS 812 115.* Washington, DC: National Highway Traffic Safety Administration, February 2015.

[5] Ling Wang, Hao Zhong, Wanjing Ma, Mohamed Abdel-Aty, and Juneyoung Park. How many crashes can connected vehicle and automated vehicle technologies prevent: A meta-analysis. *Accident Analysis & Prevention*, 136:105299, 2020.

[6] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020.

[7] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access*, 8:58443–58469, 2020.

[8] Weipeng He, Petr Motlicek, and Jean-Marc Odobez. Deep neural networks for multiple speaker detection and localization. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 74–79. IEEE, 2018.

[9] Chuang Gan, Hang Zhao, Peihao Chen, David Cox, and Antonio Torralba. Self-supervised moving vehicle tracking with stereo sound. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2019.

[10] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulò. Deep neural decision forests. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2015.

[11] J. R. Quinlan. Induction of decision trees. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

[12] A. Montillo, J. Tu, J. Shotton, J. Winn, J.E. Iglesias, D.N. Metaxas, and A. Criminisi. Entanglement and differentiable information gain maximization. In *Decision Forests for Computer Vision and Medical Image Analysis*, chapter 19, pages 273–293. Springer, January 2013.

[13] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Comput.*, 6(2):181–214, March 1994.

[14] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, May 2019.

[15] Ji Hou, Saining Xie, Benjamin Graham, Angela Dai, and Matthias Nießner. Pri3d: Can 3d priors help 2d representation learning? In *Proc. of the IEEE/CVF International Conference on Computer Vision*, pages 5693–5702, 2021.

And now for something completely different.

Monty Python

2

Hearing What You Cannot See: Acoustic Vehicle Detection Around Corners

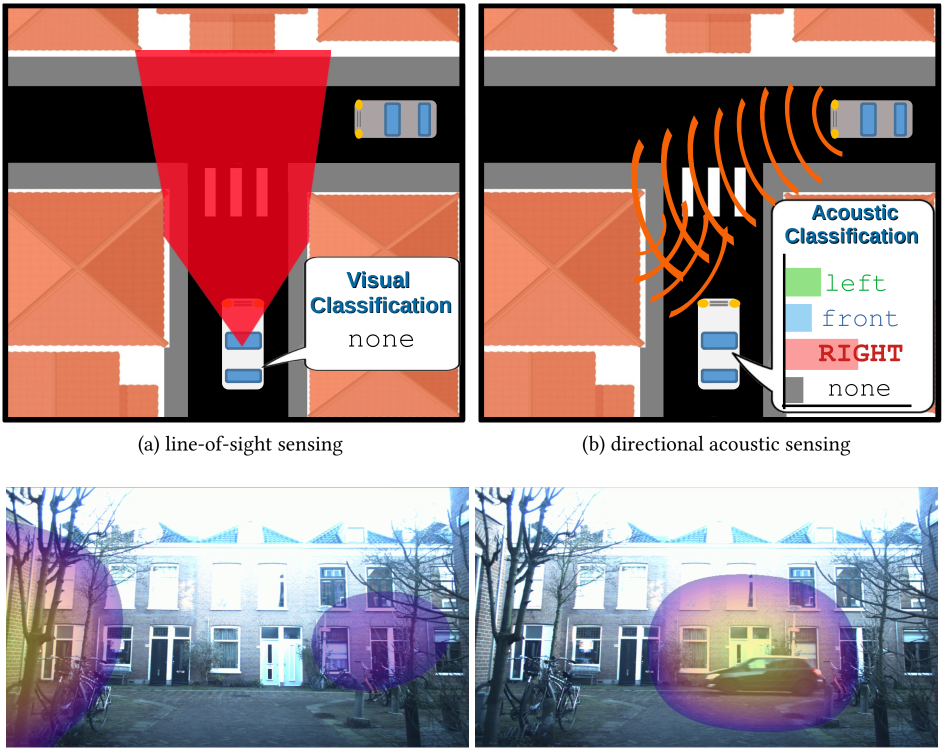
This work proposes to use passive acoustic perception as an additional sensing modality for intelligent vehicles. It is demonstrated that approaching vehicles behind blind corners can be detected by sound before such vehicles enter in line-of-sight. On data collected using a research vehicle equipped with a roof-mounted microphone array, it is shown that wall reflections provide information on the presence and direction of occluded approaching vehicles. A novel method is presented to classify if and from what direction a vehicle is approaching before it is visible, using as input Direction-of-Arrival features that can be efficiently computed from the streaming microphone array data. Since the local geometry around the ego-vehicle affects the perceived patterns, several environment types are systematically studied, and generalization across these environments is investigated. With a static ego-vehicle, an accuracy of 0.92 is achieved on the hidden vehicle classification task. Compared to a state-of-the-art visual detector, Faster R-CNN, the proposed pipeline achieves the same accuracy more than one second ahead, providing crucial reaction time in the studied situations. While the ego-vehicle is driving, acoustic detection still shows positive results by achieving an accuracy of 0.84 within one environment type. Further, failure cases are studied across environments to identify future research directions.

Parts of this chapter have been published as: Yannick Schulz, Avinash Kini Mattar, Thomas M. Hehn, and Julian F.P. Kooij. "Hearing what you cannot see: Acoustic vehicle detection around corners", IEEE Robotics and Automation Letters, 6, 2, p. 2587-2594, 2021, Institute of Electrical and Electronic Engineers.

2.1 Introduction

Highly automated and self-driving vehicles currently rely on three complementary main sensors to identify visible objects, namely camera, lidar, and radar. However, the capabilities of these conventional sensors can be limited in urban environments when sight is obstructed by narrow streets, trees, parked vehicles, and other traffic. Approaching road users may therefore remain undetected by the main sensors, resulting in dangerous situations and last-moment emergency maneuvers [1]. While future wireless vehicle-to-everything communication (V2X) might mitigate this problem, creating a robust omnipresent communication layer is still an open problem [2] and excludes road users without wireless capabilities. Acoustic perception does not rely on line-of-sight and provides a wide range of complementary and important cues on nearby traffic: There are salient sounds with specified meanings, e.g. sirens, car horns, and reverse driving warning beeps of trucks, but also inadvertent sounds from tire-road contact and engine use.

In this work, multiple cheap microphones are used to capture sound as an auxiliary sensing modality for early detection of approaching vehicles behind blind corners in urban environments. Crucially, it is shown that a data-driven pattern recognition approach can successfully identify such situations from the acoustic reflection patterns on building walls and provide early warnings before conventional line-of-sight sensing is able to (see Figure 2.1). While a vehicle should always exit narrow streets cautiously, early warnings would reduce the risk of a last-moment emergency brake.



(a) line-of-sight sensing

(b) directional acoustic sensing

(c) sound localization with a vehicle-mounted microphone array detects the wall reflection of an approaching vehicle behind a corner before it appears

Figure 2.1: When an intelligent vehicle approaches a narrow urban intersection, (a) traditional line-of-sight sensors cannot detect approaching traffic due to occlusion, while (b) acoustic cues can provide early warnings. (c) Real-time beamforming reveals reflections of the acoustic signal on the walls, especially salient on the side opposing the approaching vehicle. Learning to recognize these patterns from data enables detection before line-of-sight.

2.2 Related work

The focus of this work lies on passive acoustic sensing in mobile robotics [3–5] to detect and localize nearby sounds, which is distinct from active acoustic sensing using self-generated sound signals, e.g. [6]. While mobile robotic platforms in outdoor environments may suffer from vibrations and wind, various works have demonstrated the detection and localization of salient sounds on moving drones [7] and wheeled platforms [8, 9].

Although acoustic cues are known to be crucial for traffic awareness by pedestrians and cyclists [10], only a few works have explored passive acoustic sensing as a sensor for Intelligent Vehicles (IVs). [9, 11, 12] focus on detection and tracking in direct line-of-sight. [13, 14] address detection behind corners from a static observer. [13] only shows experiments without directional estimation. [14] tries to accurately model wave refractions, but experiments in an artificial lab setup show limited success. Both [13, 14] rely on strong modeling assumptions, ignoring that other informative patterns could be present in the acoustic data. Acoustic traffic perception is furthermore used for road-side traffic monitoring, e.g. to count vehicles and estimate traffic density [15, 16]. While the increase in Electric Vehicles (EVs) may reduce overall traffic noise, [17] shows that at 20-30km/h the noise levels for EV and internal combustion vehicles are already similar due to tire-road contact. [18] finds that at lower speeds the difference is only about 4-5 dB, though many EVs also suffer from audible narrow peaks in the spectrum. As low-speed EVs can impact the acoustic awareness of humans too [10], legal minimum sound requirements for EVs are being proposed [19, 20].

Direction-of-Arrival estimation is a key task for sound source localization, and over the past decades, many algorithms have been proposed [3, 21], such as the Steered-Response Power Phase Transform (SRP-PHAT) [22] which is well-suited for reverberant environments with possibly distant unknown sound sources. Still, in urban settings nearby walls, corners, and surfaces distort sound signals through reflections and diffraction [23]. Accounting for such distortions has shown to improve localization [8, 24], but only in controlled indoor environments where detailed knowledge of the surrounding geometry is available.

Recently, data-driven methods have shown promising results in challenging real-world conditions for various acoustic tasks. For instance, learned sound models assist monaural source separation [25] and source localization from direction-dependent attenuations by fixed structures [26]. Increasingly, deep learning is used for audio classification [27, 28], and localization [29] of sources in line-of-sight, in which case visual detectors can replace manual labeling [30, 31]. Analogous to this thesis, [32] presents the first deep learning method for sensing around corners but with automotive radar. Thus, while the effect of occlusions on sensor measurements is difficult to model [14], data-driven approaches appear to be a good alternative.

This thesis provides the following contributions: First, it is demonstrated in real-world outdoor conditions that a vehicle-mounted microphone array can detect the sound of approaching vehicles behind blind corners from reflections on nearby surfaces before line-of-sight detection is feasible. This is a key advantage for IVs, where passive acoustic sensing is still relatively under-explored. The presented experiments investigate the impact on accuracy and detection time for various conditions, such as different acoustic environments, driving versus static ego-vehicle, and compare to current visual and acoustic baselines.

Second, a data-driven detection pipeline to efficiently addresses this task and show that it outperforms model-driven acoustic signal processing. Unlike existing data-driven approaches, visual detectors cannot be used for positional labeling [30] or transfer learning [31], since the targets are visually occluded. Instead, the task is cast as a multi-class classification problem to identify if and from what corner a vehicle is approaching. It is demonstrated that Direction-of-Arrival estimation can provide robust features to classify sound reflection patterns, even without end-to-end feature learning and large amounts of data.

Third, for the experiments, a new audio-visual dataset was collected in real-world urban environments.[1] To collect data, a front-facing microphone array was mounted on

[1] Code & data:
https://github.com/tudelft-iv/occluded_vehicle_acoustic_detection

a research vehicle, which additionally has a front-facing camera. This prototype setup facilitates qualitative and quantitative experimentation of different acoustic perception tasks.

2.3 Approach

Ideally, an ego-vehicle driving through an area with occluding structures is able to early predict *if* and from *where* another vehicle is approaching, even if it is from behind a blind corner as illustrated in Figure 2.1. Concretely, this work aims to distinguish three situations as early as possible using ego-vehicle sensors only:

- an occluded vehicle approaches from behind a corner on the *left*, and only moves into view last moment when the ego-vehicle is about to reach the junction,
- same, but vehicle approaches behind a *right* corner,
- no vehicle is approaching.

This work proposes to consider this task an online classification problem. As the ego-vehicle approaches a blind corner, the acoustic measurements made over short time spans should be assigned to one in a set of four classes, $C = \{$`left`, `front`, `right`, `none`$\}$, where `left`/`right` indicates a still occluded (i.e. not yet in direct line-of-sight) approaching vehicle behind a corner on the left/right, `front` that the vehicle is already in direct line-of-sight, and `none` that no vehicle is approaching.

In Section 2.3.1, two line-of-sight baseline approaches are considered for detecting vehicles. Section 2.3.2 then elaborates on the proposed extension to acoustic non-line-of-sight detection. Section 2.3.3 provides details of the vehicle's novel acoustic sensor setup used for data collection.

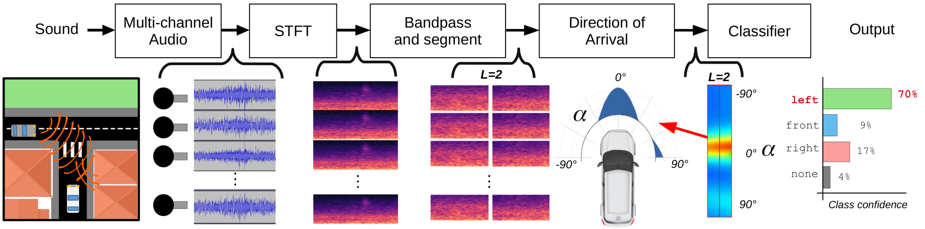


Figure 2.2: Overview of the proposed acoustic detection pipeline, see Section 2.3.2 for an explanation of the steps.

2.3.1 Line-of-sight detection

First, it is discussed how the task would be addressed with line-of-sight vehicle detection using either conventional cameras, or using past work on acoustic vehicle detection.

Visual detection baseline Cameras are currently one of the de-facto choices for detecting vehicles and other objects within line-of-sight. Data-driven Convolutional Neural Networks have proven to be highly effective on images. However, visual detection can only detect vehicles that are already (partially) visible, and thus only distinguishes between `front` and `none`. To demonstrate this, Faster R-CNN [33] is used, a state-of-the-art visual object detector, on the ego-vehicle's front-facing camera as a visual baseline.

Acoustic detection baseline Next, an ego-vehicle equipped with an array of M microphones is considered. As limited training data hinders learning features (unlike [30, 31]), beamforming is used to estimate the Direction-of-Arrival (DoA) of tire and engine sounds originating from the approaching vehicle. DoA estimation directly identifies the presence and direction of such sound sources and has been shown to work robustly in unoccluded conditions [9, 11]. Since sounds can be heard around corners, and low frequencies diffract ("bend") around corners [23], one might wonder: Does the DoA of the sound of an occluded vehicle correctly identify from where the vehicle is approaching? To test this hypothesis for the target real-world application, the second baseline follows [9, 11] and directly uses the most salient DoA angle estimate.

Specifically, the implementation uses the Steered-Response Power-Phase Transform (SRP-PHAT) [22] for DoA estimation. SRP-PHAT relates the spatial layout of sets of microphone pairs and the temporal offsets of the corresponding audio signals to their relative distance to the sound source. To apply SRP-PHAT on M continuous synchronized signals, only the most recent δt seconds are processed. On each signal, a Short-Time Fourier Transform (STFT) is computed with a Hann windowing function, and a frequency bandpass for the $[f_{min}, f_{max}]$ Hz range. Using the generalized cross-correlation of the M STFTs, SRP-PHAT computes the DoA energy $r(\alpha)$ for any given azimuth angle α around the vehicle. Here $\alpha = -90^\circ/0^\circ/+90^\circ$ indicates an angle toward the left/front/right of the vehicle respectively. If the hypothesis holds that the overall salient sound direction $\alpha_{\max} = \arg\max r(\alpha)$ remains intact due to diffraction, one only needs to determine if $\alpha_{\max}$ is beyond some sufficient threshold α_{th}. The baseline thus assigns class `left` if $\alpha_{\max} < -\alpha_{\text{th}}$, `front` if $-\alpha_{\text{th}} \leq \alpha_{\max} \leq +\alpha_{\text{th}}$, and `right` if $\alpha_{\max} > +\alpha_{\text{th}}$. This baseline is evaluated on the easier task of only separating these three classes, and ignoring the `none` class.

2.3.2 Non-line-of-sight acoustic detection

In contrast to line-of-sight detection, DoA estimation alone is unsuited for occluded vehicle detection (and confirm this in Section 2.4.3). Salient sounds produce sound wave reflections on surfaces, such as walls (see Figure 2.1c), thus the DoA does not indicate the actual location of the source. Modeling the sound propagation [8] while driving through uncontrolled outdoor environments is challenging, especially as accurate models of the local geometry are missing. Therefore, a data-driven approach is employed, and the *full energy distribution* from SRP-PHAT is treated as robust features for a classifier that capture all reflections.

An overview of the proposed processing pipeline is shown in Figure 2.2. M STFTs are created, using a temporal window of δt seconds, Hann windowing function and a frequency bandpass of $[f_{min}, f_{max}]$ Hz. Notably, no noise filtering or suppression is applied. To capture temporal changes in the reflection pattern, the STFTs are split along the temporal dimension into L non-overlapping segments. For each segment, the DoA energy is computed at multiple azimuth angles α in front of the vehicle. The azimuth range $[-90^\circ, +90^\circ]$ is divided into B equal bins $\alpha_1, \cdots, \alpha_B$. From the original M signals, thus L response vectors $\boldsymbol{r}_l = [r_l(\alpha_1), \cdots, r_l(\alpha_B)]^\top$ are obtained. Finally, these are concatenated to a

$(L \times B)$-dimensional feature vector $\boldsymbol{x} = [\boldsymbol{r}_1, \cdots, \boldsymbol{r}_L]^\top$, for which a Support Vector Machine is trained to predict C. Note that increasing the temporal resolution by having more segments L comes at the trade-off of an increased final feature vector size and reduced DoA estimation quality due to shorter time windows.

2.3.3 Acoustic perception research vehicle

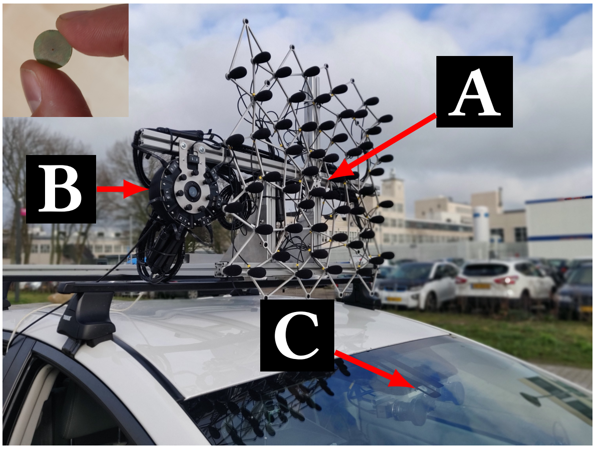
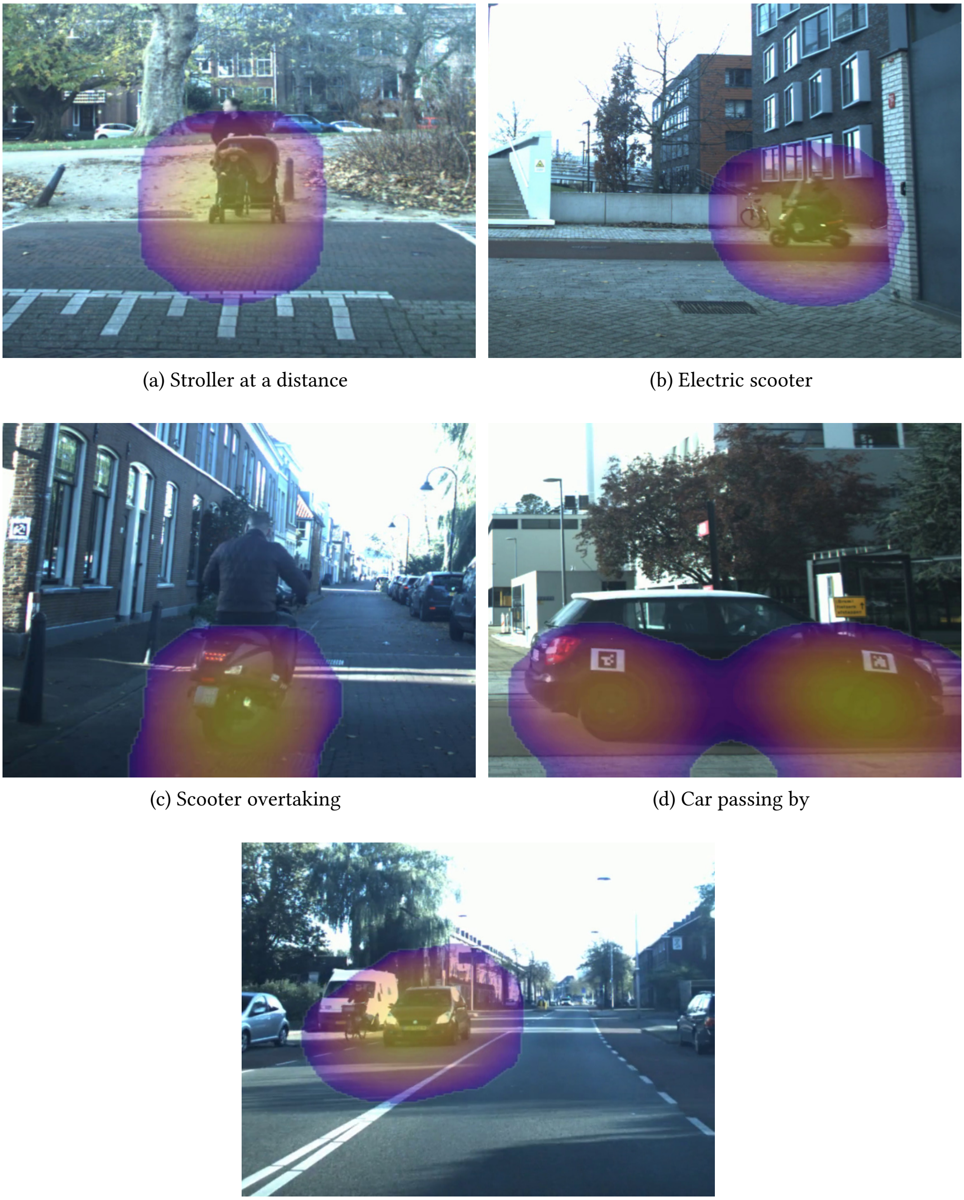


Figure 2.3: Sensor setup of the test vehicle. A: Center of the 56 MEMS acoustic array. B: signal processing unit. C: front camera behind windscreen. Inset: the diameter of a single MEMS microphone is only 12mm.

To collect real-world data and demonstrate non-line-of-sight detection, a custom microphone array was mounted on the roof rack of a research vehicle [34], a hybrid electric Toyota Prius. The microphone array hardware consists of 56 ADMP441 MEMS microphones, and supports data acquisition at 48 kHz sample rate, 24 bits resolution, and synchronous sampling. It was bought from *CAE Software & Systems GmbH* with a metal frame. On this $0.8m \times 0.7m$ frame the microphones are distributed semi-randomly while the microphone density remains homogeneous. The general purpose layout was designed by the company through stochastic optimization to have a large variance in inter-microphone distances and serve a wide range of acoustic imaging tasks. The vehicle is also equipped with a front-facing camera for data collection and processing. The center of the microphone array is about 1.78m above the ground, 0.54m above, and 0.50m behind the used front camera, see Figure 2.3. As depicted in the Figure's inset, the microphones themselves are only 12mm wide. They cost about US$1 each.

A signal processing unit receives the analog microphone signals and sends the data over Ethernet to a PC running the Robot Operating System (ROS). Using ROS, the synchronized microphone signals are collected together with other vehicle sensor data. Processing is done in python, using *pyroomacoustics* [21] for acoustic feature extraction, and *scikit-learn* [35] for classifier training.

This setup is not intended as a production prototype, but provides research benefits: The 2D planar arrangement provides both horizontal and vertical high-resolution DoA responses, which can be overlaid as 2D heatmaps [36] on the front camera image to visually study the salient sources (Section 2.4.1). By testing subsets of microphones, it is possible to assess the impact of the number of microphones and their relative placement (Section 2.4.7). In the future, the array should only use a few microphones at various locations around the vehicle.



(a) Stroller at a distance

(b) Electric scooter

(c) Scooter overtaking

(d) Car passing by

(e) Oncoming car

Figure 2.4: Qualitative examples of 2D Direction-of-Arrival estimation overlaid on the camera image (zoomed). (a): Stroller wheels are picked up even at a distance. (b), (c): Both conventional and more quiet electric scooters are detected. (d): The loudest sound of a passing vehicle is typically the road contact of the individual tires. (e): Even when the ego-vehicle drives at ~ 30 km/h, oncoming moving vehicles are still registered as salient sound sources.

2.4 Experiments

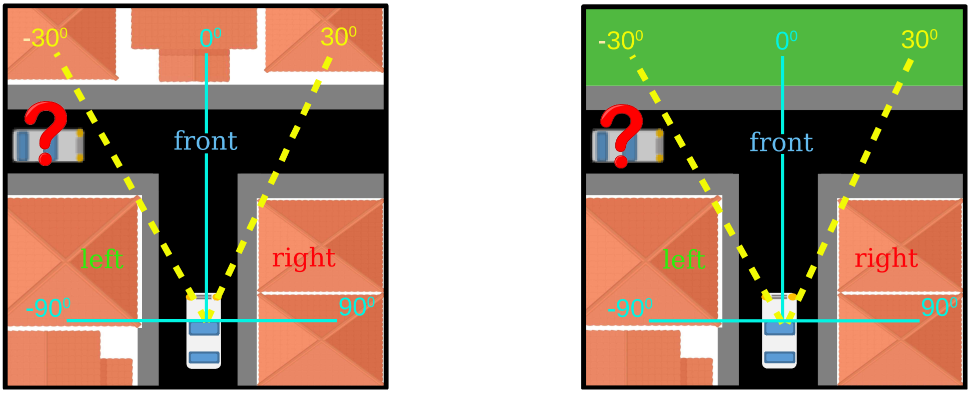
To validate the method, a novel dataset is created using the acoustic research vehicle in real-world urban environments. The quality of acoustic beamforming is demonstrated in such conditions before turning to the main experiments.

2.4.1 Line-of-sight localization – qualitative results

As explained in Section 2.3.3, the heatmaps of the 2D DoA results can be overlaid with the camera images. Figure 2.4 shows some interesting qualitative findings in real urban conditions. The examples highlight that beamforming can indeed pick up various important acoustic events for autonomous driving in line-of-sight, such as the presence of vehicles and some vulnerable road users (e.g. strollers). Remarkably, even electric scooters and oncoming traffic *while the ego-vehicle is driving* are recognized as salient sound sources. A key observation from Figure 2.1c is that sounds originating behind corners reflect in particular patterns on nearby walls. Overall, these results show the feasibility of acoustic detection of (occluded) traffic.

2.4.2 Non-line-of-sight dataset and evaluation metrics

The quantitative experiments are designed to separately control and study various factors that could influence acoustic perception. Multiple recordings of the situations explained in Section 2.3 are collected at five T-junction locations with blind corners in the inner city of Delft. The locations are categorized into two types of walled acoustical environments, namely types A and B (see Figure 2.5). At these locations, common background noise, such as construction sites and other traffic, was present at various volumes. For safety and control, no recordings were made in the presence of other motorized traffic on the roads at the target junction.



(a) **Type A**: completely walled

(b) **Type B**: walled exit

Figure 2.5: Schematics of considered environment types. The ego-vehicle approaches the junction from the bottom. Another vehicle might approach behind the left or right blind corner. Dashed lines indicate the camera FoV.

The recordings can further be divided into *Static* data, made while is the ego-vehicle in front of the junction but not moving, and more challenging *Dynamic* data where the ego-vehicle reaches the junction at ~15 km/h (see the supplementary video). Static data is easily collected, and ensures that the main source of variance is the approaching vehicle's changing position.

For the static case, the ego-vehicle was positioned such that the building corners are still visible in the camera and occlude the view onto the intersecting road (on average a distance of ~7-10m from the intersection). Different types of passing vehicles were recorded, although in most recordings the approaching vehicle was a Škoda Fabia 1.2 TSI (2010) driven by one of the authors. For the Dynamic case, coordinated recordings with the Škoda Fabia were conducted to ensure that encounters were relevant and executed in a safe manner. Situations with `left`/`right`/`none` approaching vehicles were performed in arbitrary order to prevent undesirable correlation of background noise to some class labels. In ~70% of the total Dynamic recordings and ~19.5% of the total Static recordings, the ego-vehicle's noisy internal combustion engine was running to charge its battery.

ID	`left`	`front`	`right`	`none`	Sum
SA1 / DA1	14 / 19	30 / 38	16 / 19	30 / 37	90/113
SA2 / DA2	22 / 7	41 / 15	19 / 8	49 / 13	131/ 43
SB1 / DB1	17 / 18	41 / 36	24 / 18	32 / 35	114/107
SB2 / DB2	28 / 10	55 / 22	27 / 12	43 / 22	153/ 66
SB3 / DB3	22 / 19	45 / 38	23 / 19	45 / 36	135/112
SAB / DAB	103/ 73	212/149	109/ 76	199/143	*623/441*

Table 2.1: Samples per subset. In the ID, S/D indicates Static/Dynamic ego-vehicle, A/B the environment type (see figure 2.5).

Sample extraction For each Static recording with an approaching target vehicle, the time t_0 is manually annotated as the moment when the approaching vehicle enters direct line-of-sight. Since the quality of the t_0 estimate is bounded by the ego-vehicle's camera frame rate (10 Hz), the last image *before* the incoming vehicle is visible serves as a conservative estimate of t_0. Thus, there is no line-of-sight at $t \leq t_0$. At $t > t_0$ the vehicle is considered visible, even though it might only be a fraction of the body. For the Dynamic data, this annotation is not feasible as the approaching car may be in direct line-of-sight, yet outside the limited field-of-view of the front-facing camera as the ego-vehicle has advanced onto the intersection. Thus, annotating t_0 based on the camera images is not representative of line-of-sight detection. To still compare the results across locations, the time τ_0, the moment when the ego-vehicle is at the same position as in the corresponding Static recordings, is manually annotated. All Dynamic recordings are aligned to that time as it represents the moment where the ego-vehicle should make a classification decision, irrespective if an approaching vehicle is about to enter line-of-sight or still further away.

From the recordings, short δt = 1s audio samples are extracted. Let t_e, the end of the time window $[t_e - 1\text{s}, t_e]$, denote a sample's time stamp at which a prediction could be made. For Static `left` and `right` recordings, samples with the corresponding class label are extracted at $t_e = t_0$. For Dynamic recordings, `left` and `right` samples are extracted at

$t_e = \tau_0 + 0.5\text{s}$. This ensures that during the 1s window the ego-vehicle is on average close to its position in the Static recordings. In both types of recordings, `front` samples are extracted 1.5s after the `left`/`right` samples, e.g. $t_e = t_0 + 1.5\text{s}$. Class `none` samples were from recordings with no approaching vehicles. Table 2.1 lists statistics of the extracted samples at each recording location.

Data augmentation Table 2.1 shows that the data acquisition scheme produced imbalanced class ratios, with about half the samples for `left`, `right` compared to `front`, `none`. The experiments therefore explore *data augmentation*. By exploiting the symmetry of the angular DoA bins, augmentation will double the `right` and `left` class samples by reversing the azimuth bin order in all $\boldsymbol{r}_l$, resulting in new features for the opposite label, i.e. as if additional data was collected at mirrored locations. Augmentation is a training strategy only, and thus not applied to test data to keep results comparable, and distinct for `left` and `right`.

Metrics The overall accuracy and the per-class Jaccard index (a.k.a. Intersection-over-Union) are reported, where the latter serves as a robust measure of one-vs-all performance. First, for each class c the True Positives/Negatives (TP_c/TN_c), and False Positives/Negatives (FP_c/FN_c) are computed, treating target class c as positive and the other three classes jointly as negative. Given the total number of test samples N, the overall accuracy is then $\left(\sum_{c \in C} TP_c\right)/N$ and the per-class Jaccard index is $J_c = TP_c/(TP_c + FP_c + FN_c)$.

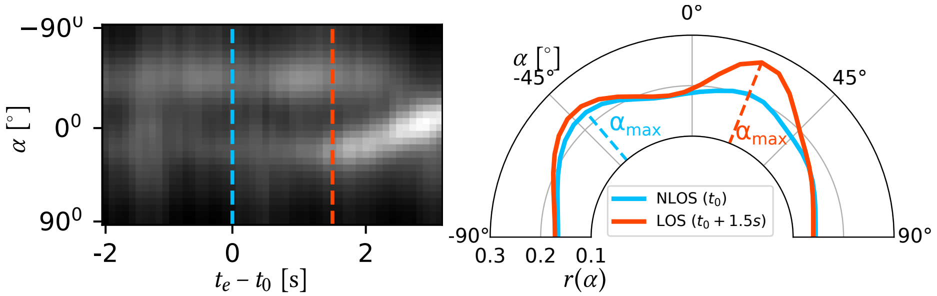
Run	Accuracy	$J_{\texttt{left}}$	$J_{\texttt{front}}$	$J_{\texttt{right}}$	$J_{\texttt{none}}$
* *(reference)*	**0.92**	0.79	0.89	**0.87**	0.83
* wo. data augment.	**0.92**	0.75	0.91	0.78	0.83
* w. $\delta t = 0.5\text{s}$	0.91	0.75	0.89	**0.87**	0.82
* w. $L = 1$	0.86	0.64	0.87	0.73	0.79
* w. $L = 3$	**0.92**	0.74	0.92	0.82	0.81
* w. $L = 4$	0.90	0.72	0.90	0.77	0.83
* w. SVM $\lambda = 0.1$	0.91	0.78	0.89	0.81	0.82
* w. SVM $\lambda = 10$	0.91	**0.81**	0.86	0.84	0.83
DoA-only [9, 11]	0.64	0.11	0.83	0.28	-
Faster R-CNN [37]	0.60	0.00	**0.99**	0.00	**0.98**

Table 2.2: Baseline comparison and hyperparameter study w.r.t. the reference configuration: SVM $\lambda = 1$, $\delta t = 1$, $L = 2$, data augmentation. Results on Static data. * denotes the proposed pipeline.

2.4.3 Training and impact of classifier and features

First, the overall system performance and hyperparameters are evaluated on all Static data from both type A and B locations (i.e. subset ID 'SAB') using 5-fold cross-validation. The folds are fixed once for all experiments, with the training samples of each class equally distributed among folds.

The frequency range is fixed to $f_{min} = 50\text{Hz}, f_{max} = 1500\text{Hz}$, and the number of azimuth bins to $B = 30$ (Section 2.3.2). For efficiency and robustness, a linear Support Vector



(a) DoA energy over time

(b) DoA relative to ego-vehicle

Figure 2.6: DoA energy over time for the recording shown in Figure 2.1c. When the approaching vehicle is not in line-of-sight (NLOS), e.g. at t_0, the main peak is a reflection on the wall ($\alpha_{max} < -30^\circ$) opposite of that vehicle.

Machine (SVM) is used with $l2$–regularization weighted by hyperparameter λ. Other hyperparameters to explore include the sample length $\delta t \in \{0.5\text{s}, 1\text{s}\}$, the segment count $L \in \{1,2,3,4\}$, and using/not using data augmentation.

The final choice and reference is the SVM with $\lambda = 1$, $\delta t = 1$s, $L = 2$, and data augmentation. Table 2.2 shows the results for changing these parameter choices. The overall accuracy for all these hyperparameters choices is mostly similar, though per-class performance does differ. The reference achieves top accuracy, while also performing well on both `left` and `right`. The hyperparameters are kept for all following experiments.

The table also shows the results of the DoA-only baseline explained in Section 2.3.1 using $\alpha_{\text{th}} = 50^\circ$, which was found through a grid search in the range $[0^\circ, 90^\circ]$. As expected, the DoA-only baseline [9, 11] shows weak performance for all metrics. While the sound source is occluded, the most salient sound direction does not represent its origin, but its reflection on the opposite wall (see Figure 2.1). The temporal evolution of the full DoA energy for a car approaching from the `right` is shown in Figure 2.6. When it is still occluded at t_0, there are multiple peaks and the most salient one is a reflection on the left ($\alpha_{max} \approx -40^\circ$). Only once the car is in line-of-sight ($t_0 + 1.5$s) the main mode clearly represents its true direction ($\alpha_{max} \approx +25^\circ$). The left and right image in Figure 2.1c also show such peaks at t_0 and $t_0 + 1.5$s, respectively.

The bottom row of the table shows the visual baseline, a Faster R-CNN R50-C4 model trained on the COCO dataset [37]. To avoid false positive detections, the score threshold is set to 75% and additionally a bounding box height of 100 pixels is required to ignore cars far away in the background, which were not of interest. Generally, this threshold is already exceeded once the hood of the approaching car is visible. While performing well on `front` and `none`, this visual baseline shows poor overall accuracy as it is physically incapable of classifying `left` and `right`.

2.4.4 Detection time before appearance

Ultimately, the goal is to know whether the acoustic method can detect approaching vehicles earlier than the state-of-the-art visual baseline. For this purpose, their online performance is compared next.

The static recordings are divided into a fixed training (328 recordings) and test (83 recordings) split, stratified to adequately represent labels and locations. The training was conducted as in Section 2.4.3 with `left` and `right` samples extracted at $t_e = t_0$. The visual baseline is evaluated on every camera frame (10 Hz). The detector is evaluated on a sliding window of 1s across the 83 test recordings. To account for the transition period when the car may still be partly occluded, `front` predictions by both methods are accepted as correct starting at $t = t_0$. For recordings of classes `left` and `right`, these classes are accepted until $t = t_0 + 1.5$s, allowing for temporal overlap with `front`.

Figure 2.7 illustrates the accuracy on the test recordings for different evaluation times t_e. The overlap region is indicated by the gray area after $t_e = t_0$ and its beginning thus marks when a car enters the field of view. At $t_e = t_0$, just before entering the view of the camera, the approaching car can be detected with 0.94 accuracy by the proposed method. This accuracy is achieved more than one second ahead of the visual baseline, showing that acoustic detection gives the ego-vehicle additional reaction time. After 1.5s a decreasing accuracy is reported since the leaving vehicle is not annotated and only `front` predictions are considered true positives. The acoustic detector sometimes still predicts `left`, or `right` once the car crossed over. The Faster R-CNN accuracy also decreases: after 2s the car is often completely occluded again.

Figure 2.8 shows the per-class probabilities as a function of extraction time t_e on the test set, separated by recording situations. The SVM class probabilities are obtained with the method in [38]. The probabilities for `left` show that on average the model initially predicts that no car is approaching. Towards t_0, the `none` class becomes less likely and the model increasingly favors the correct `left` class. A short time after t_0, the prediction flips to the `front` class, and eventually switches to `right` as the car leaves line-of-sight. Similar (mirrored) behavior is observed for vehicles approaching from the right. The probabilities of `left`/`right` rise until the approaching vehicle is almost in line-of-sight, which corresponds to the extraction time of the training samples. The `none` class is constantly predicted as likeliest when no vehicle is approaching. Overall, the prediction matches the events of the recorded situations remarkably well.

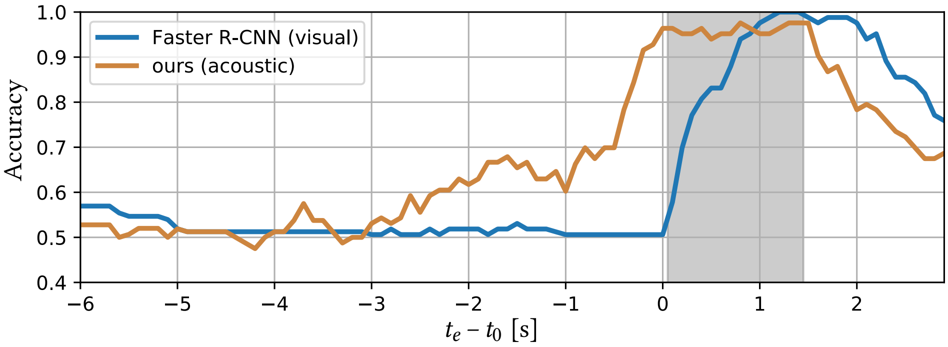
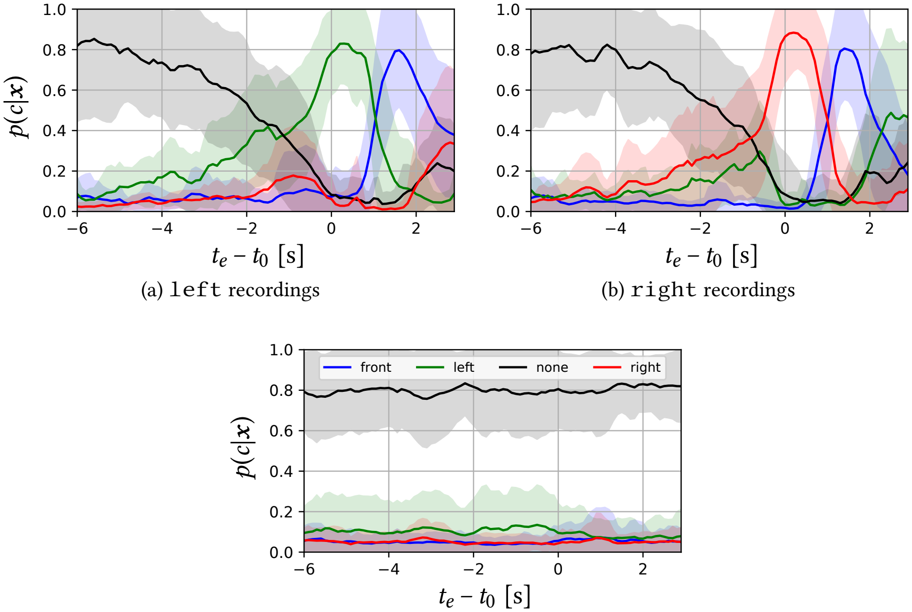


Figure 2.7: Accuracy over test time t_e of the acoustic approach and the visual baseline on 83 Static recordings. Gray region indicates the other vehicle is half-occluded and two labels, `front` and either `left` or `right`, are considered correct.



(a) `left` recordings

(b) `right` recordings

(c) `none` recordings

Figure 2.8: Mean and standard deviation of predicted class probabilities at different times t_e on test set recordings of the Static data (blue is `front`, green is `left`, red is `right`, and black is `none`). Each figure shows recordings of a different situation. The approaching vehicle appears in view just after $t_e - t_0 = 0$.

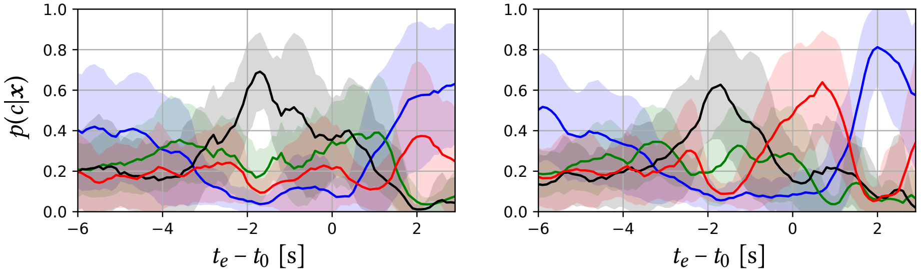
Subset	Accuracy	$J_{\texttt{left}}$	$J_{\texttt{front}}$	$J_{\texttt{right}}$	$J_{\texttt{none}}$
DAB	0.76	0.41	0.80	0.44	0.65
DA	0.84	0.66	0.85	0.64	0.72
DB	0.75	0.33	0.81	0.42	0.64

Table 2.3: Cross-validation results per environment on Dynamic data.

2.4.5 Impact of the moving ego-vehicle

Next, the classifier is evaluated by cross-validation per environment subset, as well as on the full Dynamic data. As for the Static data, 5-fold cross-validation is applied to each subset, keeping the class distribution balanced across folds.

Table 2.3 lists the corresponding metrics for each subset. On the full Dynamic data (DAB), the accuracy indicates decent performance, but the metrics for `left` and `right` classes are much worse compared to the Static results in Table 2.2. Separating subsets DA and DB reveals that the performance is highly dependent on the environment type. In fact, even with limited training data and large data variance from a driving ego-vehicle, decent classification performance is obtained in type A environments, and it is noticeable that low `left` and `right` performance mainly results from type B environments. It is possible that the more confined type A environments reflect more target sounds and are



(a) `left` recordings

(b) `right` recordings

Figure 2.9: Mean and standard deviation of predicted class probabilities at different times t_e on `left` and `right` test set recordings of the Dynamic data. The ego-vehicle reached the location of training data when $t_e - \tau_0 = 0.5$s.

better shielded from potential noise sources.

The temporal behavior of the method on Dynamic data is also analyzed. Unfortunately, a fair comparison with a visual baseline is not possible: the ego-vehicle often reaches the intersection early, and the approaching vehicle is within line-of-sight but still outside the front-facing camera's field of view (cf. τ_0 extraction in Section 2.4.2). Yet, the evolution of the predicted probabilities can be compared to those on the Static data in Section 2.4.4. Figure 2.9 illustrates the average predicted probabilities over 59 Dynamic test set recordings from all locations, after training on samples from the remaining 233 recordings. The classifier on average correctly predicts `right` samples (Figure 2.9b), between $t_e = \tau_0$ to $t_e = \tau_0 + 0.5$s. Of the `left` recordings at these times, many are falsely predicted as `none`, only few are confused with `right`. Furthermore, the changing ego-perspective of the vehicle results in alternating DoA-energy directions and thus class predictions, compared to the Static results in Figure 2.8. This indicates that it might help to include the ego-vehicle's relative position as an additional feature, and obtain more varied training data to cover the positional variations.

2.4.6 Generalization across acoustic environments

In the following, it is studied how the performance is affected when the classifier is trained on all samples from one environment type and evaluated on all samples of the other type. In Table 2.4, combinations of training and test sets are listed. Compared to the results for Static and Dynamic data (see Tables 2.2 and 2.3), the reported results in the table show a general trend: If the classifier is trained on one environment and tested on the other, it performs worse than when samples of the same location are used. In particular, the classifier trained on SB and tested on SA is not able to correctly classify samples of `left` and `right` while inverse training and testing perform much better. On the Dynamic data, such pronounced effects are not visible, but overall the accuracy decreases compared to the Static data. In summary, the reflection patterns vary from one environment to another, yet at some locations the patterns appear more distinct and robust than those at others.

Training	Test	Accuracy	$J_{\texttt{left}}$	$J_{\texttt{front}}$	$J_{\texttt{right}}$	J_{none}
SB	SA	0.66	0.03	0.66	0.03	0.62
SA	SB	0.79	0.42	0.82	0.61	0.67
DB	DA	0.53	0.16	0.70	0.25	0.16
DA	DB	0.56	0.21	0.50	0.29	0.46

Table 2.4: Generalization across locations and environments.

2.4.7 Microphone array configuration

An array with 56 microphones enables evaluation of different spatial configurations with $M < 56$. For various subsets of M microphones, 100 out of $\binom{56}{M}$ possible microphone configurations are randomly sampled, and cross-validated on the Static data. Interestingly, the best configuration with $M = 7$ already achieves similar accuracy as with $M = 56$. With $M = 2/3$ the accuracy is already 0.82/0.89, but with worse performance on `left` and `right`. The large variance between samples highlights the importance of a thorough search of spatial configurations. Reducing M also leads to faster inference time, specifically 0.24/0.14/0.04s for $M = 56/28/14$ using an unoptimized implementation.

2.5 Conclusions

A vehicle-mounted microphone array can be used to acoustically detect approaching vehicles behind blind corners from their wall reflections. In the experimental setup, the proposed method achieved an accuracy of 0.92 on the 4-class hidden car classification task for a static ego-vehicle, and up to 0.84 in some environments while driving. An approaching vehicle was detected with the same accuracy as the visual baseline already more than one second ahead, a crucial advantage in such critical situations.

While these initial findings are encouraging, the results have several limitations. The experiments included only a few locations and a few different oncoming vehicles, and while the proposed method performed well in one environment, it had difficulties in the other, and did not perform reliably in unseen test environments. To expand the applicability, more representative data is needed to capture a broad variety of environments, vehicle positions, and velocities, and the presence of multiple sound sources. Rather than generalizing across environments, additional input from map data or other sensor measurements could help to discriminate acoustic environments and to classify the reflection patterns accordingly. More data also enables end-to-end learning of low-level features, potentially capturing cues that the DoA-based approach currently ignores (e.g. Doppler, sound volume), and perform multi-source detection and classification in one pass [30]. Ideally, a suitable self-supervised learning scheme is developed [31], though a key challenge is that actual occluded sources cannot immediately be visually detected.

2.6 References

[1] Christoph G Keller, Thao Dang, Hans Fritz, Armin Joos, Clemens Rabe, and Dariu M Gavrila. Active pedestrian safety by automatic braking and evasive steering. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1292–1304, 2011.

[2] Zachary MacHardy, Ashiq Khan, Kazuaki Obana, and Shigeru Iwashina. V2X access technologies: Regulation, research, and remaining challenges. *IEEE Communications Surveys & Tutorials*, 20(3):1858–1877, 2018.

[3] Sylvain Argentieri, Patrick Danes, and Philippe Souères. A survey on sound source localization in robotics: From binaural to array processing methods. *Computer Speech & Language*, 34(1):87–112, 2015.

[4] Caleb Rascon and Ivan Meza. Localization of sound sources in robotics: A review. *Robotics & Autonomous Systems*, 96:184–210, 2017.

[5] Lin Wang and Andrea Cavallaro. Acoustic sensing from a multi-rotor drone. *IEEE Sensors Journal*, 18(11):4570–4582, 2018.

[6] David B Lindell, Gordon Wetzstein, and Vladlen Koltun. Acoustic non-line-of-sight imaging. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6780–6789, 2019.

[7] Keita Okutani, Takami Yoshida, Keisuke Nakamura, and Kazuhiro Nakadai. Outdoor auditory scene analysis using a moving microphone array embedded in a quadrocopter. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3288–3293. IEEE, 2012.

[8] Inkyu An, Myungbae Son, Dinesh Manocha, and Sung-eui Yoon. Reflection-aware sound source localization. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 66–73. IEEE, 2018.

[9] Yoonho Jang, Jaekwang Kim, and Jinhak Kim. The development of the vehicle sound source localization system. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, pages 1241–1244. IEEE, 2015.

[10] Agnieszka Stelling-Kończak, Marjan Hagenzieker, and Bert Van Wee. Traffic sounds and cycling safety: The use of electronic devices by cyclists and the quietness of hybrid and electric cars. *Transport Reviews*, 35(4):422–444, 2015.

[11] Mitsunori Mizumachi, Atsunobu Kaminuma, Nobutaka Ono, and Shigeru Ando. Robust sensing of approaching vehicles relying on acoustic cues. *Sensors*, 14(6):9546–9561, 2014.

[12] Adarsh Venkata Padmanabhan, Hariram Ravichandran, et al. Acoustics based vehicle environmental information. Technical report, SAE, 2014.

[13] Kensaku Asahi, Hideki Banno, Osami Yamamoto, Akira Ogawa, and Keiichi Yamada. Development and evaluation of a scheme for detecting multiple approaching vehicles through acoustic sensing. In *IEEE Intelligent Vehicles Symposium*, pages 119–123. IEEE, 2011.

[14] Victor Singh, Katherine E Knisely, et al. Non-line-of-sight sound source localization using matched-field processing. *Journal of the Acoustical Society of America*, 131(1):292–302, 2012.

[15] Takuya Toyoda, Nobutaka Ono, Shigeki Miyabe, Takeshi Yamada, and Shoji Makino. Traffic monitoring with ad-hoc microphone array. In *International Workshop on Acoustic Signal Enhancement*, pages 318–322. IEEE, 2014.

[16] Shigemi Ishida, Jumpei Kajimura, Masato Uchino, Shigeaki Tagashira, and Akira Fukuda. SAVeD: Acoustic vehicle detector with speed estimation capable of sequential vehicle detection. In *International Conference on Intelligent Transportation Systems*, pages 906–912. IEEE, 2018.

[17] Ulf Sandberg, Luc Goubert, and Piotr Mioduszewski. Are vehicles driven in electric mode so quiet that they need acoustic warning signals. In *International Congress on Acoustics*, 2010.

[18] Lykke M Iversen and Rasmus Stahlfest Holck Skov. Measurement of noise from electrical vehicles and internal combustion engine vehicles under urban driving conditions. *Euronoise*, 2015.

[19] Ryan Robart, Etienne Parizet, Jean-Christophe Chamard, et al. eVADER: A perceptual approach to finding minimum warning sound requirements for quiet cars. In *AIA-DAGA 2013 Conference on Acoustics*, 2013.

[20] Sang Kwon Lee, Seung Min Lee, Taejin Shin, and Manug Han. Objective evaluation of the sound quality of the warning sound of electric vehicles with a consideration of the masking effect: Annoyance and detectability. *International Journal of Automotive Technology*, 18(4):699–705, 2017.

[21] Robin Scheibler, Eric Bezzam, and Ivan Dokmanić. Pyroomacoustics: A python package for audio room simulation and array processing algorithms. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 351–355. IEEE, 2018.

[22] Joseph Hector DiBiase. *A high-accuracy, low-latency technique for talker localization in reverberant environments using microphone arrays*. Brown University Providence, RI, 2000.

[23] Maarten Hornikx and Jens Forssén. Modelling of sound propagation to three-dimensional urban courtyards using the extended Fourier pstd method. *Applied Acoustics*, 72(9):665–676, 2011.

[24] Wen Zhang, Parasanga N Samarasinghe, Hanchi Chen, and Thushara D Abhayapala. Surround by sound: A review of spatial audio recording and reproduction. *Applied Sciences*, 7(5):532, 2017.

[25] Keiichi Osako, Yuki Mitsufuji, et al. Supervised monaural source separation based on autoencoders. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 11–15. IEEE, 2017.

[26] Ashutosh Saxena and Andrew Y Ng. Learning sound location from a single microphone. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1737–1742. IEEE, 2009.

[27] Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, 24(3):279–283, 2017.

[28] Abhinav Valada, Luciano Spinello, and Wolfram Burgard. Deep feature learning for acoustics-based terrain classification. In *Robotics Research*, pages 21–37. Springer, 2018.

[29] Nelson Yalta, Kazuhiro Nakadai, and Tetsuya Ogata. Sound source localization using deep learning models. *Journal of Robotics and Mechatronics*, 29(1):37–48, 2017.

[30] Weipeng He, Petr Motlicek, and Jean-Marc Odobez. Deep neural networks for multiple speaker detection and localization. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 74–79. IEEE, 2018.

[31] Chuang Gan, Hang Zhao, Peihao Chen, David Cox, and Antonio Torralba. Self-supervised moving vehicle tracking with stereo sound. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2019.

[32] Nicolas Scheiner, Florian Kraus, Fangyin Wei, et al. Seeing around street corners: Non-line-of-sight detection and tracking in-the-wild using doppler radar. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2068–2077, 2020.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[34] L Ferranti, B Brito, E Pool, Y Zheng, et al. SafeVRU: A research platform for the interaction of self-driving vehicles with vulnerable road users. In *IEEE Intelligent Vehicles Symposium*, pages 1660–1666. IEEE, 2019.

[35] Fabian Pedregosa, Gaël Varoquaux, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[36] Ennes Sarradj and Gert Herold. A python framework for microphone array data processing. *Applied Acoustics*, 116:50–58, 2017.

[37] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[38] Ting-Fan Wu, Chih-Jen Lin, and Ruby C Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5(Aug):975–1005, 2004.

And now for something completely different.

Monty Python

3

End-to-end Learning of Decision Trees and Forests

Conventional decision trees have a number of favorable properties, including a small computational footprint, interpretability, and the ability to learn from little training data. However, they lack a key quality that has helped fuel the deep learning revolution: being end-to-end trainable. [1] have addressed this deficit but at the cost of losing a main attractive trait of decision trees: the fact that each sample is routed along a small subset of tree nodes only. An end-to-end learning scheme for deterministic decision trees and decision forests is presented. Thanks to a new model and expectation-maximization training scheme, the trees are fully probabilistic at train time, but after an annealing process become deterministic at test time. In experiments, the effect of annealing is explored visually and quantitatively, and it is found that the method performs on par or superior to standard learning algorithms for oblique decision trees and forests. Further, it is demonstrated on image datasets that the proposed approach can learn more complex split functions than common oblique ones, and facilitates interpretability through spatial regularization.

Parts of this chapter have been published as: Thomas M. Hehn, Julian F.P. Kooij, and Fred A. Hamprecht. "End-to-end learning of decision trees and forests", International Journal of Computer Vision, 128, 4, p. 997-1011, 2020, Springer US.. Results in sections 3.3.1, 3.3.4, 3.3.5, and 3.3.6 have been updated after fixing a bug that had been reported in the public code repository. The contributions and conclusions remain valid.

3.1 Introduction

Neural networks are currently the dominant classifier in computer vision [2, 3], whereas decision trees and decision forests have proven their worth when training data or computational resources are scarce [4, 5]. One can observe that both neural networks and decision trees are composed of basic computational units, the perceptrons and nodes, respectively. A crucial difference between the two is that in a standard neural network, *all* units are evaluated for every input, while in a reasonably balanced decision tree with I inner split nodes, only $\mathcal{O}(\log I)$ split nodes are visited. That is, in a decision tree, a sample is routed along a single path from the root to a leaf, with the path conditioned on the sample's features. Various works are now exploring the relationship between both classification approaches [6, 7], such as the Deep Neural Decision Forests (DNDFs) [1]. Similar to deep neural networks, DNDFs require evaluating all computational paths to all leaf nodes in a tree for each test sample, which results in high accuracy, but incurs large computational and memory costs especially as trees grow deeper.

This work proposes an orthogonal approach. The idea is to stick to traditional decision trees and forests as inference models for their advantages, while improving learning of such trees through end-to-end training with back-propagation, one of the hallmarks of neural networks. It is efficiency, induced by the sparsity of the sample-dependent computational graph, that piques interest in decision trees. Further, decision trees provide relative interpretability. End-to-end training allows optimizing all levels of a decision tree jointly. Furthermore, features can now be jointly learned through linear nodes, but also through more complex split functions such as small convolutional neural networks (CNNs). This is a feature that has so far been missing in deterministic decision trees, which are usually constructed greedily without subsequent tuning. A mechanism to remedy this deficit is proposed.

3.1.1 Related work

Random forests are ensembles of decision trees, and were introduced by [8]. In this section, use cases for trees and forests, choices for split functions, different optimization strategies, and the connection to deep neural networks are reviewed.

Applications While neural networks have these days superseded all other approaches in terms of achievable accuracy on many benchmarks [2, 3, 9, 10], state-of-the-art networks are not easy to interpret, are fairly hungry for training data, often require weeks of GPU training, and have a computational and memory footprint that limits their use on small embedded devices. Decision trees and decision tree ensembles, such as random forests, generally achieve lower accuracy on large datasets but are fundamentally more frugal. They have shown their effectiveness in a variety of classification tasks [4, 11] and also found wide application in computer vision, e.g. [5, 12–16]. They are well suited for tasks where computational resources are limited, e.g. real-time human pose estimation in the Microsoft Kinect [13], or few and unbalanced training samples are available, e.g. during online object tracking [15].

Decision trees are also found in expert systems since they are widely recognized as interpretable models which divide a complex classification task into several simpler ones. For instance, [17–19] use their interpretability for diabetes research, and [20] interpret their outcome prediction of in vitro fertilization. Likewise, [21] explains the application of decision trees in business analytics.

Split functions There have been several attempts to train decision trees with more complex split functions. [22] have benchmarked oblique random forests on various binary classification problems. These oblique random forests used linear and non-linear classifiers at each split in the decision trees and thereby combined more than one feature at a time. [23] have successfully approximated the information gain criterion using a sigmoid function and a smoothness hyperparameter. Expanding these ideas, [24] have trained small convolutional neural networks (CNNs) at each split in a decision tree to perform binary segmentation. [25] also apply a greedy strategy to learn neural networks for each split node and hence learn the structure of the tree. Notably, these approaches use gradient optimization techniques, but are lacking joint optimization of an entire tree, i.e. end-to-end learning of the entire model.

Optimization [26] have shown that the problem of finding an optimal decision tree is NP-complete. As a consequence, the common approach is to find axis-aligned splits by exhaustive search, and learn a decision tree with a greedy level-by-level training procedure as proposed by [27]. In order to improve their performance, it is common practice to engineer split features for a specific task [16, 28–30]. Evolutionary algorithms are another group of optimization methods which can potentially escape local optima, but are computationally expensive and heuristic, requiring to tune many parameters (cf. [4] for a survey).

[31] propose an algorithm for the optimization of an entire tree with a given structure. They show a connection between optimizing oblique splits and structured prediction with latent variables. As a result, they formulate a convex-concave upper bound on the tree's empirical loss. The same upper bound is used to find an initial tree structure in a greedy algorithm. Their method is restricted to linear splits and relies on the kernel trick to introduce higher-order split features. Alternating Decision Forests [32] instead include a global loss when growing the trees, thereby optimizing the whole forest jointly.

Some works have explored gradient-based optimization of a full decision tree model already. While [33] focused on a fuzzy approach of decision tree, [34] introduced hierarchical mixtures of experts. In the latter model, the predictions of expert classifiers are weighted based on conditional path probabilities in a fully probabilistic tree.

[1] make use of gradient-based decision tree learning to learn a deep CNN and use it as a feature extractor for an entire ensemble of decision trees. They use sigmoid functions to model the probabilistic routes and employ a log-likelihood objective for training. However, their inference model is unlike a standard tree as it stays fuzzy or probabilistic after training. When predicting new samples, all leaves and paths need to be evaluated for every sample, which subverts the computational benefits of trees. Furthermore, they consider only balanced trees, so the number of evaluated split functions at test time grows exponentially with increased tree depth.

Connections to deep neural networks Various works explore the connections between neural networks and traditional decision tree ensembles. [35, 36] cast decision tree ensembles to neural networks, which enables gradient descent training. As long as the structure of the trees is preserved, the optimized parameters of the neural network can also be mapped back to the decision forest. Subsequently, [37] map stacked decision forests to CNNs and found an approximate mapping back. [38] focus on using the learned predictions of neural networks as a training target for probabilistic decision trees.

A related research direction is to learn conditional computations in deep neural networks. In [6, 39–42], several models of neural networks with separate, conditional data flows are discussed. Still, the structure of the resulting inference models is fixed a priori.

3.1.2 Contributions

This work extends a previous conference contribution [43] where end-to-end learning for decision trees was introduced. Here, an extension to decision forests is added, which is compared to state-of-the-art methods for training forests, and additional results on interpretability and the effect of the steepness parameter are provided. Compared to existing research, this work provides the following contributions:

- It is proposed to learn deterministic decision trees and forests in an end-to-end fashion. Unlike related end-to-end approaches [1], trees with deterministic nodes at test time are obtained. This results in efficient inference as each sample is only routed along one unique path of only $\mathcal{O}(\log I)$ out of the I inner nodes in a tree. To reduce variance, multiple trees can also be combined in a decision forest ensemble. Furthermore, an end-to-end trainable tree can provide interpretable classifiers on learned visual features, similar to how decision trees are used in financial or medical expert systems on handcrafted features. In this context, it is shown the benefit of regularizing the spatial derivatives of learned features when samples are images or image patches.

- To enable end-to-end training of a decision tree, differentiable probabilistic nodes are used at *train time only*. A new probabilistic split criterion generalizes the long-established information gain [44]. A key aspect of this new tree formulation is the introduction of a steepness parameter for the decision [23]. The proposed criterion is asymptotically identical to information gain in the limit of very steep non-linearities but allows to better model class overlap in the vicinity of a split decision boundary.

- A matching optimization procedure is proposed. During training, the probabilistic trees are optimized using the Expectation-Maximization algorithm [45]. Importantly, the steepness parameter is incrementally adjusted in an annealing scheme to make decisions ever more deterministic, and bias the model towards crispness. The proposed procedure also constructs the decision trees level-by-level, hence trees will not grow branches any further than necessary. Compared to initialization with balanced trees, [1] the proposed approach reduces the expected depth of the tree, which further improves efficiency.

Section 3.2 formalizes the new optimization procedure and probabilistic decision tree formulation. In section 3.3, the performance of the proposed method is compared to that of related work on decision trees and decision forests. The benefits of the annealing scheme are evaluated when training decision forests for a given number of epochs. The improved interpretability by regularizing the spatial derivatives of learned features on images or image patches is demonstrated, Further, it is shown that CNNs can be used as split features. Finally, section 3.4 presents the conclusions and suggests future work.

3.2 Methods

Consider a classification problem with input space $\mathcal{X} \subset \mathbb{R}^p$ and output space $\mathcal{Y} = \{1, ..., K\}$. The training set is defined as $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\} = \mathcal{X}_t \subset \mathcal{X}$ with corresponding classes $\{y_1, ..., y_N\} = \mathcal{Y}_t \subset \mathcal{Y}$.

A probabilistic decision tree model enables end-to-end learning. Nevertheless, the proposed model remains deterministic at test time. To account for this discrepancy, a steepness parameter is introduced to gradually enforce more deterministic splits during training. This addition is further motivated by the connection of the learning objective to the information gain criterion (see section 3.2.6).

3.2.1 Standard decision tree and notation

In binary decision trees (figure 3.1c), split functions $s : \mathbb{R} \longrightarrow [0,1]$ determine the path of a sample through the tree, conditioned on that sample's features. The split function controls whether the splits are deterministic or probabilistic. For deterministic decision trees as proposed in [27], the split function is a step function $s(x) = \Theta(x)$ with $\Theta(x) = 1$ if $x > 0$ and $\Theta(x) = 0$ otherwise. The leaf node that is at the end of a path finally provides the prediction for an input sample.

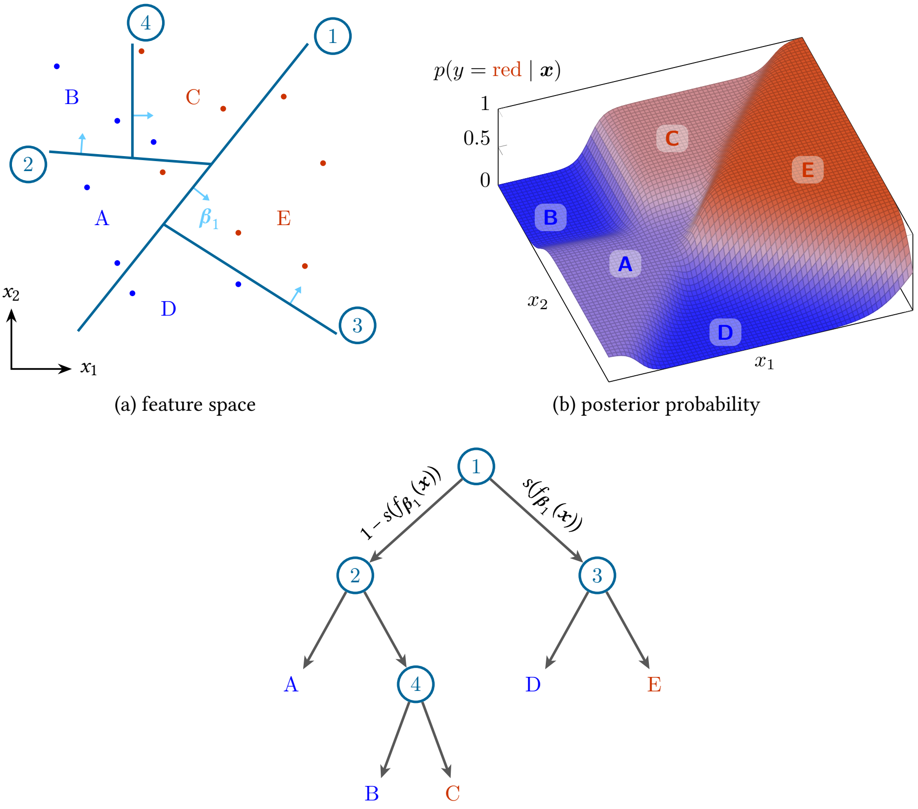
Split nodes At each split node $i \in \{1, ..., I\}$, a function $f_{\boldsymbol{\beta}_i} : \mathbb{R}^p \longrightarrow \mathbb{R}$ with parameters $\boldsymbol{\beta}_i$ computes a split feature from the input sample. This split feature is then the input for the split function s. Split features of *oblique splits* are computed through linear combination of the input sample $\boldsymbol{x}$ and the parameters, i.e. $f_{\boldsymbol{\beta}_i}(\boldsymbol{x}) = (\boldsymbol{x}^T, 1) \cdot \boldsymbol{\beta}_i$ with $\boldsymbol{\beta}_i \in \mathbb{R}^{p+1}$. Similarly, an axis-aligned split perpendicular to axis a is represented by an oblique split whose only non-zero parameters are at index a and $p+1$. $\boldsymbol{\theta}_{\boldsymbol{\beta}} = (\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_I)$ denotes the collection of all split parameters in the tree.

Leaf nodes Each leaf $\ell \in \{1, ..., L\}$ stores the parameters of a categorical distribution over classes $k \in \{1, ..., K\}$ in a vector $\boldsymbol{\pi}_\ell \in [0,1]^K$. These vectors are normalized such that the probability of all classes in a leaf sum to $\sum_{k=1}^{K} (\boldsymbol{\pi}_\ell)_k = 1$. The notation $\boldsymbol{\theta}_{\boldsymbol{\pi}} = (\boldsymbol{\pi}_1, ..., \boldsymbol{\pi}_L)$ includes all leaf parameters in the tree.

Paths Each leaf is reached by precisely one unique set of split outcomes, called a path. The probability that a sample $\boldsymbol{x}$ takes the path to a leaf ℓ is defined as

$$\mu_\ell(\boldsymbol{x}; s, \boldsymbol{\theta}_{\boldsymbol{\beta}}) = \prod_{r \in \mathcal{R}_\ell} s(f_{\boldsymbol{\beta}_r}(\boldsymbol{x})) \prod_{l \in \mathcal{L}_\ell} \left(1 - s(f_{\boldsymbol{\beta}_l}(\boldsymbol{x}))\right). \tag{3.1}$$

Here, $\mathcal{R}_\ell \subset \{1, ..., I\}$ denotes the splits on the path which contain ℓ in the right subtree. Analogously, $\mathcal{L}_\ell \subset \{1, ..., I\}$ denotes splits which contain ℓ in the left subtree. In figure 3.1c this means that $\mathcal{R}_{\mathrm{B}} = \{2\}$ and $\mathcal{L}_{\mathrm{B}} = \{1, 4\}$. Also note that in the following, the dependency on s is omitted, except when a specific function s is considered.


4
1
B
C
2
β_1
A
E
x_2
D
3
x_1
$p(y = \text{red} \mid \boldsymbol{x})$
1
0.5
0
C
E
B
A
x_2
D
x_1
1
$1 - s(f_{\beta_1}(\boldsymbol{x}))$
$s(f_{\beta_1}(\boldsymbol{x}))$
2
3
A
4
D
E
B
C

(a) feature space

(b) posterior probability

(c) tree graph

Figure 3.1: Probabilistic oblique decision trees. a) A feature space with a binary classification problem tessellated by an example oblique decision tree. The oblique splits (1-4) partition the feature space into five different leaves (A-E). b) The predicted $p(y = \text{red} \mid \boldsymbol{x})$ (equation 3.2) of the oblique decision tree when a probabilistic split (equation 3.3) is used. c) The corresponding tree diagram.

The prediction of the entire decision tree is given by multiplying the path probability with the corresponding leaf prediction:

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \sum_{\ell=1}^{L} (\boldsymbol{\pi}_{\ell})_y \, \mu_\ell(\boldsymbol{x};\boldsymbol{\theta}_{\boldsymbol{\beta}}). \tag{3.2}$$

Here, $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\boldsymbol{\beta}}, \boldsymbol{\theta}_{\boldsymbol{\pi}})$ comprises all parameters in the tree. This representation of a decision tree allows choosing between different split features and different split functions, by varying the functions f and s, respectively.

3.2.2 Probabilistic decision tree

A defining characteristic of the proposed method is that the decision tree is probabilistic during training, similar to [1]. Rather than sending a sample deterministically down the right (or left) subtree depending on its features x, it is sent right with a probability

$$s(f(x)) = \sigma(f(x)) = \frac{1}{1 + e^{-f(x)}}. \tag{3.3}$$

This corresponds to regarding each split in the tree as a Bernoulli decision with mean $\sigma(f(x))$. As a result, equation 3.2 is the expected value over the possible outcomes. Figure 3.1b shows the prediction from equation 3.2 in the probabilistic case for a class $y =$ “red” on the classification problem illustrated in figure 3.1a.

To train the probabilistic decision trees, the empirical log-likelihood of the training data is chosen as the maximization objective:

$$\max_{\boldsymbol{\theta}} \mathfrak{L}(\boldsymbol{\theta}; \mathcal{X}_t, \mathcal{Y}_t) = \max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log p(y_n | \boldsymbol{x}_n; \boldsymbol{\theta}). \tag{3.4}$$

Importantly, while a *probabilistic* decision tree is used for training, a *deterministic* decision tree is used for prediction on test samples. To better match the models used at train and test time, a hyperparameter γ is introduced, which steers the steepness of the split function by scaling the split feature [23]

$$s(f(x)) = \sigma_\gamma(f(x)) = \sigma(\gamma f(x)). \tag{3.5}$$

Note, for $\gamma \longrightarrow \infty$ the model resembles a deterministic decision tree, since $\sigma_\infty(f(x)) = \Theta(f(x))$. During training, γ is iteratively increased, akin to a temperature cooling schedule in deterministic annealing [46].

3.2.3 Expectation-Maximization

To optimize the log-likelihood of equation 3.4, a gradient-based, EM-style optimization strategy is proposed, which requires f and s to be differentiable with respect to the split parameters $\boldsymbol{\beta}_i$. The derivation of the EM-algorithm for this model follows the spirit of [45]. Additional latent random variables are introduced $z_{n,\ell}$, which indicate that leaf ℓ generated the class label of a given data point $\boldsymbol{x}_n$. Including these latent variables, the optimization objective now becomes the complete data log-likelihood

$$\mathfrak{L}(\boldsymbol{\theta}; \mathcal{X}_t, \mathcal{Y}_t, \mathcal{Z}_t) = \sum_{n=1}^{N} \sum_{\ell=1}^{L} z_{n,\ell} \log \left((\boldsymbol{\pi}_\ell)_{y_n} \mu_\ell(\boldsymbol{x}_n; \boldsymbol{\theta}_{\boldsymbol{\beta}}) \right). \tag{3.6}$$

E-Step In the Expectation-Step, the expected value of the complete-data log-likelihood over the latent variables given the previous parameters $\boldsymbol{\theta}'$ is computed

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}') = E_{\mathcal{Z}_t|\mathcal{X}_t,\mathcal{Y}_t;\boldsymbol{\theta}'}[\mathfrak{L}(\boldsymbol{\theta}; \mathcal{X}_t, \mathcal{Y}_t, \mathcal{Z}_t)]. \tag{3.7}$$

For this purpose, it is necessary to compute the probability that $z_{n,\ell} = 1$ for each training sample n:

$$\begin{aligned} h_{n,\ell} &:= p(z_{n,\ell} = 1 \mid \boldsymbol{x}_n, y_n; \boldsymbol{\theta}') && (3.8) \\ &= \frac{p(y_n \mid z_{n,\ell} = 1, \boldsymbol{x}_n; \boldsymbol{\theta}') p(z_{n,\ell} = 1 \mid \boldsymbol{x}_n; \boldsymbol{\theta}')}{p(y_n \mid \boldsymbol{x}_n; \boldsymbol{\theta}')} && (3.9) \\ &= \frac{(\boldsymbol{\pi}'_\ell)_{y_n} \mu_\ell(\boldsymbol{x}_n; \boldsymbol{\theta}'_{\boldsymbol{\beta}})}{\sum_{\ell'=1}^{L} (\boldsymbol{\pi}'_{\ell'})_{y_n} \mu_{\ell'}(\boldsymbol{x}_n; \boldsymbol{\theta}'_{\boldsymbol{\beta}})}. && (3.10) \end{aligned}$$

Thus, the expectation value of the complete-data log-likelihood yields

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}') = \sum_{n=1}^{N} \sum_{\ell=1}^{L} h_{n,\ell} \log\left((\boldsymbol{\pi}_\ell)_{y_n} \mu_\ell(\boldsymbol{x}_n; \boldsymbol{\theta}_{\boldsymbol{\beta}})\right). \tag{3.11}$$

M-Step In the Maximization-Step, the expectation value computed in the E-Step (equation 3.11) is maximized to find the updated parameters $\boldsymbol{\theta}$,

$$\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}'). \tag{3.12}$$

Due to the latent variables that were introduced, it is now possible to separate the parameter dependencies in the logarithm into a sum. As a result, the leaf predictions and split parameters are optimized separately.

The optimization of the leaf predictions including the normalization constraint can be computed directly,

$$(\boldsymbol{\pi}_\ell)_k = \frac{\sum_{n=1}^{N} [y_n = k] h_{n,\ell}}{\sum_{n=1}^{N} h_{n,\ell}}. \tag{3.13}$$

Here, the Iverson bracket $[y_n = k]$ equals 1 if $y_n = k$ and 0 otherwise.

The optimization of the split parameters in the M-Step is performed using gradient-based optimization. The separated objective for the split parameters without the leaf predictions is

$$\max_{\boldsymbol{\theta}_{\boldsymbol{\beta}}} \sum_{n=1}^{N} \sum_{\ell=1}^{L} h_{n,\ell} \log \mu_\ell(\boldsymbol{x}_n; \boldsymbol{\theta}_{\boldsymbol{\beta}}). \tag{3.14}$$

In practice the first-order gradient-based stochastic optimization Adam [47] is used to optimize this objective.

In summary, each iteration of the algorithm requires evaluation of equations 3.10 and 3.13, as well as at least one update of the split parameters based on equation 3.14. This iterative algorithm can be applied to a binary decision tree of any given structure.

3.2.4 Complex splits and spatial regularization

The proposed optimization procedure only requires the split features f to be differentiable with respect to the split parameters. As a result, it is possible to implement more complex splits than axis-aligned or oblique splits. For example, it is possible to use a small convolutional neural network (CNN) as split feature extractor for f and learn its parameters (section 3.3.4).

Furthermore, the optimization objective can also include regularization constraints on the parameters. This is useful to avoid overfitting and learn more robust patterns. When the inputs are from images, spatial regularization also reveals more discernible spatial structures in the learned parameters without sacrificing accuracy (section 3.3.3). To encourage the learning of coherent spatial patterns at each split, a spatial regularization term [48] is added

$$-\lambda \sum_{i=1}^{I} \boldsymbol{\beta}_i^T M \boldsymbol{\beta}_i \tag{3.15}$$

to the maximization objective of the split features of equation 3.14. Here, matrix M denotes the Laplacian matrix when interpreting the image as a grid graph. For a single pixel, corresponding to weight β_i, the diagonal element M_{ii} contains the number of neighboring pixels. If pixels i and j are neighboring pixels, then $M_{ij} = M_{ji} = -1$. All remaining elements in M are 0. This regularization term penalizes spatial finite differences, encouraging similar parameters for neighboring pixels. The hyperparameter λ controls the regularization strength, with higher λ leading to stronger regularization.

3.2.5 Decision tree construction

The previous sections outlined how to fit a decision tree to training data, given a fixed tree topology (parameter learning). Additionally to this deterministic decision tree *Finetuning*, a *Greedy* algorithm constructs a tree by successively splitting nodes and optimizing them on subsets of the training data.

A limit on the number of training epochs serves as a stopping criterion for training of a single split. The size of the tree can be limited either by the maximum number of leaf nodes or the depth of the tree. Furthermore, in cases with a very small subset of the training data, it may happen that training of a split fails and all training samples are passed to a single child of that node. For these cases a maximum number of attempts to fit a split function is introduced. Step-by-step, the *Greedy* algorithm works as follows:

1. Initialize the decision tree with a single candidate node as the tree root.
2. Split the training data into subsets. Starting from the root node with the entire training dataset, the data is successively decomposed using deterministic routing. As a result, non-overlapping subsets are assigned to the candidate nodes.
3. For each candidate node:
    (a) If the training data subset of the candidate node is pure or the maximum number of attempts has been reached, skip steps 3b to 3d for this node and fix it as a leaf node.
    (b) Replace node with a new tree stump, i.e. one split node and two leaf nodes.
    (c) Optimize only the tree stump using the *Finetune* algorithm (see section 3.2.3) on the assigned training data subset for the specified number of epochs.
    (d) If training the stump failed, then try training a new stump by repeating from 3a.
4. Find leaf node candidates that may be split according to the specified tree limits.

5. If candidate nodes are found, repeat from 2. Otherwise, stop, the decision tree construction is finished.

The main intent of this greedy algorithm is that trees are only grown further when necessary and thereby reduce the number of computations and parameters in the model (see section 3.3.6). The distribution of the training data in step 2 means that each node only shares training data with its ancestors. In particular, this means that, at first, the root split is trained on the entire training data. At some point the training data is pure, i.e. all samples are of the same class, and this node can be fixed as a leaf node.

During the training of a tree stump, only one split node and two leaf nodes are optimized. As a result, the log-likelihood objective (equation 3.4) then resembles an approximation of the widely used information gain criterion [44, 49] (section 3.2.6).

After this greedy structure learning, the nodes in the entire resulting tree can be finetuned jointly as described in section 3.2.3, this time with probabilistic routing of all training data.

3.2.6 Relation to information gain and leaf entropies

In the following, it is shown that maximization of the log-likelihood of the probabilistic decision tree model approximately minimizes the weighted entropies in the leaves. The steeper the splits become, the better the approximation.

To establish this connection hyperparameter γ is used to control the steepness of the probabilistic split function (equation 3.5). The function $\ell(\boldsymbol{x})$ returns the index of the leaf that sample $\boldsymbol{x}$ reaches when the path is evaluated deterministically

$$\ell(\boldsymbol{x}) = \sum_{\ell=1}^{L} \ell \lim_{\gamma \to \infty} \mu_\ell(\boldsymbol{x}; \sigma_\gamma, \boldsymbol{\theta}_{\boldsymbol{\beta}}). \tag{3.16}$$

This simplifies the log-likelihood objective (equation 3.4) to

$$\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log(\boldsymbol{\pi}_{\ell(\boldsymbol{x}_n)})_{y_n} \tag{3.17}$$

because each sample reaches only one leaf. Let $N_{\ell,k}$ be the number of training samples in leaf ℓ with class k and $N_\ell = \sum_{k=1}^{K} N_{\ell,k}$ denote all training samples in leaf ℓ. Since training samples with the same class and in the same leaf contribute the same term, the equations may be rearranged to

$$\max_{\boldsymbol{\theta}} \sum_{\ell=1}^{L} \sum_{k=1}^{K} N_{\ell,k} \log(\boldsymbol{\pi}_\ell)_k. \tag{3.18}$$

With $\gamma \longrightarrow \infty$, the optimal leaf predictions are the same as in a standard, deterministic decision tree, i.e. $(\boldsymbol{\pi}_\ell)_k = \frac{N_{\ell,k}}{N_\ell}$. Accordingly, the objective can be rewritten as

$$\max_{\boldsymbol{\theta}} \lim_{\gamma \to \infty} \mathfrak{L}(\boldsymbol{\theta}; \mathcal{X}_t, \mathcal{Y}_t) = \min_{\boldsymbol{\theta}} \sum_{\ell=1}^{L} \frac{N_\ell}{N} H_\ell. \tag{3.19}$$

Here, $H_\ell = -\sum_{k=1}^{K} (\boldsymbol{\pi}_\ell)_k \log(\boldsymbol{\pi}_\ell)_k$ denotes the entropy in leaf ℓ.

In conclusion, for $\gamma \longrightarrow \infty$, maximizing the log-likelihood objective minimizes a weighted sum of leaf entropies. For the special case of a single split with two leaves, this is the same as maximizing the information gain. Consequently, the log-likelihood objective (equation 3.4) can be regarded as a generalization of the information gain criterion [44] to an entire tree.

3.2.7 Decision forest

Following the ideas introduced by [8], decision trees can be combined into a decision forest. Specifically, each decision tree is constructed with the *Greedy* algorithm on the full dataset. Afterward, using the *Finetune* algorithm, each tree is optimized end-to-end. Note that the result is a *decision forest* rather than a *random forest*, since each tree is trained independently on all train data rather instead of on random subsets.

In order to reduce the correlation between the decision tree predictions, each split function is only trained on a subset of the available features. For each split, this feature subset is sampled from a uniform distribution or, in the case of 2D images, will consist of connected 2D patches of the image.

Let $\boldsymbol{\theta}_t$ denote all parameters of a single tree t out of T trees in the ensemble $\mathcal{T}$ of learned trees. The final prediction of the decision forest for a single sample $\boldsymbol{x}$ is computed as the mean prediction of the single trees (equation 3.2):

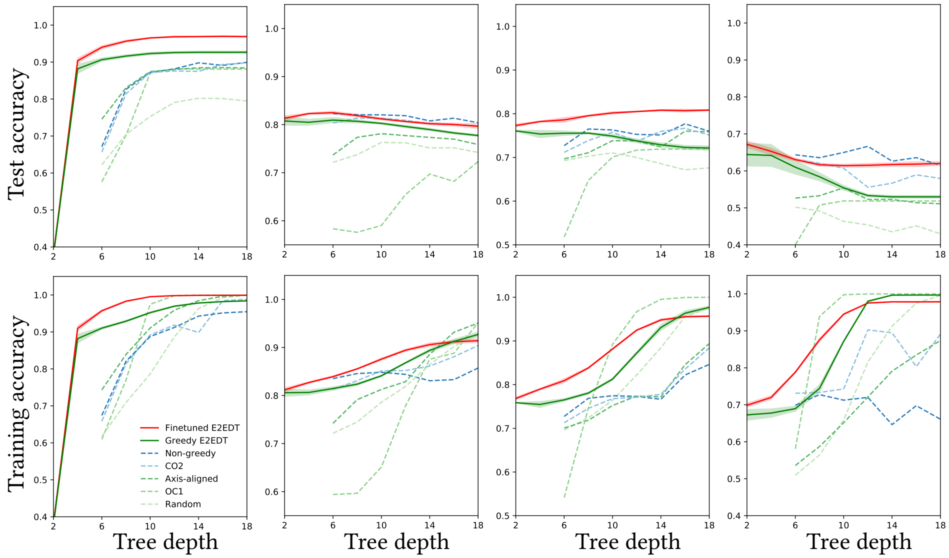
$$p(y|\boldsymbol{x};\mathcal{T}) = \frac{1}{T}\sum_{t\in\mathcal{T}} p(y|\boldsymbol{x};\boldsymbol{\theta}_t). \tag{3.20}$$

3.3 Experiments

The experiments are conducted on data from various domains. For quantitative comparison of the proposed end-to-end learned oblique decision trees (*E2EDT*), the performance is evaluated on the multivariate but unstructured datasets used in [31] (section 3.3.1). In order to understand the learning process of the probabilistic training and deterministic inference model, the models are visually examined on an image segmentation dataset (section 3.3.2). Next, it is shown that the proposed algorithm can learn meaningful spatial features on *MNIST*, *FashionMNIST*, and *ISBI*, as has previously been demonstrated in neural networks but not in decision trees (section 3.3.3). It is also demonstrated that a deterministic decision tree with complex split nodes can be trained end-to-end, by using a small neural network in each split node (section 3.3.4). Further, the effect of the steepness annealing in an end-to-end learned decision forest is quantitatively evaluated (*E2EDF*, section 3.3.5), and the model is compared to state-of-the-art decision forests (section 3.3.6) with respect to the trade-off between computational load and accuracy.

For gradient-based split parameter optimization, the Adam optimizer [47] with default parameters ($\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$) and a batch size of 1000 with shuffled batches is used. All data is normalized to zero mean and unit variance based on the training data. Unless stated otherwise, this setup is used for all experiments. The source code of the implementation using PyTorch [50] is available online[1].

[1] Code is available at `http://www.github.com/tomsal/endtoenddecisiontrees`



(a) MNIST (b) SensIT (c) Connect4 (d) Protein

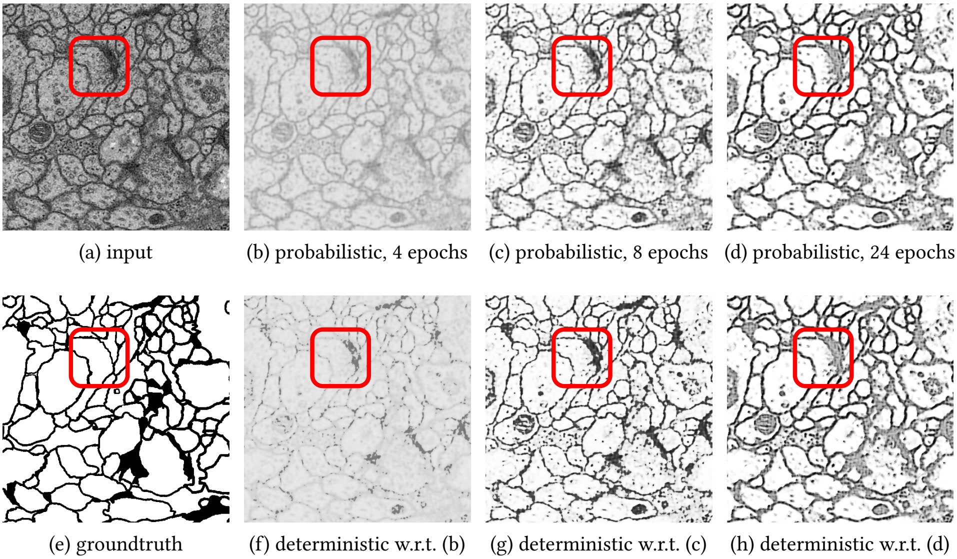
Figure 3.2: Accuracy on test and training sets of various optimization methods for deterministic oblique decision trees. For the proposed approach, results with *Greedy* initialization only (solid, light red line) and after being *Finetuned* (solid, dark red line) are shown. The results of the baseline algorithms (dashed lines) were reported in [31], see text for more details. The maximum tree depth varies from 2 to 18 with stepsize 2.

3.3.1 Performance of oblique decision trees

The performance of the proposed algorithm is compared to all results reported in [31] in terms of accuracy. In order to provide a fair comparison, neither pruning, ensembles, nor regularization are used.

Datasets [31] reports results on the following four datasets: *MNIST* [51], *SensIT* [52], *Connect4* [53] and *Protein* [54]. The multi-class classification datasets are obtained from the LIBSVM repository [55]. When a separate test set is not provided, the data is randomly split into a training set with 80% of the data and use 20% for testing. Likewise, when no validation set is provided, 20% of the training set is randomly extracted as validation set.

Compared algorithms The algorithms that are compared use a deterministic decision tree for prediction, with either oblique or axis-aligned splits. The following baselines were evaluated in [31]: *Axis-aligned*: conventional axis-aligned splits based on information gain; *OC1*: oblique splits optimized with coordinate descent as proposed in [56]; *Random*: selected the best of randomly generated oblique splits based on information gain; *CO2*: greedy oblique tree algorithm based on structured learning [57]; *Non-greedy*: non-greedy oblique decision tree algorithm based on structured learning [31].


(a) input (b) probabilistic, 4 epochs (c) probabilistic, 8 epochs (d) probabilistic, 24 epochs

(e) groundtruth (f) deterministic w.r.t. (b) (g) deterministic w.r.t. (c) (h) deterministic w.r.t. (d)

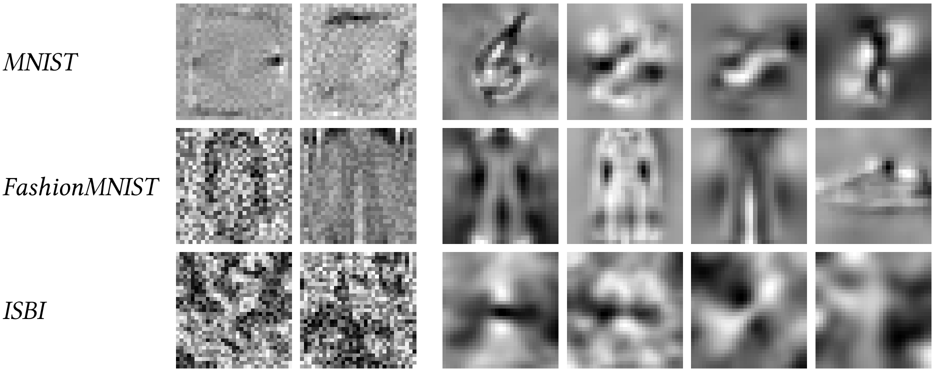
Figure 3.3: Posterior probability (equation 3.2) after various epochs of learning a single oblique decision tree on the ISBI binary segmentation dataset. (a) shows the input image and (e) the corresponding ground-truth labels. (b)-(d) illustrate the posterior of the probabilistic tree as γ increases at various stages during training. (f)-(h) illustrate the posterior of the deterministic equivalents at those stages after setting $\gamma \longrightarrow \infty$. Except for (a), darker means higher probability for class “membrane”. Note how discrepancies between the predictions of the probabilistic tree and its deterministic counterpart disappear as steepness γ increases. The highlighted region is discussed in the text.

The results of these algorithms are compared with two variants of the proposed method. Here, *Greedy E2EDT* denotes a greedy initialization where each oblique split is computed using the EM optimization. For each depth, the *Finetune E2EDT* algorithm is applied to the tree obtained from the *Greedy E2EDT* algorithm at that depth. In the following, they are referred to as *Greedy* and *Finetune*.

Hyperparameters and initialization The split steepness hyperparameter is set to $\gamma = 1.0$ initially and increased by $\Delta\gamma$ after each epoch. One epoch consists of the split parameter $\boldsymbol{\theta}_{\beta}$ updates of all training batches as well as the update of the leaf predictions $\boldsymbol{\theta}_{\pi}$. A grid search over the number of training epochs in $\{20, 35, 50, 65\}$ and the steepness increase $\Delta\gamma \in \{1, 0.1, 0.01, 0.001, 0.0001\}$ is conducted using a train/validation split.

The grid search results in the following combinations: $\Delta\gamma = 0.001$ for 20 epochs on *MNIST*, $\Delta\gamma = 0.001$ for 65 epochs on *SensIT*, $\Delta\gamma = 0.001$ for 50 epochs on *Connect4*, and $\Delta\gamma = 0.001$ for 20 epochs on *Protein*. The test data is only used to report the final performance and all other hyperparameters are kept fixed. Initial split directions are sampled from the unit sphere and the categorical leaf predictions are initialized uniformly.

Results Figure 3.2 shows the test and training statistical accuracy of the different decision tree learning algorithms. The accuracy of a classifier is defined as the ratio of correctly



(a) Without spatial regularization (b) With spatial regularization

Figure 3.4: Visualizations of oblique split parameters learned with and without spatial regularization (section 3.2.4). Each row shows a selection of parameters that were learned on a different dataset, namely *MNIST* [51], *FashionMNIST* [58], and *ISBI* [9]. Parameters trained with spatial regularization show visible structures and patterns, whereas parameters learned without regularization appear noisy.

classified samples in the respective set. It was evaluated for a single tree at various maximum depths. The red solid lines show the result of the proposed algorithm, and the dashed lines represent results reported by [31].

The proposed algorithms achieve higher test accuracy than previous work, especially in extremely shallow trees. On the *MNIST* and *Connect4* data sets, previous approaches are outperformed for oblique decision trees at all depths. In particular, an oblique decision tree of depth 4 is already sufficient to surpass all competitors.

On *SensIT* and *Protein*, the proposed approach performs better than or on par with the *Non-greedy* approach proposed in [31]. Note that further hyperparameter tuning may reduce overfitting, e.g. on the *Protein* dataset, and thus the results may improve.

In conclusion, the proposed (*E2EDT*) algorithm is able to learn more accurate deterministic oblique decision trees than the previous approaches.

3.3.2 Visual convergence of training and inference model

During training, the probabilistic training model s gradually steered towards a deterministic model by increasing the steepness γ. Next, the difference between the probabilistic training model and the deterministic inference model is visually examined. For this purpose, an oblique decision tree is trained on a binary image segmentation task on the ISBI challenge dataset [9]. This challenging image segmentation benchmark comprises serial section Transmission Electron Microscopy images (figure 3.3a) and binary annotations of neurons and membranes (figure 3.3e). For every pixel, a 9×9 window around the current pixel serves as input features to an oblique decision tree. Consequently, the learned parameters at each split node can be regarded as a spatial kernel. The tree is initialized as a balanced oblique decision tree of depth 6 and the *Finetune* algorithm is used to optimize the entire tree with the default steepness increase of $\Delta\gamma = 0.1$ per epoch.

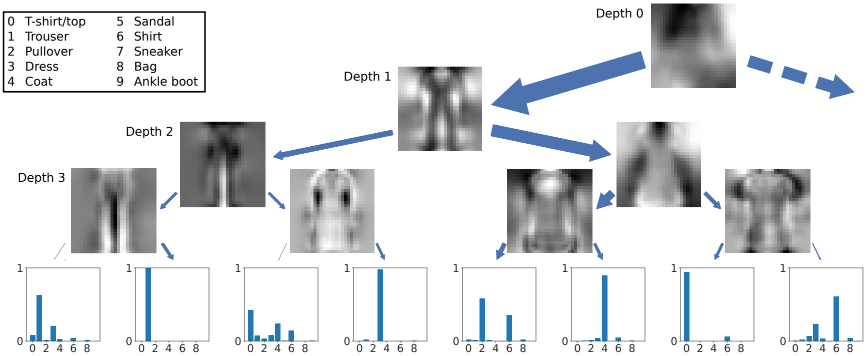


Figure 3.5: Visualization of an oblique decision tree learned on FashionMNIST [58] with spatial regularization. The right branch of the root (dashed arrow) is hidden for clarity. The split parameters are visualized as in figure 3.4. Intuitively, the better the input image matches the parameter image, the more likely the sample will go to the right child. If the input image better resembles the negative parameter image, the sample will go to the left. The thickness of an arrow indicates the number of training samples following the path when the decision tree is evaluated deterministically. Distributions at the leaves are visualized as bar plots. The x-axis denotes classes (see legend in the top-left), and the y-axis corresponds to the class probability.

Results Figure 3.3a shows a sample image of the input and figure 3.3e the corresponding ground-truth labels. Figures 3.3b to 3.3d illustrate the posterior probability (equation 3.2) predicted by the probabilistic training model at different training stages. The posterior probabilities of the corresponding inference models are shown below, in figures 3.3f to 3.3h. The visualization of the prediction shows pixels more likely to be of class “membrane” with darker color.

The gradual convergence of the probabilistic training model and the deterministic inference model is well visible. After 4 epochs, the probabilistic model (figure 3.3b) already reflects the structure of the segmentation problem. The corresponding deterministic model is more fragmented and also exhibits stronger confidence values, i.e. darker pixels. This is especially clear to see in the highlighted red squares in figure 3.3. The discrepancy between the probabilistic and deterministic models is reduced after 8 epochs (figures 3.3c and 3.3g). However, the deterministic posterior estimate reveals high confidence even in misclassified areas (higher contrast). This effect is dampened after 24 epochs. The corresponding leaf probabilities now learned to estimate confidence and thus show more gray areas in difficult regions. The differences between the probabilistic (figure 3.3d) and the deterministic (figure 3.3h) predictions are hardly visible anymore.

3.3.3 Interpretation of spatially regularized parameters

Now, the effects of spatial regularization are investigated (section 3.2.4) on the parameters of oblique decision trees learned with the proposed algorithm. Recall that regularization penalizes differences in adjacent parameters. For this purpose, oblique decision trees are trained on the *MNIST* digit dataset [51], the *FashionMNIST* fashion product dataset [58] and the *ISBI* image segmentation dataset [9]. For *MNIST* and *FashionMNIST*, the training images consist of 28 × 28 images. For the segmentation task on *ISBI*, a sliding window of size 31 × 31 is used as input features for each pixel in the center of the window.

Results In figure 3.4, selected parameters of the oblique splits at various depths with and without regularization are visualized. The learned parameter vectors are reshaped to the respective training image dimensions and linearly normalized to the full grayscale range. In both cases, parameter vectors were selected that display interesting visible structures.

The parameters without regularization appear very noisy. In contrast, with regularization, the algorithm learns smoother parameter patterns, without decreasing the accuracy of the decision trees. The patterns learned on the *MNIST* show visible sigmoidal shapes and even recognizable digits. On the *FashionMNIST* dataset, the regularized parameters display the silhouettes of coats, pants, and sneakers. Likewise, the proposed algorithm is able to learn the structures of membranes on the real-world biological electron microscopy images from the *ISBI* dataset.

Figure 3.5 visualizes half of a learned tree for the *FashionMNIST* dataset. One can see that at deeper splits, more distinctive features of various fashion products are tested. The split parameters of the first split at depth 3 show bright trousers and its right child predicts the class “trousers”. The same holds for the second split at depth 3, showing a bright silhouette of a dress and its right child predicts “dress”. The parameters of the third split at depth 3 reveal some kind of upper body clothes, but it is difficult to determine the kind. Yet, these parameters separate samples of class “pullover” and “shirt” (left child) from class “coat” (right child). Such decision tree illustrations thus reveal important features that drive the internal decisions made towards the final prediction. This provides a useful tool to interpret the proposed model, akin to the use of decision trees in expert systems of other domains [19, 21].

3.3.4 CNN split features

In the following, the effectiveness of CNNs as split features in a decision tree on *MNIST* is investigated. At each split, a very simple CNN of the following architecture is trained: Convolution 5 × 5 kernel @ 3 output channels ⟶ Max Pool 2 × 2 ⟶ ReLU ⟶ Convolution 5 × 5 @ 6 ⟶ Max Pool 2 × 2 ⟶ ReLU ⟶ Fully connected layer 96 × 50 ⟶ ReLU ⟶ Fully connected layer 50 × 1. The final scalar output is the split feature, which is the input to the split function.

Again, greedy training initializes the tree, however, the nodes are split in a best-first manner, based on the highest information gain. As a result, the trees can be fairly unbalanced despite impure leaves. At a maximum of 10 leaves training is stopped to increase interpretability and efficiency by having one expert leaf per class.

Results In this setting, a single decision tree achieves a test accuracy of $98.2\% \pm 0.1\%$ deterministic evaluation of nodes. For comparison, a standard random forest ensemble with 100 trees only reaches $96.79\% \pm 0.07\%$.

Such decision tree models provide interesting benefits in interpretability and efficiency, which are the main advantages of decision trees. When a sample was misclassified it is straightforward to find the split node that is responsible for the error. This offers interpretability as well as the possibility to improve the overall model. Other methods, such as *OneVsOne* or *OneVsRest* multi-class approaches, provide similar interpretability, however at a much higher cost at test time. This is due to the fact that in a binary decision tree with K leaves, i.e. a leaf for each class, it is sufficient to evaluate $\mathcal{O}(\log K)$ split nodes.

In *OneVsOne* and *OneVsAll* it is necessary to evaluate $K(K-1)/2$ and K different classifiers at test time, respectively.

3.3.5 Steepness annealing analysis

In section 3.2, the introduction and annealing of the steepness hyperparameter is motivated based on two observations. Firstly, steeper decisions, although hindering end-to-end learning, reflect the final inference model more closely. Secondly, in the limit of steep decisions, the learning objective approximates the information gain (see section 3.2.6), which is well established for decision tree learning.

In this experiment, the effectiveness of annealing the steepness hyperparameter is investigated. For this purpose, decision forest ensembles of oblique deterministic decision trees are trained (see section 3.2.7). To study the impact on the performance, different annealing schemes for the steepness are used. The steepness is always initialized as $\gamma = 1.0$ and $\Delta\gamma > 0$ denotes the enforced increase in steepness after each epoch. Thus, $\Delta\gamma = 0$ effectively ignores the hyperparameter as it will stay constant during training. This comparison is performed for three different settings of the number of epochs (15, 30, 45). This means that during the *Greedy* tree construction each split is trained for exactly this number of epochs. Afterward, each tree is optimized end-to-end based on the *Finetune* algorithm for three times as many epochs as during the construction phase (e.g. 30 epochs *Greedy* and 90 epochs *Finetune* training). This choice is motivated by validation experiments that showed the importance of the *Finetune* algorithm in the decision forest and do not affect the comparison of different $\Delta\gamma$.

Datasets The procedure is the same as described in section 5.1 of [1], and uses the same datasets, number of features, and number of trees as they do. These datasets are *Letter* [59], *USPS* [60], and *MNIST* [51]. The features are randomly chosen for each split separately. For completeness, details on the dataset and specific settings are listed in table 3.2 in the appendix.

Results Table 3.1 shows the accuracy of an oblique decision forest when trained with different steepness increase. It is important to note that $\Delta\gamma = 0$ does not mean that the steepness is fixed throughout training. The model may still learn to have steeper decisions by increasing the L2-norm of the split parameters $\boldsymbol{\theta_\beta}$. After training for the same number of epochs, a steepness increase of $\Delta\gamma = 0.1$ per epoch consistently outperforms the trained models without annealing $\Delta\gamma = 0.0$ on the *Letter* and *USPS* datasets. On these two datasets, the final deterministic model improves by a large margin when trained with steepness increase. Interestingly, on *MNIST*, the decision forests without steepness increase perform slightly better than the corresponding models with $\Delta\gamma = 0.01$ when training for more epochs. A possible interpretation is that larger datasets do not benefit from this. Compared to the other datasets and considering the simplicity of the task, *MNIST* can be considered a large dataset. Conversely, further tuning of γ may show different results. Generally, the default steepness annealing choice of $\Delta\gamma = 0.1$ per epoch performs well.

Dataset	$\Delta\gamma$	15 Epochs	30 Epochs	45 Epochs
Letter	0.0	82.1	87.5	89.3
Letter	0.01	83.9	89.7	92.2
Letter	0.1	90.6	**94.4**	**95.5**
Letter	1.0	**93.2**	93.1	93.2
USPS	0.0	92.5	95.5	96.0
USPS	0.01	93.8	95.9	96.8
USPS	0.1	95.8	**96.8**	**97.1**
USPS	1.0	**96.5**	96.2	95.9
MNIST	0.0	98.0	**98.2**	**98.2**
MNIST	0.01	**98.1**	98.0	98.1
MNIST	0.1	97.9	97.8	97.7
MNIST	1.0	96.7	96.7	96.6

Table 3.1: Comparison of the validation accuracy of the proposed end-to-end learned deterministic decision forests for different values of the gradual steepness increase $\Delta\gamma$ on various datasets (see appendix, table 3.2). Best results per dataset and number of training epochs are highlighted **bold**.

3.3.6 Trade-off between computational load and accuracy

Due to the conditional data flow, deterministic decision forests only evaluate a fraction of the entire model during prediction and thus require significantly fewer computations than a probabilistic forest model. Now, the trade-off in computational load and accuracy of the proposed end-to-end learned deterministic decision forests is compared to the state-of-the-art probabilistic *shallow Neural Decision Forests* (sNDF) by [1]. For this purpose, the decision forest (*E2EDF*) is evaluated on the same datasets which were used in their evaluation (see section 3.3.5). Both models, *E2EDF* and *sNDF*, are based on oblique splits and use the same maximum depth per tree and the same number of trees in a forest as the *sNDF*.

Additionally, the results are compared to other deterministic tree ensemble methods: the standard random forest (*RF*), boosted trees (*BT*) and alternating decision forests (*ADF*). The corresponding results were reported by [32] and are always based on 100 trees in the ensemble with a maximum depth of either 10, 15, or 25. Since their work only lists ranges of explored parameter settings, the estimated computational load (i.e. number of split evaluations) is based on the most favorable parameter settings.

Since *BT*, *RF* and *ADF* are in practice limited to linear split functions, the *E2EDF* models are restricted to oblique splits as well in this comparison. To train the *E2EDF* models, the default steepness increase of $\Delta\gamma = 0.1$ per epoch is used. On *USPS* as well as *Letter*, the models are trained for 45 epochs, whereas on *MNIST*, training is done only for 15 epochs due to the larger amount of training data. Note that, as in section 3.3.5, the final tree is *Finetuned* for three times as many epochs as during the *Greedy* training (e.g. for *USPS*: 45 epochs *Greedy* and 135 epochs Finetune). Training is done on the full training data, i.e. including validation data, and evaluate on the provided test data. The reported accuracy is averaged over three runs.

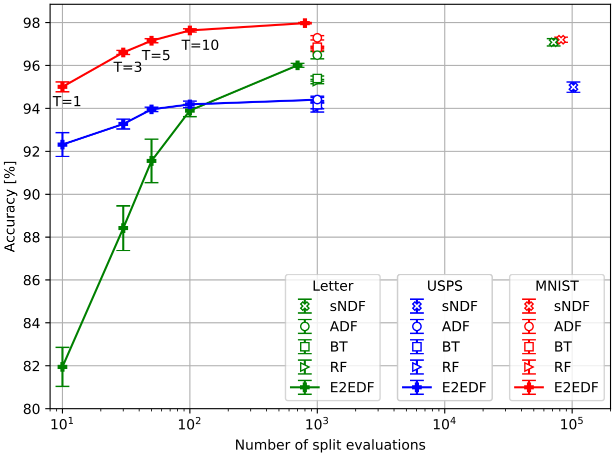


Figure 3.6: Trade-off between computational load and accuracy of oblique decision forest models. The computational load is represented by the number split function evaluations (x-axis, log scale) in the forest required for a single sample at test time. For *RF*, *BT* and *ADF*, the split function evaluations are estimated in favor of those methods. For the *E2EDF* models, the figure shows results with the same number of trees as used by *sNDF*, and additionally includes results with fewer trees in the forest, namely $T \in \{1, 3, 5, 10\}$. *Figure is best viewed in color.*

Results The trade-off in terms of computational load and accuracy of the different decision forest models is shown in figure 3.6. Deterministic *E2EDF* achieves higher average accuracy than *RF* and *BT* on all datasets and outperforms all other methods on *MNIST*. Compared to *ADF*, the results of *E2EDF* are competitive, although relative performance varies between datasets. A possible explanation is that the *ADF* results were obtained using different hyperparameters that allow more and deeper trees, which can lead to significant differences as shown in [32].

On *Letter* and *USPS*, *sNDF* achieves higher accuracy but at several orders of magnitude higher computational cost as it lacks the conditional data flow property. In fact, a single tree in the *sNDF* requires a total of 1023 split evaluations, which is more than for the entire forest models, namely up to 1000 evaluations on USPS. A complete overview of the number split function evaluations per algorithm is given in table 3.3 in the appendix.

Figure 3.6 further presents the impact of using fewer decision trees in the forest model by illustrating the performance of small ensembles ($T \in \{1, 3, 5, 10\}$). On *MNIST* and *USPS*, it is observed that even smaller *E2EDF* ensembles with only $T = 10$ trees already obtains competitive accuracy.

Finally, note that due to the greedy initialization of trees, the actual number of splits is less than the maximum depth would allow. The trained *E2EDF* trees only required on average 755 ± 2 (*Letter*), 370 ± 5 (*USPS*) and 945 ± 1 (*MNIST*) split functions, while the maximum number of split decisions for a tree of depth 10 is $2^{10} - 1 = 1023$. Overall, having fewer trees and fewer decisions in the forest reduces the required number of split evaluations at test time, and thus enables even more efficient inference.

3.4 Conclusions

E2EDT presents a new approach to train deterministic decision trees with gradient-based optimization in an end-to-end manner. The approach uses a probabilistic tree formulation during training to facilitate back-propagation and optimize all splits of a tree jointly.

It was found that by adjusting the steepness of the decision boundaries in an annealing scheme, the method learns increasingly more crisp trees that capture uncertainty as distributions at the leaf nodes, rather than as distributions over multiple paths. The resulting optimized trees are therefore deterministic rather than probabilistic, and run efficiently at test time as only a single path through the tree is evaluated. This approach outperforms previous training algorithms for oblique decision trees. In a forest ensemble, the proposed method shows competitive or superior results to the state-of-the-art *sNDF*, even though the trees only evaluate a fraction of the split functions at test time. Unlike *ADF*, the model is not restricted to only using oblique split functions, thanks to gradient-based optimization. It is shown that it is straightforward to include more complex split features, such as convolutional neural networks, or to add spatial regularization constraints. Another demonstrated benefit is that the learned decision tree can also help interpret how the decision of a visual classification tasks is constructed from a sequence of simpler tests on visual features.

Future work can proceed in various directions. First, alternatives for the annealing scheme could be explored, e.g. the changes in the steepness of tree splits might be adjusted dynamically rather than in a fixed schedule. Second, so far each tree was optimized independently, but potentially optimizing and refining the whole forest jointly could yield further improvements, similar to *ADF* and *sNDF*.

Overall, the presented approach provides high flexibility and the potential for accurate models that maintain interpretability and efficiency due to the conditional data flow.

3.5 References

[1] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulò. Deep neural decision forests. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2015.

[2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[4] Rodrigo Coelho Barros, Márcio Porto Basgalupp, Andre CPLF De Carvalho, and Alex A Freitas. A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):291–312, 2012.

[5] A. Criminisi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis.* Springer, 2013.

[6] Y. Ioannou, D. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown, and A. Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint:1603.01250*, 2016.

[7] Suhang Wang, Charu Aggarwal, and Huan Liu. Using a random forest to inspire a neural network and improving on it. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 1–9. SIAM, 2017.

[8] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[9] Albert Cardona, Stephan Saalfeld, Stephan Preibisch, Benjamin Schmid, Anchi Cheng, Jim Pulokas, Pavel Tomancak, and Volker Hartenstein. An integrated micro- and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy. *PLOS Biology*, 8(10):1–17, 10 2010.

[10] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *arXiv preprint:1405.0312*, 2014.

[11] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[12] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, page 511. IEEE, 2001.

[13] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1297–1304, June 2011.

[14] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1532–1545, 2014.

[15] Le Zhang, Jagannadan Varadarajan, Ponnuthurai Nagaratnam Suganthan, Narendra Ahuja, and Pierre Moulin. Robust visual tracking using oblique random forests. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5589–5598. IEEE, 2017.

[16] Marius Cordts, Timo Rehfeld, Markus Enzweiler, Uwe Franke, and Stefan Roth. Tree-structured models for efficient multi-cue scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1444–1454, 2017.

[17] Apilak Worachartcheewan, Chanin Nantasenamat, Chartchalerm Isarankura-Na-Ayudhya, Phannee Pidetcha, and Virapong Prachayasittikul. Identification of metabolic syndrome using decision tree analysis. *Diabetes Research and Clinical Practice*, 90(1):e15 – e18, 2010.

[18] Orit Pinhas-Hamiel, Uri Hamiel, Yuval Greenfield, Valentina Boyko, Chana Graph-Barel, Marianna Rachmiel, Liat Lerner-Geva, and Brian Reichman. Detecting intentional insulin omission for weight loss in girls with type 1 diabetes mellitus. *International Journal of Eating Disorders*, 46(8):819–825, 2013.

[19] Guan-Mau Huang, Kai-Yao Huang, Tzong-Yi Lee, and Julia Tzu-Ya Weng. An interpretable rule-based diagnostic classification of diabetic nephropathy among type 2 diabetes patients. *BMC Bioinformatics*, 16(1):S5, Jan 2015.

[20] Ruey-Shiang Guh, Tsung-Chieh Jackson Wu, and Shao-Ping Weng. Integrating genetic algorithm and decision tree learning for assistance in predicting in vitro fertilization outcomes. *Expert Systems with Applications*, 38(4):4437 – 4449, 2011.

[21] Barry De Ville. *Decision trees for business intelligence and data mining: Using SAS enterprise miner.* SAS Institute, 2006.

[22] B. H. Menze, B. M. Kelm, D. N. Splitthoff, U. Koethe, and F. A. Hamprecht. *On Oblique Random Forests*, pages 453–469. Springer, 2011.

[23] A. Montillo, J. Tu, J. Shotton, J. Winn, J.E. Iglesias, D.N. Metaxas, and A. Criminisi. Entanglement and differentiable information gain maximization. In *Decision Forests for Computer Vision and Medical Image Analysis*, chapter 19, pages 273–293. Springer, January 2013.

[24] Dmitry Laptev and Joachim M Buhmann. Convolutional decision trees for feature learning and segmentation. In *German Conference on Pattern Recognition*, pages 95–106. Springer, 2014.

[25] Samuel Rota Bulo and Peter Kontschieder. Neural decision forests for semantic image labelling. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2014.

[26] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15 – 17, 1976.

[27] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees.* Chapman & Hall/CRC, 1984.

[28] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 775–781 vol. 2, June 2005.

[29] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1022–1029, June 2009.

[30] Peter Kontschieder, Pushmeet Kohli, Jamie Shotton, and Antonio Criminisi. Geof: Geodesic forests for learning coupled predictors. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2013.

[31] M. Norouzi, M. D. Collins, M. Johnson, D. J. Fleet, and P. Kohli. Efficient non-greedy optimization of decision trees. In *Advances in Neural Information Processing Systems*, December 2015.

[32] S. Schulter, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. Alternating decision forests. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 508–515, June 2013.

[33] A. Suárez and J. F. Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, December 1999.

[34] M. I. Jordan. A statistical approach to decision tree modeling. In *Proceedings of the Seventh Annual Conference on Computational Learning Theory*, COLT '94, pages 13–20, New York, NY, USA, 1994.

[35] I. K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, Oct 1990.

[36] J. Welbl. Casting random forests as artificial neural networks (and profiting from it). In *German Conference on Pattern Recognition*, 2014.

[37] David Richmond, Dagmar Kainmueller, Michael Yang, Eugene Myers, and Carsten Rother. Mapping auto-context decision forests to deep convnets for semantic segmentation. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference*, pages 144.1–144.12. BMVA Press, September 2016.

[38] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint:1711.09784*, 2017.

[39] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for fast test-time prediction. *arXiv preprint:1702.07811*, 2017.

[40] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2363–2372, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[41] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *European Conference on Computer Vision*, September 2018.

[42] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018.

[43] Thomas M Hehn and Fred A Hamprecht. End-to-end learning of deterministic decision trees. In *German Conference on Pattern Recognition*, pages 612–627. Springer, 2018.

[44] J. R. Quinlan. Induction of decision trees. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

[45] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Comput.*, 6(2):181–214, March 1994.

[46] Kenneth Rose, Eitan Gurewitz, and Geoffrey C. Fox. Statistical mechanics and phase transitions in clustering. *Phys. Rev. Lett.*, 65:945–948, Aug 1990.

[47] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[48] Paul H. C. Eilers and Brian D. Marx. Flexible smoothing with B-splines and penalties. *Statistical Science*, 11:89–121, 1996.

[49] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[50] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[52] Marco F Duarte and Yu Hen Hu. Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):826–838, 2004.

[53] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[54] Jung-Ying Wang. Application of support vector machines in bioinformatics. Master's thesis, National Taiwan University, Department of Computer Science and Information Engineering, 2002.

[55] R.-E. Fan and C.-J. Lin. Libsvm data: Classification, regression and multi-label. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`, 2011.

[56] K. V. S. Murthy. *On Growing Better Decision Trees from Data*. PhD thesis, The Johns Hopkins University, 1996.

[57] M. Norouzi, M. D. Collins, D. J. Fleet, and P. Kohli. Co2 forest: Improved random forest by continuous optimization of oblique splits. *arXiv preprint:1506.06155*, 2015.

[58] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint:1708.07747*, 2017.

[59] Peter W Frey and David J Slate. Letter recognition using holland-style adaptive classifiers. *Machine learning*, 6(2):161–182, 1991.

[60] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

3.6 Appendix

Table 3.2 lists the dataset specific parameters used in the experiments of sections 3.3.5 and 3.3.6. These parameters are the same as in [1].

Table 3.3 lists the number of the split function evaluations for the methods discussed in section 3.3.6.

Dataset	Letter	USPS	MNIST
Features	16	16×16	28×28
Classes	26	10	10
No. training samples	16000	7291	60000
No. test samples	4000	2007	10000
No. validation samples	3000	1500	10000
Features per split	8	10×10	15×15
Trees	70	100	80

Table 3.2: Properties of the datasets used in the decision forest experiments, respectively published by [51, 59, 60]. The lower part describes the processing of the data. The table also lists the size of the validation data that was randomly taken from the training data. Features per split indicate the number of features that are randomly sampled to find a split in a tree. In the case of 2D inputs, these random features are 2D patches located randomly at different positions of the input. The final row lists the number of trees in the forest on each dataset, which are equal to the number of trees used in [1].

	Letter	USPS	MNIST
sNDF (single tree)	1023	1023	1023
sNDF (forest)	71610	102300	81840
RF, BT, ADF (forest)	≤ 1000	≤ 1000	≤ 1000
E2EDF (single tree)	≤ 10	≤ 10	≤ 10
E2EDF (forest)	≤ **700**	≤ **1000**	≤ **800**

Table 3.3: Comparison of the computational load of the models evaluated in the experiments. The table shows the number of oblique splits that need to be evaluated in each model per prediction of a single sample. Probabilistic trees in *sNDF* evaluate every split function in the tree and thus requires $2^{D_{\max}} - 1$ dot products per sample and tree. In deterministic trees (*RF*, *BT*, *ADF*, and *E2EDF*), the number of split function evaluations grows linearly with increasing depth. Due to the tree construction, trees may not always reach the maximum depth. Here the table reports the worst case, but assumes the most favorable maximum tree depth for *RF*, *BT*, and *ADF*.

And now for something completely different.

Monty Python

4

Fast and Compact Image Segmentation using Instance Stixels

State-of-the-art stixel methods fuse dense stereo disparity and semantic class information, e.g. from a Convolutional Neural Network (CNN), into a compact representation of driveable space, obstacles, and background. However, they do not explicitly differentiate instances within the same semantic class. Several ways are investigated to augment single-frame stixels with instance information, which can be extracted by a CNN from the RGB image input. As a result, the novel Instance Stixels method efficiently computes stixels that account for boundaries of individual objects, and represents instances as grouped stixels that express connectivity.

Experiments on the Cityscapes dataset demonstrate that including instance information into the stixel computation itself, rather than as a post-processing step, increases the segmentation performance (i.e. Intersection over Union and Average Precision). This holds especially for overlapping objects of the same class. Furthermore, it is shown that the approach is superior in terms of segmentation performance and computational efficiency compared to combining the separate outputs of Semantic Stixels and a state-of-the-art pixel-level CNN. Instance Stixels achieve a processing throughput of 28 frames per second on average for 8 pixel wide stixels on images from the Cityscapes dataset at 1792x784 pixels. The Instance Stixels software is made freely available for non-commercial research purposes.

Parts of this chapter have been published as: Thomas M. Hehn, Julian F.P. Kooij, and Dariu M. Gavrila. “Fast and Compact Image Segmentation using Instance Stixels”, IEEE Transactions on Intelligent Vehicles, 7, 1, p. 45-56, 2022, Institute of Electrical and Electronic Engineers.

4.1 Introduction

Self-driving vehicles require a detailed understanding of their environment in order to react and avoid obstacles as well as to find their path towards their final destination. In particular, stereo vision sensors obtain pixel-wise 3D location information about the surrounding, providing valuable spatial information on nearby free space and obstacles. However, as the processing should be as fast as possible, it is essential to find a compact and efficiently computable representation of sensor measurements which is still capable to provide adequate information about the environment [2, 3]. A common approach is to create a dynamic occupancy grid for sensor fusion [4] and tracking [5], which provides a top-down grid cell representation of occupied space surrounding the ego-vehicle. Still, directly aggregating depth values into an occupancy grid alone would disregard the rich semantic information from the intensity image, and the ability to exploit the local neighborhood to filter noise in the depth image.

A popular alternative in the intelligent vehicles domain is the “stixel” representation, which exploits the image structure to reduce disparity artifacts, and is computed efficiently [6]. By grouping pixels into rectangular, column-wise super-pixels based on the disparity information, stixels reduce the complexity of the stereo information. Later, class label information obtained from deep learning has been incorporated into the stixel computation and representation, so-called Semantic Stixels [1]. Yet, obstacles are still just a loose collection of upright “sticks” on an estimated ground plane, lacking object-level information. For example, the car stixels in the middle row of figure 4.1 do not indicate where one car starts and its neighboring car ends.

This thesis introduces an object-level environment representation extracted from stereo vision data based on stixels. The proposed method improves upon state-of-the-art stixel methods [1, 7] that only consider semantic class information, by adding instance information extracted with a convolutional neural networks (CNN) from the input RGB image. This provides several benefits: First, the stixels boundaries around objects are improved by fusing disparity, semantic, and instance information in the stixel computation. Second, stixels belonging to an object are connected vertically and horizontally (see bottom image of figure 4.1) by clustering them based on semantic and instance information. Third, the processing is more efficient than computing Semantic Stixels [1] and per-pixel instance labels separately.

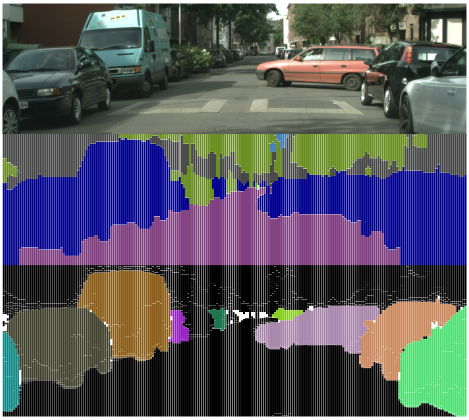


Figure 4.1: Top: Input RGB image (corresponding disparity image not shown). Middle: Semantic Stixels [1] use a semantic segmentation CNN to create a compact stixel representation which accounts for class boundaries (stixel borders: white lines, arbitrary colors per class). Note that a single stixel sometimes covers multiple instances, e.g. multiple cars. Bottom: The proposed *Instance Stixels* algorithm also accounts for instance boundaries using additional information learned by a CNN and clusters stixels into coherent objects (arbitrary colors per instance).

4.2 Related work

The idea of stixels, regarding objects as sticks standing perpendicular on a ground plane, was introduced by [6]. The stixel algorithm has found diverse applications in the autonomous driving domain. Stixels were used as an integral part of the pipeline for the Bertha Benz drive [8]. [9] develop a collision warning system using only stereo-based stixels and [10] used stixels to detect small unknown objects, such as lost cargo. The original idea was further extended in [11]to a multi-layer representation which used a probabilistic approach, i.e. stixels do not need to be connected to the ground plane anymore. In the multi-layer representation, stixels segment the entire image into rectangular super-pixels, classified as ground, object or sky. Additionally, a dynamic programming scheme was presented for efficient real-time computation of stixels. For even faster computation, this dynamic programming scheme was then also implemented for the Graphical Processing Unit (GPU) by [12]. In [13] stixels were compared with other super-pixel algorithms as basis for multi-cue scene labeling.

The general stixel framework offers various possibilities for extensions and modifications. For instance, [14] compared the effects of different methods for initial ground manifold modeling. Driven by the requirements of autonomous driving, [15] applied a Kalman filter to track single stixels. Stixel tracking was then further improved by [16]. Yet, stixels are generally tracked independently and not as parts of an object. In order to obtain object information [17–20] group the Dynamic Stixels based on shape cues and graph cuts and thus rely on tracking Stixels over time. Stixels are also applied in semantic scene segmentation with a more general classes than ground, object, and sky. For this purpose, semantic information can be obtained by using object detectors for suitable classes [21] or Random Decision Forest classifiers [22] and then including that information in the Stixel generation process. [1] extend this idea by incorporating the output of a Fully

Convolutional Neural Network (FCN) in the probabilistic stixel framework. They named their approach Semantic Stixels. Based on Semantic Stixels and focusing on non-flat road scenarios, [7] generalize stixels to also model slanted surfaces, e.g. not strictly perpendicular to the road anymore, including piece-wise linear road surfaces.

Meanwhile, many more deep neural network architectures have been proposed in the computer vision literature to improve classification and image segmentation tasks on a per-pixel basis. For instance, Residual Neural Networks [23] facilitate the training of deeper networks by learning residual functions. Dilated Residual Networks [24] (DRN) improve on this work by maintaining a higher resolution throughout the fully connected network, while working with the same receptive field. As a consequence, they are useful for applications that require spatial reasoning such as object detection or, as in this case, instance segmentation. In order to enforce consistency between semantic and instance segmentation, recently the term panoptic segmentation was introduced in [25] and has led to further improvement in the field [26]. Unfortunately, one cannot treat instance segmentation as a classification problem, as is done for semantic segmentation. The main reason is that the number of instances varies per image, which prohibits a one-to-one mapping of network output channels to instances. Instead of predicting instance labels directly, [27] trains a CNN to map each pixels in an image onto a learned low-dimensional space such that pixels from the same instance map close together. Object masks are then obtained in post-processing by assigning pixels to cluster centers in this space. [28] instead use supervised learning to map pixels to a specific target space, namely the 2D offsets from the given pixel towards its instance's center, and then rely on clustering all pixels into instances. The Box2Pix method [29] uses 2D center offset predictions for instances, but instead of clustering, they are associated with bounding boxes found through a bounding box detection branch. In order to avoid an additional bounding box detection branch, [30] learn a clustering bandwidth and confidence per pixel and thereby speed up the grouping of pixels to instances.

The objective in this thesis is to create efficient stixel representations rather than pixel-accurate instance segmentation in images, and to avoid the overhead of clustering all pixels into instances before reducing them to a compact representation. Still, insights from the work on per-pixel instance segmentation are adopted to improve stixel computation, deal with the unknown number of instances in an image, and enable the clustering of stixels into instances. Building upon a prior conference publication [31], the main contributions are thus summarized as:

- This work presents Instance Stixels, a method to include instance information into stixels computation, which creates better stixels, and allows grouping to instance IDs from a single stereo frame.

- This work investigates three different ways to include the instance information, and show that adding the information into the stixel computation itself results in more accurate in- stance representations than only using it to cluster Seman- tic Stixels or alternatively assigning Semantic Stixels to instances using pixel-based methods. Further, this work compares the trade-off between computation speed and instance segmentation performance for these three variations to showcase the favorable properties of Instance Stixels.

- This work investigates the use of a novel regularizer for Instance Stixels which replaces the former prior term in Stixels. This simplifies the model and leads to improved instance segmentation.
- The entire implementation of the optimized pipeline for Semantic Stixels and Instance Stixels is provided as open- source to the scientific community for non-commercial research purposes.

4.3 Methods

This section will first briefly summarize the original disparity Stixel and Semantic Stixel formulations in subsection 4.3.1. Subsection 4.3.2 then explains how to integrate the instance information from a trained CNN into the stixel computation itself for improved stixel segmentation. Finally, subsection 4.3.3 will discuss how the instance information can be used to cluster stixels belonging to the same object.

The clustering step could be applied to any stixel computation method. This work therefore considers two options:

- Clustering stixels from a standard Semantic Stixels method [1], such that instance offset information is only considered here at this final clustering step. This baseline approach corresponds to the red arrow in Figure 4.2. In the experiments, this combination is referred to as the **Semantic Stixels + Instance** method.
- Clustering based on the novel instance-aware stixels computation from section 4.3.2, see the blue arrow in Figure 4.2. This novel combination is named **Instance Stixels**.

Conceptually, Instance Stixels are a natural extension to Semantic Stixels as they extend disparity and semantic segmentation information with additional instance information to compute a compact representation from a stereo image pair. These stixels also receive an object id which groups them into instances.

4.3.1 Stixels

In the following, an outline of the derivation of the original Stixels and Semantic Stixels framework is presented. For a more detailed derivation, see [32] and [1].

Disparity Stixels

Following the notation of [32], the full stixel segmentation of an image is denoted as $L = \{L_u | 0 \leq u < W\}$ with W being the total number of stixel columns in the image. Thus, given a selected stixel width w, it follows that $W = \frac{\text{image width}}{w}$. The segmentation of column u contains $L_u = \{s_n | 1 \leq n \leq N_u \leq h\}$ contains at least one but at most height h stixels s_n. A stixel $s_n = (v_n^b, v_n^t, c_n, f_n(v))$ is described by the bottom and top rows, respectively v_n^b and v_n^t, that delimit the stixel. Additionally, a stixel is associated with a class $c_n \in \{g, o, s\}$ (i.e. ground, object, sky) and a function f_n which maps each row of the image to an estimated disparity value.

The aim is to find the best stixel segmentation L^* given a measurement (e.g. a disparity image) D, i.e. it maximizes the posterior probability

$$L^* = \arg\max_L p(L|D). \tag{4.1}$$

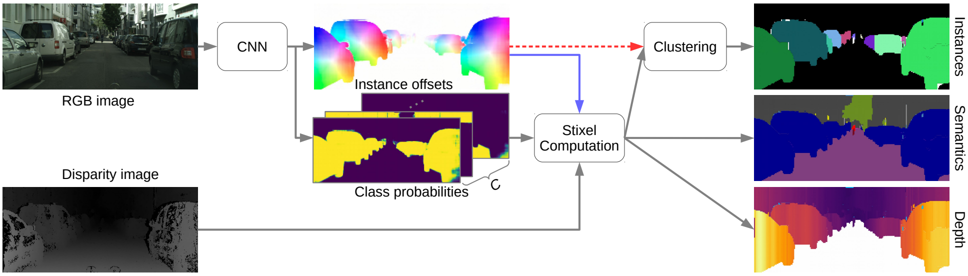


Figure 4.2: Instance stixel pipeline applied to an *RGB* and *disparity* input image pair obtained from a stereo camera. The RGB image is processed by a Convolutional Neural Network (*CNN*) to predict *offsets* to the instance centers (HSV color coded) and per-pixel semantic *class probabilities* (visualized as color gradient). The class probabilities are fused with the disparity input image in the *Stixel computation* to provide a super-pixel representation of the traffic scene, which unifies *Semantics*, *Depth* and additionally *Instance* output (left images). In the baseline algorithm (*Semantic Stixels + Instance*, dashed red arrow) the obtained stixels are clustered based on the instance offsets to assign stixels to instances (not shown). In contrast, the proposed algorithm (*Instance Stixels*, blue arrow) fuses the instance offset information with the other two channels in the Stixel Computation. Subsequently, stixels are also clustered to form instances, but with improved adherence of stixels to instance boundaries (top right image, arbitrary colors).

According to Bayes' rule, this can be rewritten as

$$p(L|D) = \frac{p(D|L)p(L)}{p(D)}. \tag{4.2}$$

Here, the normalization factor $p(D)$, constant in L, can be discarded in the maximization task. Since each column $u \in \{0, ..., W-1\}$ of the image is treated independently, the MAP objective can further be simplified:

$$L^* = \arg\max_{L} \prod_{u=0}^{W-1} p(D_u|L_u)p(L_u). \tag{4.3}$$

Here, $p(D_u|L_u)$ denotes the column's likelihood of the disparity data, and $p(L_u)$ is a prior term modeling the pairwise interaction of vertically adjacent stixels. This is explained in more detail in [11].

Assuming all rows are equally likely to separate two stixels, the column likelihood term can be written as the product of individual terms for N_u stixels, $L_u = \{s_1, ..., s_{N_u}\}$. Since only disparity values of the rows within each stixel contribute to its likelihood, those terms can in turn be factorized over the rows $v_n^b \leq v \leq v_n^t$ of each stixel $n \in \{1, ..., N_u\}$. Hence, the final objective is [32]:

$$L^* = \arg\max_{L} \prod_{u=0}^{W-1} \prod_{n=1}^{N_u} \prod_{v=v_n^b}^{v_n^t} p(d_v|s_n, v)p(L_u). \tag{4.4}$$

Here the term $p(d_v|s_n, v)$ includes different disparity models per geometric class. For sky stixels this model is simple: $f_{\text{sky}}(v) = 0$. The disparity of object stixels is assumed to be normally distributed around the mean stixel disparity $f_{\text{object},n}(v) = \frac{1}{v_n^t+1-v_n^b}\sum_{v'=v_n^b}^{v_n^t} d_v$.

Furthermore, ground stixels rely on a previous estimation of the ground plane parameters α (the slope) and v_{horizon} (horizon estimate in the image), which can be obtained for example from v-disparity [33]. The assumed disparity model for ground stixels $f_{\text{ground}}(v) = \alpha(v_{\text{horizon}} - v)$ is then linear and the same for all columns. Further details can be found in [12].

In practice, the MAP problem (equation 4.4) is written as an energy *minimization* problem by turning the product over probabilities into a sum of negative log probabilities, which is then solved efficiently through Dynamic Programming (DP) [12, 32]. DP will efficiently minimize the energy function

$$E(L_u) = \sum_{n=1}^{N_u} E_p(s_{n-1}, s_n) + E_d(s_n) \tag{4.5}$$

for many stixel hypotheses $L_u = \{s_1, ..., s_{N_u}\}$, which consists of unary terms $E_d(s_n)$ and pairwise energy terms $E_p(s_{n-1}, s_n)$. Intuitively, the unary energy term $E_d(s_n)$ describes the disparity deviation of the disparity models described above. The pairwise term for $n = 1$ reads $E_p(s_0, s_1)$ and is a special case since s_0 is not defined. In all other cases, this pairwise term only evaluates the plausibility of a given stixel segmentation. Note that this in particular means that this pairwise term is independent of the disparity data. These details are omitted here for simplicity [32].

Semantic Stixels

The Semantic Stixels method [1] introduced an additional semantic data term to associate each stixel with one class label $l_n \in \{1, ..., C\}$. Thus, Semantic Stixels are characterized by $s_n = (v_n^b, v_n^t, c_n, f_n(v), l_n)$. First, a semantic segmentation CNN is trained on RGB images with annotated per-pixel class labels. Then, when testing on a test image, the softmax outputs $\sigma(p, l)$ for all semantic classes l of all pixels p are kept (note that in a standard semantic segmentation task, only the class label of the strongest softmax output would be kept). The unary data term $E_d(s_n)$ of the original disparity stixel computation is then replaced by $E_u(s_n) = E_d(s_n) + \omega_l E_l(s_n)$, thereby adding semantic information from the network activations,

$$E_l(s_n) = -\sum_{p \in \mathcal{P}_n} \log \sigma(p, l_n). \tag{4.6}$$

Here $\mathcal{P}_n$ are all pixels in stixel s_n, and ω_l a weight factor.

4.3.2 Instance Stixels

Instance Stixels expand the idea of Semantic Stixels by additionally training a CNN to output a 2D estimation of the position of the instance center for each pixel. This estimation is predicted in image coordinates, as proposed in [28, 29]. More specifically, the CNN predicts 2D offsets $\Omega_p \in \mathbb{R}^2$ (i.e. x and y direction) per pixel, which are relative to the pixel's location in the image. As a consequence, for all pixels p belonging to the same instance j, adding their ground truth offset $\hat{\Omega}_p$ to the pixel location (x_p, y_p) will result in the same instance center location

$$\hat{\boldsymbol{\mu}}_j = \hat{\Omega}_p + (x_p, y_p). \tag{4.7}$$

Such a network is referred to as the *Offset CNN* and an example of its output is visualized in figure 4.2. The ground truth instance centers are defined as the center of mass of the ground truth instance masks. Note that instances are commonly only considered for certain semantic classes, e.g. cars, pedestrians, and bicycles. Let $\mathcal{I} \subset \mathbb{N}$ denote said set of instance relevant classes. For all other classes, the target offset is $(0, 0)$.

Instance Stixels incorporate the Offset CNN prediction into the stixel computation. Let $\boldsymbol{\mu}_p$ denote the instance center estimate obtained from the CNN for some pixel p, and $\bar{\boldsymbol{\mu}}_n = \sum_{p \in \mathcal{P}_n} \boldsymbol{\mu}_p$ the mean over all pixels in an instance stixel $s_n = (v_n^b, v_n^t, c_n, f_n(v), l_n, \bar{\boldsymbol{\mu}}_n)$. The instance term is modeled depending on the center estimates of the pixels and the mean instance center of the current stixel hypothesis s_n:

$$E_i(s_n) = \begin{cases} \sum_{p \in \mathcal{P}_n} \|\boldsymbol{\mu}_p - \bar{\boldsymbol{\mu}}_n\|_2^2, & \text{if } l_n \in \mathcal{I} \\ \sum_{p \in \mathcal{P}_n} \|\boldsymbol{\mu}_p - (x_p, y_p)\|_2^2, & \text{otherwise.} \end{cases} \tag{4.8}$$

In other words, for instance classes, the instance term favors stixels which combine pixels that consistently point to the same instance center. For non-instance classes, i.e. $l_n \notin \mathcal{I}$, offsets $\Omega_p = \boldsymbol{\mu}_p - (x_p, y_p)$ deviating from zero contribute to the instance energy term. Without this, classes with instance information would generally have higher energy and thus be less likely than the non-instance classes.

With the instance energy term, the unary energy becomes

$$E_u(s_n) = \omega_d E_d(s_n) + \omega_s E_s(s_n) + \omega_i E_i(s_n). \tag{4.9}$$

This also introduces weights ω_d and ω_i for the disparity and instance terms for more control on the segmentation.

A useful side effect is that each instance stixel receives a mean estimate of its instance center pixel coordinates, which will be used when clustering stixels into objects, discussed in Section 4.3.3.

4.3.3 Clustering stixels with instance information

The following describes how the output from an Offset CNN can be used in a post-processing step to cluster stixels. Note the favorable computational complexity of grouping a low number of stixels rather than individual pixels as in conventional instance segmentation tasks, e.g. 2000 stixels vs. 1.4M pixels.

First, the per-pixel offsets from the Offset CNN are aggregated into a per-stixel offset estimate by averaging the CNN's predictions over the pixels in the stixel (this is already done for Instance Stixels, as noted in Section 4.3.2). Hence, each stixel is equipped with an estimate of its instance center in 2D image coordinates, as well as a semantic class label.

Then, the estimated instance centers and semantic class prediction are used to group stixels to form instances. Separately for each semantic class, estimated instance centers are clustered. Note that this condition on the semantic class also qualifies Instance Stixels for panoptic segmentation. The final clustering is done using the DBSCAN algorithm [34] as it estimates the number of clusters (i.e. instances) and performs well when the data has dense clusters. DBSCAN has only two parameters: the maximum distance between neighboring data points ε and the minimum size, as in cardinality, γ of the neighborhood of a data point in order to consider this point a core point. Additionally, a size filter parameter is introduced which prevents stixels that are smaller (i.e. cover less rows) than ρ to be considered a core point. This modification prevents small stixels, which lie on the border of two instances, to merge those instances together. Nevertheless, they are assigned to one of those adjacent instances during the clustering procedure.

4.3.4 Unary Regularization

The original Stixel MAP formulation considers a prior term $p(L_u)$ (equation 4.4) which models pairwise interactions of vertically adjacent stixels. The prior term contains detailed models of the expected segmentation. For example, it models the probability of a ground stixel to be found below a sky stixel and vice versa. In the end, the modeled probabilities are usually estimated heuristically.

At the same time, this prior term acts as a regularizer. Without this regularization effect, the resulting stixels tend to be very small simply to fit the data terms as well as possible. In an extreme case with stixels of a width of 1 pixel, this would lead to stixels of also height equal to 1 pixel, which means in the end that each stixel corresponds to a single pixel. Consequently, the stixel segmentation would not be any more compact than the pixel-wise representation.

This modeling of pairwise interactions is especially useful for disparity-based stixels, since there more detailed semantic information is missing. Instance Stixels however do extract semantic and instance information from the RGB images and thus this modeling may be unnecessary. Therefore, this work proposes to replace this prior term by a simple *unary regularization* term

$$E_p(s_n) = \frac{w_R}{v_n^t + 1 - v_n^b} \tag{4.10}$$

which penalizes small stixels. The regularization constant w_R is the only parameter that needs to be determined and is comparable to the different weighting factors of the data terms.

4.4 Implementation

An open-source Instance Stixels implementation is published together with this work, which has been optimized for computational performance on the Cityscapes dataset [35]. As input, it requires the RGB and disparity image of a scene and outputs a set of stixels comprising information about 3D position, semantic class, and instance label. Note that in general, Instance Stixels may also operate only on the RGB image without relying on an disparity image and as a result, do not compute the depth of a stixel.

The first step in the Instance Stixel pipeline as depicted in figure 4.2 is the CNN which predicts for each pixel the probability of each semantic class and the 2D instance center offset vectors in pixels. On Cityscapes, this results in an output depth of $19 + 2 = 21$ channels in total. Any standard semantic segmentation network architecture could be used as the basis for the Semantic Segmentation and Offset CNN by increasing the output depth by 2 channels and training those to predict instance offset vectors. In the implementation, Dilated Residual Networks [24] (DRN) are used as the underlying architecture due to their favorable properties for these tasks, as discussed in Section 4.2. Furthermore, this implementation exploits the fact that, unlike the general method presented in that paper, stixels of a fixed width of 8 pixels are computed and thus removes the upsampling layers in the DRN architecture. The implementation of the DRN is largely based on the PyTorch [36] code provided by the authors of [24]. In order to optimize CNN inference for efficiency, the implementation makes use of mixed precision capabilities of NVIDIA Volta GPUs using the Apex utilities [37] without loss of accuracy.

The second step in the pipeline consists of the actual stixel computation. For this purpose, the implementation extends the open-source disparity Stixel CUDA-implementation introduced in [12]. Amongst other features, such as the computation of Semantic Stixels according to [1] and handling of invalid disparity measurements, the extension comprises the Instance Stixels presented here. Techniques to optimize for efficiency, such as the use of prefix sums (aka. cumulative sums), have been adapted and reused from the original implementation. [12] provides a detailed explanation of those ideas.

Lastly, the stixels are clustered based on the mean instance center estimate. To this end, the GPU-based DBSCAN implementation of cuML [38] is utilized and customized to include the size filter ρ described in section 4.3.3.

In summary, all components are implemented on the GPU which reduces the effective number of required host-device copy operations to two, namely copying the RGB and disparity images to device memory and retrieving the resulting stixel segmentation from device memory. The source code of the implementation is available online[1].

4.5 Experiments

4.5.1 Dataset, metrics, and pre-processing

The computation of stixels require an RGB camera image and the corresponding disparity image obtained from a stereo camera setup. The Cityscapes dataset [35] is used for the experiments, as it consists of challenging traffic scenarios. Further, it provides ground truth annotations for semantic and instance segmentation. The performance on these two tasks is evaluated using the standard Cityscapes metrics [35].

Semantic segmentation performance is measured by the Intersection-over-Union (IoU) $= \frac{TP}{TP+FP+FN}$, where TP, FP, and FN denote the number of true positives, false positives, and false negatives over all pixels in the dataset split. An instance mask is considered correct if the overlap with its ground truth mask surpasses a specific threshold. The Average Precision (AP) corresponds to an average over the precision for multiple thresholds. Average Precision (AP$^{50\%}$) only considers an overlap of at least 50% as true positive. The metric also allows to provide a confidence score for each instance mask. All confidence scores are set to 1 for all compared algorithms.

The disparity images provided in the Cityscapes dataset exhibit noisy regions introduced due to bad disparity measurements at the vertical image edges and the hood of the car. Inaccurate disparity data may harm the performance of disparity-based Stixels. Although Semantic Stixels are already more robust due to the second modality, such effects should be suppressed. Therefore, all images are cropped symmetrically (top: 120px, bottom: 120px, left: 128px, right: 128px) to ensure that the experiments are not influenced by disparity errors. Following [1], the official validation set serves as test set. Therefore, the official training set is split into a separate training *subtrain* and validation set *subtrainval* (validation cities: Hanover, Krefeld, Stuttgart).

[1] Code available at `https://github.com/tudelft-iv/instance-stixels`

4.5.2 Training the CNN

The CNN takes an RGB image as input and predicts the semantic class probabilities and two channels for the offset vectors. Thus, it is a single CNN that provides the output of the Semantic Segmentation and Offset CNN, which were discussed separately in section 4.3. For training, a loss is constructed that allows us to steer the focus between consistency and accuracy of the prediction. Here, this work considers a prediction consistent when all pixels of a ground truth instance mask point towards the *same* 2D position, i.e. all predictions for the instance center (equation 4.7) are the same. Offset accuracy is directly measured by the deviation of each single pixel from the center of mass of the ground truth instance mask. For the predicted offsets, consistency is more important than accuracy. This is best illustrated by an example: consider a single instance in an image and all predicted offsets of that instance do not point to the center of mass of the instance, but instead to a different single point. As a result, this prediction would be consistent, as all offsets point to the same point, and at the same inaccurate as that point does not match the ground truth instance mask's center of mass. Despite the fact that this single point is not the training target, the clustering of this inaccurate, but consistent prediction would work perfectly since all the pixels of the instance are mapped to a single point and thus form a distinct cluster. This observation holds for both the instance-aware stixel computation and the clustering. Nevertheless, enforcing a certain degree of accuracy avoids trivial solutions such as all pixel offsets in the image point to the same single point which would render clustering impossible.

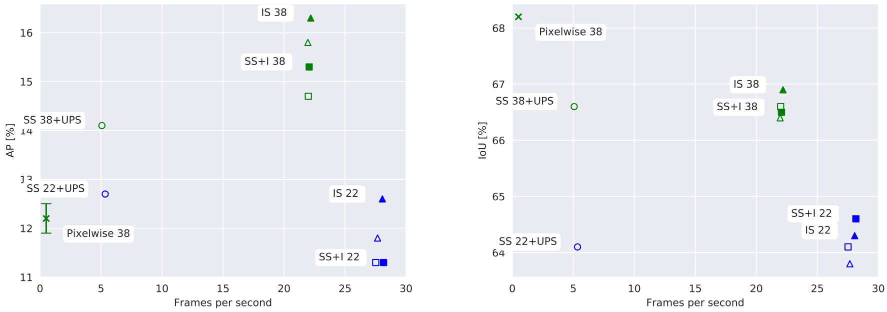
Let $j \in \mathcal{J}$ denote all ground truth instance masks in an image, $\mathcal{P}_j$ all pixels of that mask and $\mathcal{P}_B$ all background pixels which are not part of any instance mask. For all pixels p the CNN predicts an offset Ω_p and using equation 4.7 the predicted center $\boldsymbol{\mu}_p$ can be computed. Further, $\bar{\boldsymbol{\mu}}_j = \frac{1}{|\mathcal{P}_j|} \sum_{p \in \mathcal{P}_j} \boldsymbol{\mu}_p$ denotes the corresponding mean of the predicted centers and $\hat{\boldsymbol{\mu}}_j$ the center of mass of the ground truth instance mask. The proposed offset loss

$$\mathcal{L}_{\mathcal{O}} = \sum_{j \in \mathcal{J}} \left(\frac{\alpha_a}{|\mathcal{P}_j|_1} \sum_{p \in \mathcal{P}_j} ||\boldsymbol{\mu}_p - \hat{\boldsymbol{\mu}}_j||_1 + \frac{\alpha_c}{|\mathcal{P}_j|} \sum_{p \in \mathcal{P}_j} (\boldsymbol{\mu}_p - \bar{\boldsymbol{\mu}}_j)^2 \right) + \frac{\alpha_a}{|\mathcal{P}_B|} \sum_{p \in \mathcal{P}_B} ||\Omega_p||_1 \tag{4.11}$$

thus comprises a consistency term based on $\bar{\boldsymbol{\mu}}_j$, an accuracy term based on $\hat{\boldsymbol{\mu}}_j$ and a background term. The weights α_a and α_c provide the means to find a favorable trade-off between those terms. The full loss $\mathcal{L} = \mathcal{L}_O + \mathcal{L}_S$ further includes a semantic loss $\mathcal{L}_S$, namely a 2D cross-entropy semantic segmentation loss, on the first 19 semantic output channels.

It is important to note that the output (*not* the input) of the CNN is downscaled by a factor 8. The ground truth output is also downscaled by that factor for training. The reason for this is that upscaling, unless nearest neighbor upscaling is used, introduces interpolation errors that result in a smooth transition of the offset vectors between two instances. As a consequence, this would also result in an interpolation of the predicted means of two neighboring instances at pixels close to the borders, which in the end yields worse clustering results. To overcome this issue, this work uses nearest neighbor upscaling when passing the predicted images to the Stixel algorithms. The loss of resolution is compensated by the fact that the Stixels work at a resolution of width 8.

In practice, it was found that training the *drn_d_38* architecture with $\alpha_a = \alpha_c = 1e-4$ and the *drn_d_22* architecture with $\alpha_a = 1e-5$ and $\alpha_c = 1e-4$ worked well. The loss function is



(a) Instance performance vs. frames per second.

(b) Semantic performance vs. frames per second.

Figure 4.3: Trade-off between segmentation performance and processing speed. Each data point represents the average performance of an algorithm on the Cityscapes validation set (all classes, cropped to 1792x784 pixels). The colors indicate different CNN architectures (drn_d_22 or drn_d_38), the symbols differentiate the base algorithm to obtain the instances (*triangle*: Instance Stixels, *square*: Semantic Stixels + Instance, *circle*: Semantic Stixels + UPSNet, *cross*: Pixelwise. If the symbol is filled with color, the unary regularization term was used instead of the pairwise energy term in the stixel computation (section 4.3.4).

optimized using the Adam optimization [39] (learning rate of 0.001, $\beta_1 = 0.9$ and $\beta_2 = 0.999$). Further, zero mean and unit variance normalization based on the training data is applied to the input data and use horizontal flipping to augment the training data. The networks were trained for 500 epochs and with a batch size of 20 images. From these 500 epochs, the best-performing model for each architecture was chosen based on the semantic IoU on the validation set.

4.5.3 Hyperparameter optimization

The stixel algorithms offer several hyperparameters that require tuning: the weighting of the data terms for the Stixel computation ω_d, ω_s and ω_i, as well as the DBSCAN parameters ϵ, γ and ρ. The stixel framework provides more parameters from which the stixel width is set to 8 pixels throughout the experiments. The remaining parameters are set based on recommendations from [32] and [12]. For the *Pixelwise* baseline only ϵ and γ need to be tuned. Additionally, in this baseline, the large number of data points requires the clustering algorithm to process the data in batches which leads to non-deterministic results.

The parameter tuning is performed using Bayesian optimization [40] on the *subtrainval* validation set for 100 iterations. The score is computed as Semantic IoU+1.5·Instance AP. This work weighted Instance AP higher as this is the main focus. The optimization is performed separately for each algorithm unless noted otherwise.

4.5.4 Comparison of algorithmic variations

To analyze the capabilities of the proposed method, four different aspects of computing stixels with instance information are varied.

1. *Pixelwise*: In this baseline setup, the pipeline as shown in figure 4.2 is run entirely

without stixels, by removing the *Stixel Computation.* The semantic class is determined according to the largest class probabilities. During the clustering step, pixels of the same semantic class are clustered based on their predicted instance centers.

2. *SS+UPS*: Represents the combination of state-of-the-art methods to augment stixels with instance information. Based on a separate instance segmentation method, a stixel is assigned to an instance by a majority vote of the pixel-level prediction. For this purpose, this work utilizes the following state-of-the-art methods: a pretrained instance segmentation method called UPSNet [26] and Semantic Stixels [1]. On pixel-level, UPSNet achieves AP performance of 33.1% on the cropped validation set.

3. Semantic Stixels + Instance vs. Instance Stixels (*SS+I* vs. *IS*): Corresponds to setting $\omega_i = 0$, which resembles Semantic Stixels [1]), versus $\omega_i > 0$ in the stixels computation (see equation 4.9).

4. Pairwise vs. unary: Describes whether the stixel computation takes the pairwise term into account or instead regularizes the height of a stixel based on the unary regularization term as described in section 4.3.4.

5. *drn_d_22* vs. *drn_d_38*: Denotes the different base architectures of the Dilated Residual Network [24] used to predict semantic probabilities and instance offsets. The architecture *drn_d_38* is deeper and requires more memory.

Due to the fact that the *subtrainval* set overlaps with the training set of the UPSNet, the *subtrainval* set cannot be used for hyperparameter tuning. Hence, the same weights are used for the Semantic Stixels of *SS+UPS* as the corresponding *SS+I.*

Processing speed vs. Segmentation Performance

The following compares the different stixel methods for instance segmentation regarding the trade-off of segmentation performance and processing speed. The main indicators for segmentation performance are the instance AP and the semantic IoU as described in section 4.5.1. Processing speed is measured as the number of frames the pipeline can process per second. Here, to compute the frames per second the processing time of the frames in the validation set is averaged, which takes into account the processing time of all three modules (*CNN*, *Stixel Computation* and *Clustering*, see figure 4.2), but neglects data loading and visualization. All frames are processed sequentially on an NVIDIA Titan V GPU.

Figure 4.3 illustrates the trade-off between processing speed and instance as well as semantic performance in a compact manner. Table 4.1 extends the figure by providing further segmentation metrics and also the complexity of the image representation as the average number of stixels per frame on the official Cityscapes validation set.

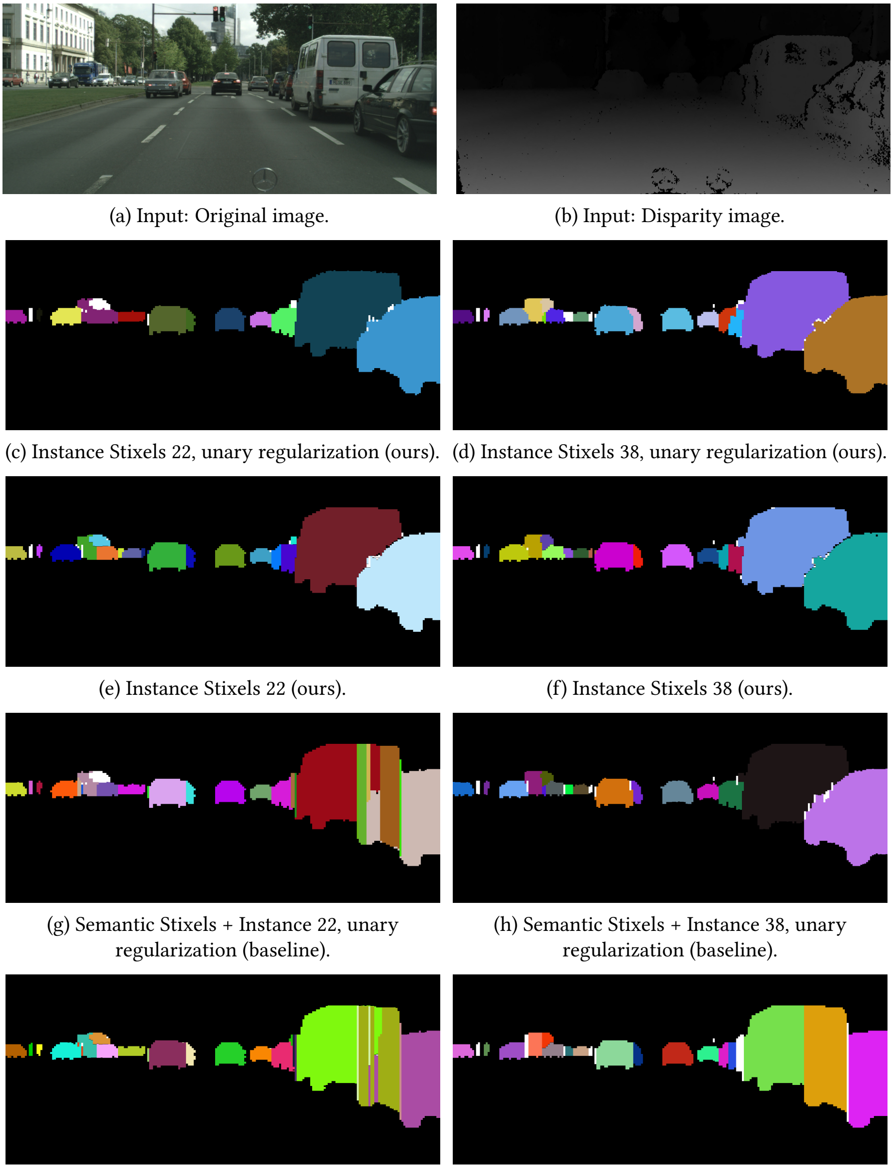
In terms of segmentation performance, the illustrations show that the choice of the network architecture of the CNN, indicated by the color of the points, has the most prominent effect (green: *drn_d_38* and blue: *drn_d_22*). For both segmentation metrics, even the best algorithm based on *drn_d_22* performing worse than the worst stixel algorithm based on *drn_d_38.* Within the same architecture, however, Instance Stixels (*IS*) generally perform better than Semantic Stixels + Instance (*SS+I*) in terms of instance AP, but not always in terms of semantic IoU. Further, for both algorithms (*IS* and *SS+I*), using the unary regularization term (filled symbols) surpasses its pairwise counterpart (non-filled symbols) or at least remains on par. Interestingly, the CNN architecture choice also affects the comparison in instance AP of Instance Stixels and Semantic Stixels + UPSNet (*SS+UPS*). For *drn_d_22*, *IS 22* with unary regularization achieves similar instance AP as *SS+UPS 22*. For *drn_d_38*, *SS+UPS* obtains the worst instance AP of all stixel methods. The semantic IoU of *SS+UPS* is limited by its *SS+I* counterpart by construction. Overall, Instance Stixels based on the *drn_d_38* architecture and using the unary regularization outperforms all other stixel-based algorithms in both segmentation metrics. Only the *Pixelwise* algorithm surpasses this performance in the semantic IoU, but not the instance AP. The same observations generally also hold for the extended instance and semantic segmentation metrics $\mathrm{AP}^{50\%}$ and the category IoU [35] as listed in table 4.1.

To a certain degree, segmentation performance comes at a trade-off regarding processing speed. Notably, the speed is mainly determined by the choice of the CNN as well. The *Pixelwise* pipeline is by far the slowest algorithm for these tasks at only 0.5 frames per second. Stixel methods based on *drn_d_38* are favorable compared to methods relying on UPSNet, but not as fast as methods based on *drn_d_22*. Among the same architecture, the differences in processing speed are only minor and are listed in table 4.1. Additional analysis showed that the processing speed is steady over all frames, regardless of the number of instances or stixels in an image. The complexity of the segmentation, quantified by the average number of stixels per frame, varies between algorithms exhibiting no obvious correlation. Among the Instance Stixels the highest average number stixels per frame is at most 2673.

Qualitative analysis

The consequences of the different algorithm variations as described in section 4.5.4 are depicted in figure 4.4 when applied on a real traffic scene image from the *subtrainval* set. Figures 4.4a and 4.4b show the input data. Instance segmentation results in the left column (4.4 (c),(e),(g) and (i)) based on the drn_d_22 architecture show in general more errors than in the right column (4.4 (d),(f),(h) and (j)) which is based on the drn_d_38 architecture. Especially figures (g),(i) and (j) show several stixels which overlap two instances.

Figure 4.5 visualizes the full results (3D position, semantic and instance segmentation) of Instance Stixels (*drn_d_38*, unary regularization) on three scenes (columns). Based on the input RGB images (top row), the CNN predicts the offset vectors (center rows). The offset vectors are visualized in HSV color space, where the hue indicates the direction and the saturation indicates the magnitude of the offsets. The fourth row shows the segmentation of the scenes. The overlaid colors illustrate the semantic class per pixel, whereas the white contours around objects mark the borders of instances. The bottom row shows top down views of the scene based on the per stixel disparity information and location within the input image. In these illustrations, the road and the sidewalk are illustrated as polygons. Their boundaries are based on the ground plane estimation. Sky stixels are discarded and non-instance stixels are drawn as circles. Their radius indicates the size of the respective stixel in the image. Stixels of the same instance are connected by a line. Per column, only the stixel that are closest to the ego-vehicle are connected. As a result of the instance segmentation, outliers can also be filtered. Specifically, stixels that are further than 3 meters away from the mean top down position of the instance are not included. Also, the stixel artifacts of the Mercedes-Benz Star are removed from the top down view based on their position in the image.



(a) Input: Original image.

(b) Input: Disparity image.

(c) Instance Stixels 22, unary regularization (ours).

(d) Instance Stixels 38, unary regularization (ours).

(e) Instance Stixels 22 (ours).

(f) Instance Stixels 38 (ours).

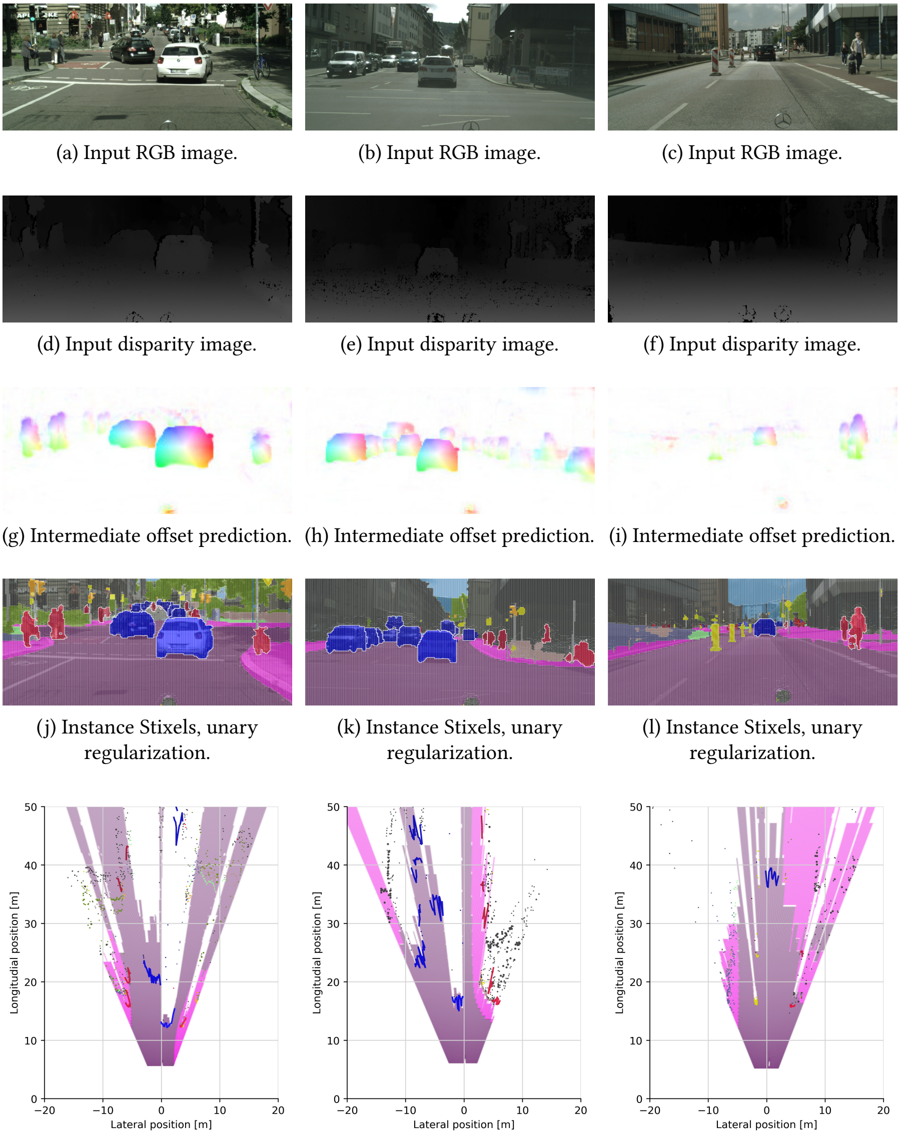
(g) Semantic Stixels + Instance 22, unary regularization (baseline).

(h) Semantic Stixels + Instance 38, unary regularization (baseline).

(i) Semantic Stixels + Instance 22 (baseline).

(j) Semantic Stixels + Instance 38 (baseline).

Figure 4.4: Qualitative analysis of instance segmentation results from *Semantic Stixels + Instance* (baseline) and *Instance Stixels* (proposed algorithm) using different architectures as well as comparing the pairwise energy term and the unary regularization. Figure (a) and (b): The input RGB and disparity image. Below, the left column shows instance segmentation results obtained using the drn_d_22 architecture as the basis for the CNN. Likewise, the right column Instances are indicated by arbitrary colors. White areas denote stixels that cannot be assigned to specific instances, but their predicted semantic class is an instance class. Black areas indicate that the predicted semantic class is not an instance class.



(a) Input RGB image. (b) Input RGB image. (c) Input RGB image.

(d) Input disparity image. (e) Input disparity image. (f) Input disparity image.

(g) Intermediate offset prediction. (h) Intermediate offset prediction. (i) Intermediate offset prediction.

(j) Instance Stixels, unary regularization. (k) Instance Stixels, unary regularization. (l) Instance Stixels, unary regularization.

(m) Instance Stixels 38, unary regularization. (n) Instance Stixels 38, unary regularization. (o) Instance Stixels 38, unary regularization.

Figure 4.5: Illustration of stixel segmentations including spatial top down view of the scene. Each column is a separate scene, the top two rows show the corresponding inputs, the center row shows the offsets predicted by the CNN and the bottom two rows visualize the output of Instance Stixels (*drn_d_38*, unary regularization). In the third row from the top, the overlaid color indicates the semantic class of a stixel, whereas the white contour around objects indicates the segmented instances. The last row shows the top view of the scene. Instances are visualized as lines, road and sidewalk stixels are plotted as polygons based on the obtained ground plane estimation. Stixels of class sky are discarded in this illustration. All remaining stixels (e.g. buildings and poles) as points and their radius indicates the stixels size.

	CNN	Unary regularization	AP [%]	AP^{50} [%]	IoU [%]	cat IoU [%]	FPS	Avg. number of stixels
Pixelwise	drn_d_38	-	12.5 ± 0.3*	25.3 ± 0.7*	**68.2**	**85.0**	0.5	1404928**
SS+I	drn_d_22	-	11.3	25.4	64.1	80.2	27.5	2095
SS+I	drn_d_22	✓	11.3	25.7	64.6	81.8	**28.2**	1765
SS+I	drn_d_38	-	14.7	30.3	66.6	80.8	22.0	**1270**
SS+I	drn_d_38	✓	15.3	31.6	66.5	81.3	22.1	4795
SS+UPS	drn_d_22	-	12.7	28.5	64.1	80.2	5.3	2095
SS+UPS	drn_d_38	-	14.1	30.8	66.6	80.8	5.1	**1270**
IS	drn_d_22	-	11.8	26.3	63.8	79.9	27.7	1384
IS	drn_d_22	✓	12.6	26.8	64.3	81.1	**28.1**	2673
IS	drn_d_38	-	15.8	31.1	66.4	80.5	22.0	1421
IS	drn_d_38	✓	**16.3**	**32.4**	66.9	81.9	22.2	2278

Table 4.1: Performance of the Pixelwise baseline and different variations of stixel algorithms that provide instance segmentation (rows) with respect to various metrics (columns). Results are computed on the Cityscapes validation set (all classes, cropped to 1792x784 pixels). Best results per metric are highlighted in **boldface**. * The results of the Pixelwise baseline were averaged over three runs and are reported with the corresponding standard deviation. All other algorithms are consistent over multiple runs. ** The resulting segmentation is represented as $1792 \cdot 784 = 1404928$ pixels, since no stixels are involved here.

4.6 Discussion

Results presented in section 4.5.4 show that adding the instance term, that distinguishes *IS* and *SS+I*, increases instance AP. Minor drawbacks in terms of semantic IoU may be due to hyperparameter optimization which values instance AP more than semantic IoU. Despite the increased instance AP, the segmentation for far away objects (e.g. the truck in the left-hand part of figure 4.4a) and tightly overlapping objects (e.g. pedestrians in the left-hand part figure 4.5a) remain challenging. Overall, the choice of the CNN appears to be more important to the segmentation than the effect of the instance term. Notably, using a state-of-the-art pixelwise instance segmentation CNN, such as UPSNet [26], and combining it with Semantic Stixels falls behind significantly in terms of processing speed. UPSNet requires on average 0.15 seconds of processing time per frame which on its own yields only 6.6 frames per second. Compared to the pixelwise UPSNet result, the instance AP of *SS + UPSNet* has decreased by more than 50%. A drop in overall accuracy is likely, since the stixels group pixels along predefined coarse column borders and thus inherently decrease the granularity of the prediction. Further, *SS* do not consider the instance term introduced for *IS*, thus stixels may overlap two different instances. UPSNet cannot change this afterward which leads to worse performance. Lastly, a pixelwise clustering approach shows weak instance segmentation performance at a runtime of 0.5 FPS that is dominated by the clustering algorithm suffering from the large amount of points.

The benefits of a purely stixels-based instance segmentation method however is not only observed in processing speed, but also in terms of segmentation complexity. Pixelwise methods result in more than 1.4 million independent predictions. The Instance Stixels on average require between 1384 and 2673 stixels per frame to describe the same amount of pixels. This means Instance Stixels reduce the complexity of the representation by factors between 525.6 - 1015.1.

Aside from image segmentation, Instance Stixels provide position estimates in 3D space. As a result, top down views of a scene can be extracted, similar to a grid map. In contrast to a grid map, the proposed representation is continuous and does not discretize 3D space. In this top down representation, imperfect disparity measurements, become apparent, for example in that the back of cars do not appear as straight lines. Further, it also shows the inaccuracies of the ground plane and horizon estimation, which is here based on v-disparity [33]. In the stixel model, stixels above the horizon cannot be classified as ground. This leads to artifacts as seen on the road behind the two cars in figure 4.5j. As the ground plane estimation is crude, the polygons of the road stixels overlap sometimes with stixels of obstacles. Combining Instance Stixels with LiDAR measurements as shown in [41] may improve both, the depth estimation and the ground plane estimation. As this is an orthogonal approach, not related to instance segmentation, this work made use of the object-based representation to filter outliers in the depth measurements of a single object.

The rich information about both, the static and dynamic surroundings, contained in Instance Stixels can benefit subsequent utilization in an autonomous driving pipeline. For example, Instance Stixels provide a rich and efficient representation for path planning, object tracking, and mapping.

4.7 Conclusions

This thesis introduced Instance Stixels to improve stixel segmentation by considering instance information from a CNN, and performing a subsequent stixel clustering step. The experiments showed multiple benefits of including the instance information already in the segmentation step, opposed to only clustering Semantic Stixels. First, quantitative and qualitative analyses show that Instance Stixels adhere better to object boundaries. Second, Instance Stixels provide more accurate instance segmentation than Semantic Stixels augmented with instance information from a pixel-level instance segmentation network. Third, Instance stixels still preserve the favorable stixel characteristics in terms of compactness of the segmentation representation (on average less than 2673 stixels per image) and computational efficiency (up to 28 FPS at a resolution of 1792x784). In future work, the integration of additional sensor modalities as shown in [41] and temporal information to enforce consistency are potential research directions.

4.8 References

[1] Lukas Schneider, Marius Cordts, Timo Rehfeld, David Pfeiffer, Markus Enzweiler, Uwe Franke, Marc Pollefeys, and Stefan Roth. Semantic Stixels: Depth is not enough. In *IEEE Intelligent Vehicles Symposium*, pages 110–117, 2016.

[2] Sayanan Sivaraman and Mohan Manubhai Trivedi. Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis. *IEEE Transactions on Intelligent Transportation Systems*, 14(4):1773–1795, 2013.

[3] M. Braun, S. Krebs, F. Flohr, and D. M. Gavrila. Eurocity persons: A novel benchmark for person detection in traffic scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1844–1861, Aug 2019.

[4] Dominik Nuss, Ting Yuan, Gunther Krehl, Manuel Stübler, Stephan Reuter, and Klaus Dietmayer. Fusion of laser and radar sensor data with a sequential monte carlo bayesian occupancy filter. In *IEEE Intelligent Vehicles Symposium*, pages 1074–1081, 2015.

[5] Radu Danescu, Florin Oniga, and Sergiu Nedevschi. Modeling and tracking the driving environment with a particle-based occupancy grid. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1331–1342, 2011.

[6] Hernán Badino, Uwe Franke, and David Pfeiffer. The stixel world - a compact medium level representation of the 3d-world. In *31st DAGM Symposium on Pattern Recognition*, pages 51–60, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] Daniel Hernandez-Juarez, Lukas Schneider, Antonio Espinosa, David Vázquez, Antonio M. López, Uwe Franke, Marc Pollefeys, and Juan C. Moure. Slanted stixels: Representing san francisco's steepest streets. *Proceedings of the British Machine Vision Conference*, 2018.

[8] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, ..., and E. Zeeb. Making bertha drive - an autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, 6(2):8–20, Summer 2014.

[9] Willem Sanberg, Gijs Dubbelman, and Peter de With. From stixels to asteroids: Towards a collision warning system using stereo vision. In *IS&T International Symposium on Electronic Imaging*, 2019.

[10] Sebastian Ramos, Stefan Gehrig, Peter Pinggera, Uwe Franke, and Carsten Rother. Detecting unexpected obstacles for self-driving cars: Fusing deep learning and geometric modeling. *IEEE Intelligent Vehicles Symposium*, pages 1025–1032, 2017.

[11] David Pfeiffer and Uwe Franke. Towards a Global Optimal Multi-Layer Stixel Representation of Dense 3D Data. *Proceedings of the British Machine Vision Conference*, pages 51.1–51.12, 2011.

[12] Daniel Hernandez-Juarez, Antonio Espinosa, Juan C. Moure, David Vázquez, and Antonio M. López. GPU-Accelerated real-Time stixel computation. *IEEE Winter Conference on Applications of Computer Vision*, pages 1054–1062, 2017.

[13] Marius Cordts, Timo Rehfeld, Markus Enzweiler, Uwe Franke, and Stefan Roth. Tree-structured models for efficient multi-cue scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1444–1454, 2017.

[14] N. H. Saleem, H. Chien, M. Rezaei, and R. Klette. Effects of ground manifold modeling on the accuracy of stixel calculations. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3675–3687, Oct 2019.

[15] D. Pfeiffer and U. Franke. Efficient representation of traffic scenes by means of dynamic stixels. In *IEEE Intelligent Vehicles Symposium*, pages 217–224, June 2010.

[16] Bertan Günyel, Rodrigo Benenson, Radu Timofte, and Luc Van Gool. Stixels Motion Estimation without Optical Flow Computation. In *European Conference on Computer Vision*, pages 528–539, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[17] Friedrich Erbs, Alexander Barth, and Uwe Franke. Moving vehicle detection by optimal segmentation of the dynamic stixel world. In *IEEE Intelligent Vehicles Symposium*, pages 951–956, 2011.

[18] Friedrich Erbs, Beate Schwarz, and Uwe Franke. Stixmentation-probabilistic stixel based traffic scene labeling. In *Proceedings of the British Machine Vision Conference*, pages 1–12, 2012.

[19] Friedrich Erbs, Beate Schwarz, and Uwe Franke. From stixels to objects - a conditional random field based approach. In *IEEE Intelligent Vehicles Symposium*, pages 586–591, 2013.

[20] Friedrich Erbs, Andreas Witte, Timo Scharwächter, Rudolf Mester, and Uwe Franke. Spider-based stixel object segmentation. In *IEEE Intelligent Vehicles Symposium*, pages 906–911, 2014.

[21] Marius Cordts, Lukas Schneider, Markus Enzweiler, Uwe Franke, and Stefan Roth. Object-level priors for stixel generation. In *German Conference on Pattern Recognition*, pages 172–183. Springer, 2014.

[22] T. Scharwächter and U. Franke. Low-level fusion of color, texture and depth for robust road scene understanding. In *IEEE Intelligent Vehicles Symposium*, pages 599–604, June 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.

[24] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[25] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollar. Panoptic segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2019.

[26] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. Upsnet: A unified panoptic segmentation network. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[27] Bert De Brabandere, Davy Neven, and Luc Van Gool. Semantic instance segmentation with a discriminative loss function. In *Deep Learning for Robotic Vision, workshop at CVPR 2017*, pages 1–2. CVPR, 2017.

[28] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[29] Jonas Uhrig, E Rehder, B Fröhlich, U Franke, and Thomas Brox. Box2pix: Single-shot instance segmentation by assigning pixels to object boxes. In *IEEE Intelligent Vehicles Symposium*, 2018.

[30] Davy Neven, Bert De Brabandere, Marc Proesmans, and Luc Van Gool. Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2019.

[31] T. M. Hehn, J. F. P. Kooij, and D. M. Gavrila. Instance stixels: Segmenting and grouping stixels into objects. In *IEEE Intelligent Vehicles Symposium*, pages 2542–2549, June 2019.

[32] David Pfeiffer. *The Stixel World.* PhD thesis, Humboldt-Universität zu Berlin, 2012.

[33] R. Labayrade, D. Aubert, and J.-P. Tarel. Real time obstacle detection in stereovision on non flat road geometry through v-disparity representation. In *IEEE Intelligent Vehicles Symposium*, pages 646–651, 2002.

[34] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[35] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, ..., and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[36] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[37] NVIDIA Corporation. Apex: A pytorch extension: Tools for easy mixed precision and distributed training in pytorch, visited on: 2019-11-26.

[38] RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018.

[39] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[40] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[41] Florian Piewak, Peter Pinggera, Markus Enzweiler, David Pfeiffer, and Marius Zöllner. Improved semantic stixels via multimodal sensor fusion. In *German Conference on Pattern Recognition*, pages 447–458. Springer, 2018.

And now for something completely different.

Monty Python

5

How do Cross-View and Cross-Modal Alignment Affect Representations in Contrastive Learning?

Various state-of-the-art self-supervised visual representation learning approaches take advantage of data from multiple sensors by aligning the feature representations across views and/or modalities. This work investigates how aligning representations affects the visual features obtained from cross-view and cross-modal contrastive learning on images and point clouds.

On five real-world datasets and on five tasks, 108 models based on four pretraining variations are trained and evaluated. The results show that cross-modal representation alignment discards complementary visual information, such as color and texture, and instead emphasizes redundant depth cues. The depth cues obtained from pretraining improve downstream depth prediction performance. Also overall, cross-modal alignment leads to more robust encoders than pretraining by cross-view alignment, especially on depth prediction, instance segmentation, and object detection.

5.1 Introduction

Pretraining of neural networks has been an established tool in computer vision for several years [2]. It allows to successfully finetune neural networks on new tasks with fewer iterations and less annotated data [3]. Commonly, models are initialized using weights pretrained for image classification on the ImageNet dataset [4], but recently, self-supervised approaches that do not rely on manual annotations have outperformed ImageNet pretraining [5].

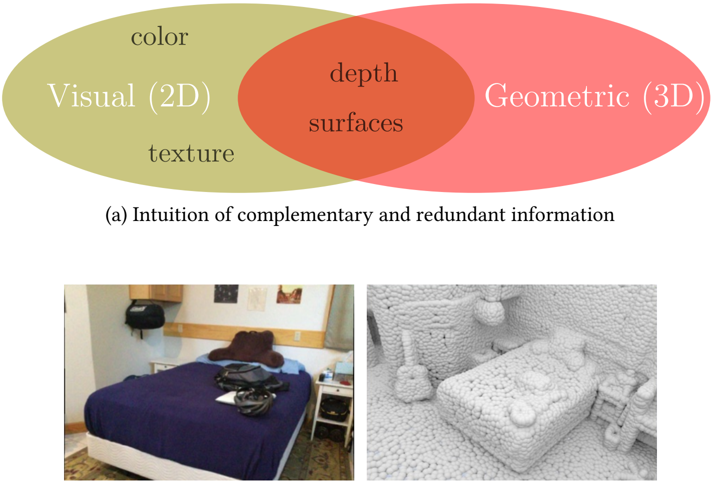
In the realm of self-supervised representation learning, contrastive learning [6] has become a popular approach for visual representation learning. In contrastive learning, the model learns to distinguish different variations of the same instance (e.g. crops of a single image) from all other instances. These variations can be created artificially or by using natural data correspondences, such as images and text [7] or from multiple viewpoints [1, 8].

A common objective is to enforce similarity between features across views or sensing modalities. This is called *representation alignment* [9]. While severe failure cases of such representation alignment have been discussed [9, 10], the effects on the representation itself were not studied, as of yet.

Figure 5.1 shows a conceptual illustration of the information available in 2D image data and 3D point clouds in the ScanNet dataset [11]. Some information, such as about depth and surfaces, is available in both modalities. Although this information may be more detailed and more accurate in one modality than in the other, both modalities can express such properties to a certain degree. In the context of sensor fusion, this is typically called *redundant* information. Other information is exclusive to a modality, which is called *complementary* information. 3D point clouds, for example, are generally color- and textureless.

Intuitively, a representation which discards complementary information is less expressive than one that includes redundant and complementary information. On the one hand, discarding texture and color information, for example, leads to a less complete visual representation. On the other hand, ImageNet pretrained models were found to be texture-biased and by increasing their shape-bias the performance on tasks such as object detection could be improved [12]. So what is the influence of complementary and redundant information on the visual representation quality?

This work studies how complementary and redundant information affect visual representation learning when exploiting cross-view and cross-modal representation alignment, and how this relates to the performance of a finetuned model on a transfer learning downstream task. For this purpose, Pri3D [1] is used, a method that uses 3D information of a scene to enable cross-view and cross-modal representation alignment for visual representations learning.



(a) Intuition of complementary and redundant information

(b) ScanNet image*

(c) ScanNet point cloud*

Figure 5.1: Redundant information is shared across modalities, while complementary information is exclusive to a modality. This work shows that cross-modal *visual* representation learning predominantly encodes redundant information of 2D and 3D data. *Images (b) and (c) were taken from [1].

5.2 Related work

Self-supervised representation learning has recently surpassed supervised pretraining on ImageNet [5]. Various approaches have been explored to learn a global feature vector per image, such as contrastive learning [5, 6] and methods that do not require negative samples during training [13, 14]. Since many computer vision tasks require per-pixel features, several works focused on learning dense representations [15, 16]. Inspired by this success for visual representations, contrastive learning has also been applied to 3D point cloud data [17–19].

Several approaches have explored cross-modal and cross-view visual representation learning. While some propose a single model [20] for multiple modalities, this work discusses training a vision-only model using cross-modal and cross-view input. To this end, some have matched images and natural language to improve the learned visual representation [7, 21], whereas others use 3D signals [1, 22, 23], or match representations of different views of a scene [1, 8].

While cross-modal feature spaces are a common technique [1, 19, 24–27], they can introduce several caveats. Especially when dealing with partially incomplete or corrupted data, representation alignment may hinder effective learning for multi-view clustering [9]. In domain adaptation, avoiding representation alignment across modalities has also proven to be beneficial to prevent discarding complementary sensor information [10]. In representation learning, cross-modal training can even lead to an emphasis on a single strong modality and potentially harm the overall performance [28].

The goal of this work is to understand the effects of cross-view and cross-modal representation alignment on contrastive learning in more detail. Specifically, corrupted or incomplete sensor data is not considered, but it is investigated how redundant and complementary information influence the learned representations, and whether similar effects as observed for shape-biased vs texture-biased networks [12] can be observed. This empirical study is based on Pri3D [1] as it uses both cross-view and cross-modal representation alignment for contrastive learning, as well as their combination. As the main contributions, this work:

1. assesses how cross-view and cross-modal representation alignment affect the complementary and redundant information encoded in the learned visual per-pixel representations,

2. evaluates the transfer learning performance in different settings on various tasks and datasets to investigate the robustness of cross-view and cross-modal alignment.

5.3 Methods

2D images and 3D point clouds have low-level redundant and complementary features, as illustrated in figure 5.1. Therefore, these modalities lend themselves to study cross-view and cross-modal representation alignment. The presented analysis of alignment in representation learning is based on Pri3D [1]. Pri3D uses 3D information for 2D visual representation learning. It employs a self-supervised contrastive learning approach that does not require any human annotation, but only RGB images and registered point clouds that can be obtained, for example, from an RGB-D dataset. This section summarizes the two contrastive losses of Pri3D and discusses how the feature spaces of the losses can be separated.

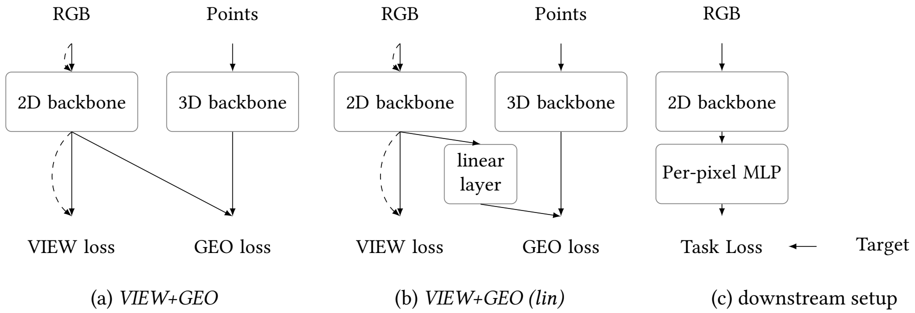
5.3.1 Pri3D: contrastive losses

Pri3D finds pairs of pixels and elements of a regular grid in 3D space, so-called voxels, for contrastive learning. Pixels and voxels are matched by their distance in 3D space to define two losses. One loss requires pairs of pixels across views, whereas the other loss is based on pairs of pixels and 3D points across modalities. Since Pri3D matches pixels and voxels based on their 3D positions, it is relatively robust with respect to incomplete sensor data.

Cross-view contrastive loss (*VIEW*) This contrastive loss matches pixels across different views of the same scene. It requires two RGB-D images from the same scene and the corresponding relative camera poses and calibration. Once two pixels from two different views A and B are within a 2 cm distance, they are considered a positive pair in the sense of contrastive learning [6]. Only pixels that are part of a positive pair are used in this loss, as the negative pairs are also constructed from these pixels. To form the negative pairs, each pixel a of view A of a positive pair (a, b) is paired with all pixels k of view B which are part of a positive pair but not $k = b$.

For each image x, an encoder-decoder style network f predicts a set feature vectors $f(x)$, and $\boldsymbol{f}_a$ denotes the L2-normalized feature vector of a pixel a. Using the set of positive pairs M, Pri3D employs a PointInfoNCE loss [18]

$$\mathcal{L}_p = -\sum_{(a,b)\in M} \log \frac{\exp(\boldsymbol{f}_a \cdot \boldsymbol{f}_b/\tau)}{\sum_{(\cdot,k)\in M} \exp(\boldsymbol{f}_a \cdot \boldsymbol{f}_k/\tau)}, \tag{5.1}$$



(a) *VIEW+GEO* (b) *VIEW+GEO (lin)* (c) downstream setup

Figure 5.2: During pretraining, the combination *VIEW+GEO* (a) optimizes the same features in the *VIEW* and *GEO* loss, while the linear layer in *VIEW+GEO (lin)* (b) separates the feature spaces of the losses. The dashed arrows indicate a second overlapping view. Only the 2D backbone performance is evaluated (c).

where τ is a temperature hyperparameter. This cross-view loss is called the *VIEW* loss, following the convention of Pri3D. Compared to other visual representation learning approaches, *VIEW* does not rely on strong augmentations that may corrupt color information [6].

Cross-modal contrastive loss (*GEO*) Unlike the *VIEW* loss, the pairs for the *GEO* loss do not consist of two pixels from different views, but instead of a pixel and a voxel. The positive pairs are again pixels and voxels within 2 cm distance in 3D space. The negative pairs are all other non-matching pixels and voxels pairs, similar to the *VIEW* loss.

The voxel is associated with a feature vector obtained from a dense 3D feature encoder. This dense 3D feature encoder is jointly learned with the dense 2D feature encoder. The loss function (equation (5.1)) remains the same with $\boldsymbol{f}_a$ and $\boldsymbol{f}_b$ now being feature vectors from the 2D and 3D feature encoders, respectively. Thus, this cross-modal loss enforces the features to be similar across the two modalities and is called the *GEO* loss, following the convention of Pri3D.

5.3.2 Pri3D: representation space separation

In Pri3D [1], the *GEO* and the *VIEW* loss use the same visual feature vector for a single pixel. Thus, the combination *VIEW+GEO* enforces mutual alignment of cross-view *and* cross-modal representations. Interestingly, in the implementation published by the original Pri3D authors[29], the feature spaces of both losses are separated by a linear layer preceding the *GEO* loss (see figure 5.2). Potentially, the original purpose of this layer was merely to match the feature dimensions of the 2D and 3D backbone (e.g. for ResNet50 2D backbone this layer projects 128 to 96 channels). Yet, this separation of feature spaces relaxes the alignment of the cross-view with the cross-modal feature representation. The feature vector of a pixel in the *GEO* loss is now computed by a linear function on the feature vector of the same pixel in the *VIEW* loss, thus the two feature vectors are not the same anymore. The cross-view and the cross-modal representations are not directly aligned. Since the effect of this linear layer was not described in the literature, both variants are included in the experiments.

In the following, this variation is distinguished by adding *(lin)* to models that include the linear layer that separates the feature representations. These and other differences to the original Pri3D paper are described in more detail in the supplementary material.

5.4 Experiments

The experiments are setup to examine how cross-view and cross-modal representation alignment in Pri3D influence the encoded complementary and redundant information and how possible differences in the representations affect downstream transfer learning performance.

5.4.1 Setup

The experiments consist of two main steps: the models are first pretrained based on the variations of Pri3D (see section 5.3). Afterward, the pretrained models are used to initialize training on downstream tasks and their performance is evaluated in a frozen (section 5.4.2), a half-frozen (section 5.4.3), and a full finetuning (section 5.4.4) setting.

Pretraining

The implementation is mainly based on the published code of Pri3D, and thus, the same pretraining protocol as presented in the paper [1] is follwed and an outline is given here. The 2D backbone is always a UNet-style ResNet50, and a UNet-style sparse convolutional serves as 3D backbone. As in [1], the 2D backbone is initialized using supervised ImageNet pretraining before performing self-supervised pretraining on the *ScanNet* dataset [11]. The *ScanNet* dataset is a collection of RGB-D sequences of indoor scenes, captured with a Kinect-like setup. On this dataset, the model is trained for 5 epochs using stochastic gradient descent with a learning rate of 0.1 with polynomial decay of 0.9 and the batch size is 64, unless explicitly mentioned otherwise. This corresponds to approximately 60k iterations and 4 days of training time on 8 Nvidia V100 GPUs for pretraining a single model. The temperature parameter in the losses (equation (5.1)) is set to 0.4.

Per-pixel downstream tasks

In sections 5.4.2 to 5.4.4 the per-pixel representations are evaluated on depth prediction, image reconstruction, and semantic segmentation while parts of the pretrained network are frozen. To decode the information from features extracted by the frozen pretrained network, a non-linear per-pixel MLP is appended (see figure 5.2c). This MLP is implemented as a sequence of 1x1 convolutions: Conv1x1@128 ⟶ ReLU ⟶ BatchNorm ⟶ Conv1x1@128 ⟶ ReLU ⟶ BatchNorm ⟶ Conv1x1@c ($c = 3$ for image reconstruction, $c = 1$ for depth estimation, and $c = \#classes$ for semantic segmentation). The MLP is followed by a bilinear fixed interpolation that upscales the output from 120×160 to 240×320, which is analogous to the semantic segmentation downstream experiments of Pri3D [1] with the UNet architecture. Note that the last convolutional layer is discarded from the pretrained model, which, during pretraining, linearly projects the features to the space in which the contrastive loss operates. It can be regarded as the head network which aims to solve the contrastive task.

For image reconstruction and depth prediction, the models are trained using an L2 loss directly on the RGB input pixel values and depth ground truth values, respectively. For semantic segmentation, the common cross-entropy loss is used to learn to predict the semantic classes of each pixel. Other than that, the same training protocol as for the semantic segmentation task in [1] is applied. The mean Intersection-over-Union (mIoU) is the metric for the semantic segmentation quality. The datasets used for depth prediction and image reconstruction are:

- *ScanNet25kframes* [11]: a subset of the *ScanNet* dataset annotated for specific tasks, e.g. semantic segmentation, with given training/test/validation splits. The RGB input images of all three splits are subsets of the *ScanNet* dataset used for pretraining. *ScanNet25kframes* is on all downstream tasks and therefore it is not distinguish from *ScanNet* explicitly.
- *NYUv2* [30]: A dataset of 1449 RGBD images with semantic segmentation annotations, capturing 464 indoor scenes.

For semantic segmentation, the follow additional datasets are used:

- *KITTI* [31]: The dataset of the KITTI semantic segmentation benchmark consists of 200 semantically annotated training as well as 200 test images (the validation set) of road traffic scenes.
- *Cityscapes* [32]: A dataset of 5000 semantically annotated frames of urban street scenes in 50 different cities.

Object-centered downstream tasks

To perform object detection and instance segmentation, the encoders of the pretrained models are used to initialize a Mask RCNN model [33] of the Detectron2 library [34] in section 5.4.5. Due to the downstream architecture, the decoders are discarded. In addition to *ScanNet25kframes* and *NYUv2*, the models are evaluated on the 2017 version of the *COCO* dataset [35] that contains more than 118k training and 5000 validation images. The Average Precision (AP) metric is used on all datasets.

5.4.2 Frozen downstream tasks

The frozen downstream tasks, where all weights of the pretrained network are kept frozen, are used to investigate how well the pretrained models capture color, depth, or semantic information by predicting this information from the learned per-pixel representations. These frozen downstream tasks show how the different alignment strategies in Pri3D influence the learned representations. For semantic segmentation, this frozen setup is similar to the linear evaluation protocol commonly used to evaluate the quality of global image features in visual representation learning [6]. In this linear evaluation protocol, the performance of a linear classifier on the global features is often regarded as a proxy metric for the representation quality. The presented setup is used to evaluate whether a non-linear MLP can provide a proxy metric for the representation quality.

The first four rows of table 5.1 show the best performance on the validation data during the training process for each model variation and frozen downstream task.

		Depth pred. - L2 loss		Image reconst. - L2 loss		Semantic Segm. mIoU			
		NYUv2	ScanNet	NYUv2	ScanNet	Cityscapes	KITTI	NYUv2	ScanNet
Frozen	VIEW	-120.93	-35.73	-79.87	-79.42	16.54	12.54	32.11	34.64
	GEO	-120.89	-26.75	-120.69	-106.73	11.36	6.98	21.89	21.53
	VIEW+GEO	-121.78	-28.88	-117.61	-103.95	12.54	7.29	25.59	28.19
	VIEW+GEO (lin)	-108.48	-25.73	-91.07	-89.58	14.33	9.01	28.96	31.95
Half-frozen	VIEW	-47.27	-10.62	-7.94	-2.63	41.79	25.69	47.03	57.34
	GEO	-41.42	-7.18	-7.21	-2.46	43.50	28.49	47.71	58.37
	VIEW+GEO	-43.34	-8.09	-8.01	-2.63	42.22	23.75	46.55	58.01
	VIEW+GEO (lin)	-43.00	-7.86	-7.85	-2.67	42.43	24.53	47.59	58.12
Full finetuning	VIEW	-46.68	-9.59	-8.19	-2.71	51.29	30.11	51.61	59.44
	GEO	-39.14	-6.82	-7.56	-2.54	50.97	29.75	52.08	59.86
	VIEW+GEO	-36.63	-7.40	-8.01	-2.68	50.51	29.26	50.34	60.72
	VIEW+GEO (lin)	-39.00	-7.24	-8.23	-2.73	49.91	29.78	51.19	60.55

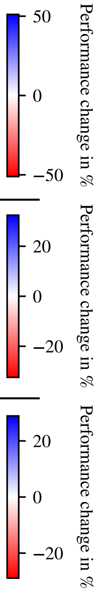


Table 5.1: Validation performance on the three tasks and four datasets. Rows 1-4 show the frozen downstream setting where the pretrained weights are fixed, in rows 5-8 the encoder of the UNet-style architecture is fixed, and in the last four rows, the entire models are finetuned. To have a consistent notion that a greater value means better performance, the negative L2-Loss is shown. Blue color indicates relative improvement with respect to the cross-view model (*VIEW*).

Compared to the cross-view model (*VIEW*), all other models perform worse on frozen image reconstruction and semantic segmentation. In contrast, the models that include the cross-modal *GEO* loss perform better than the *VIEW* on ScanNet depth prediction, while on NYUv2 this general trend is not visible. The *VIEW+GEO (lin)* model outperforms the *VIEW+GEO* model on all three frozen downstream tasks by more than 10%.

Figures 5.3 and 5.4 show the qualitative reconstruction and depth prediction results. In both cases, the ranking of the models on ScanNet is reflected in the visual reconstruction quality. The *VIEW* model achieves the best visual reconstruction, followed by the *VIEW+GEO (lin)*, *VIEW+GEO* and *GEO* models. It is notable that the results differ especially on flat texture-rich surfaces, for example in the first row where the *GEO* model does recover the wallpaper. All model variations discard most of the color information. The most detailed and smoothest depth estimation is obtained by the *VIEW+GEO (lin)* model. The depth gradient along surfaces appears smooth whereas the depth estimation based on *VIEW+GEO* and *VIEW* is speckled on flat surfaces.

In conclusion, the cross-view *VIEW* loss preserves more texture information while the cross-modal *GEO* loss leads to a better depth representation. This observation is in line with the ideas of complementary and redundant information discussed in figure 5.1. Interestingly, combining the two losses as *VIEW+GEO* and aligning the representations

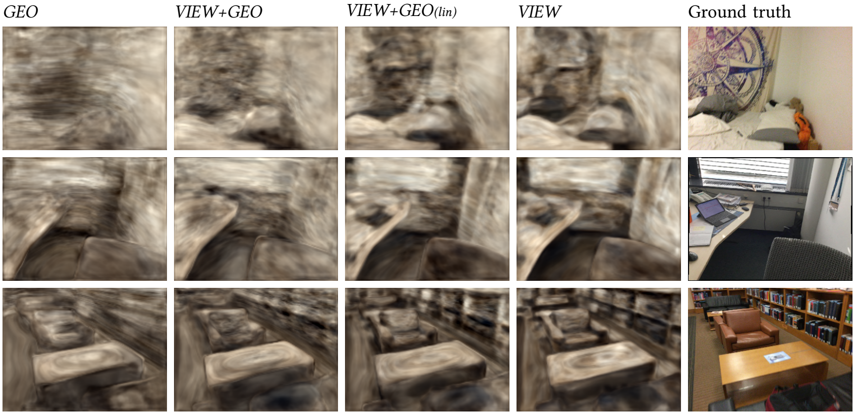


Figure 5.3: Image reconstructions obtained from different frozen representations. The rows correspond to different samples. The rightmost column is the ground truth data and also the input for the feature extractors. The other columns use different models to compute the features from which the images are reconstructed.

across views and modalities only achieves mediocre performance on all three downstream tasks. Once alignment is lifted by the linear layer in *VIEW+GEO (lin)*, all metrics improve, showing that it allows the model to preserve more complementary information. Notably, the depth prediction result is even better than for the *GEO* model, suggesting that even with respect to the redundant information, this separation is beneficial. The semantic segmentation performance differs strongly between the model variations and benefits from the cross-view loss and the linear layer in *VIEW+GEO (lin)* that lifts the direct alignment.

5.4.3 Half-frozen downstream tasks

By freezing only the encoder part of the UNet-style models, the following experiments provide insight to which part of architecture discards the texture information. The experimental setup, architecture, and datasets remain the same as for the frozen downstream tasks in section 5.4.2, except that now also the decoder is finetuned together with the MLP and only the encoder is fixed during finetuning.

The *GEO* model performs best across all tasks and datasets as seen in rows 5-8 of table 5.1. The model excels on depth prediction, and interestingly, also outperforms the *VIEW* model on image reconstruction. The combined models outperform *VIEW* on depth prediction, but the other tasks show mixed results. The linear layer *(lin)* improves the *VIEW+GEO* performance slightly.

Overall, the results are in stark contrast to the frozen downstream tasks. Given that the *GEO* model outperforms the *VIEW* model, it seems that the decoder discards the texture information in the frozen downstream tasks, and recovering the texture is possible by finetuning. At the same time, the encoder of the *GEO* model extracts very robust features across tasks and datasets.

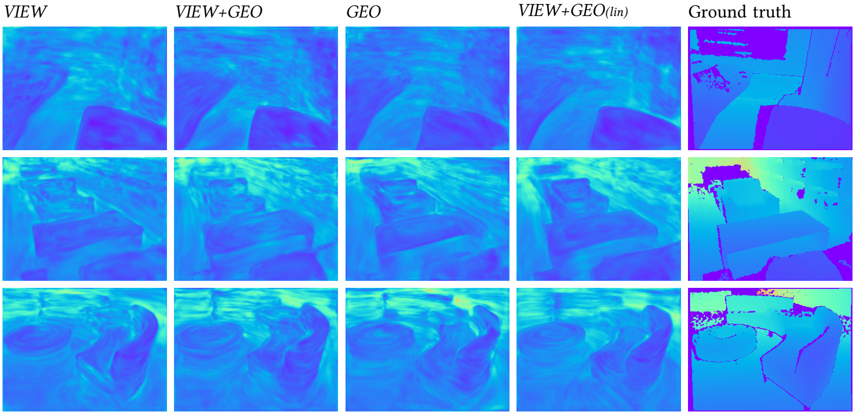


Figure 5.4: Qualitative results on the frozen depth prediction task. Each row corresponds to one sample and the rightmost column shows the ground truth depth data (saturated blue indicates invalid values). The other columns use different models to compute the frozen per-pixel features for depth prediction.

5.4.4 Full finetuning downstream tasks

To assess the practical value and generality of a visual representation, the models are now fully finetuned and evaluated on the downstream tasks. In contrast to the frozen downstream tasks, all weights in the model are finetuned during the downstream task training.[1] As the full model is finetuned, the per-pixel MLP is not needed anymore and is omitted.

Rows 9-12 in table 5.1 show the full finetuning downstream performance. Models that were trained including the *GEO* loss outperform the *VIEW* model on depth prediction. Again, *GEO* is the best model for image reconstruction. On semantic segmentation, *VIEW* performs best on *Cityscapes* and *KITTI*, but not on the other two datasets.

Also in this full finetuning setting, cross-modal alignment improves depth prediction performance. For semantic segmentation, the results are generally mixed, but it is notable that *VIEW* performs well on the two outdoor datasets. This may indicate that the cross-modal models are biased towards spatial layouts of indoor scenes. Adding the feature space separation *(lin)* to *VIEW+GEO* does not have a positive effect, but rather tends to decrease the performance.

5.4.5 Instance and object segmentation

As discussed in [12], texture-biased networks are inferior to shape-biased networks for object detection. To test whether this also holds for texture-biased compared to depth-biased networks, the pretrained encoders are finetuned for object detection and instance segmentation.

Table 5.2 shows that the cross-modal *GEO* model outperforms the *VIEW* model on 5 out of 6 task and dataset combinations. The combined models *VIEW+GEO* and *VIEW+GEO (lin)* models show mixed results, perform well on *ScanNet*, but not on *COCO*. On *NYUv2*,

[1] Note that the results deviate from the results presented in [1] since the proxy depth loss is discarded (see supplementary material).

	COCO		NYUv2		ScanNet	
	Instance Segm. AP	Object Det. AP	Instance Segm. AP	Object Det. AP	Instance Segm. AP	Object Det. AP
VIEW	36.13	39.99	15.11	18.98	19.00	26.68
GEO	36.48	40.31	15.41	19.24	19.15	26.51
VIEW+GEO	36.04	39.80	15.62	19.94	19.14	27.00
VIEW+GEO (lin)	35.92	39.63	14.78	18.46	19.22	26.82

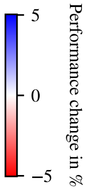


Table 5.2: Comparison of instance segmentation and object detection performance of the four different models (rows). Blue color indicates relative improvement with respect to the cross-view model (*VIEW*).

their difference is most pronounced where the linear feature separation *(lin)* leads to the worst performance among the four model variations while omitting it results in the best performance.

Combining the losses in the *VIEW+GEO* and *VIEW+GEO (lin)* models seems to be beneficial within the pretraining dataset, but not when shifting domains. Overall, although the performance differences are small, the *GEO* model appears to be more robust. This supports the idea that texture-biased networks are not as robust for object-centered tasks as the depth-biased *GEO* network.

5.5 Discussion

The comparison on the frozen downstream tasks reveals that cross-view and cross-modal representation alignment influence the complementary and redundant information encoded in the learned representations. While the *VIEW* model preserves the texture and the *GEO* model preserves the depth, the combination *VIEW+GEO* does not result in the different low-level features complementing each other, but instead a fully shared feature space is obtained where complementary information is lost and redundant details are less accentuated. *VIEW+GEO (lin)* improves the combined model on all three frozen downstream tasks and raises the question of whether this means that a generally better representation has been found. The performance on the half-frozen and full finetuning downstream tasks, however, contradict this apparent conclusion and it is even found that the *GEO* model performs most robustly across tasks and datasets. This shows that reducing texture-biased networks increases robustness is similar to findings in [12].

On semantic segmentation, the frozen downstream task shows a distinct ranking that is persistent across datasets. These distinct performance differences were not observed once the decoder is finetuned as well. A possible explanation could be that the texture-bias is to a certain degree useful but also easy to recover and thus not important to learn during pretraining. So although the strong differences on the frozen downstream tasks reveal that the diverse representations either incorporate or discard complementary information, the performance on the frozen tasks does not correlate with the finetuning downstream performance.

The comparison of *VIEW+GEO* with *VIEW* and *GEO* indicates different behavior on data within the same domain and across domains. The *VIEW+GEO* model excels especially in object detection, instance segmentation, and semantic segmentation on the *ScanNet* dataset. A similar effect of overfitting to the pretraining dataset is also reflected in the learning rate variations shown in the supplementary material. In a broader context, this could mean that aligning cross-modal and cross-view representations may harm generalization across datasets but could prove especially useful when exploiting unlabelled data within the same domain for better performance on a small annotated subset. Still, further experiments are needed to verify this hypothesis.

The presented empirical study comes with certain limitations. Although the results are obtained on various datasets and tasks, the pretraining and finetuning results are always subject to stochastic effects in the training processes. Further, this work focused only on the most obvious modality-specific information such as color and depth. The image reconstruction and depth prediction tasks only test some characteristics of the learned representations while other characteristics may be more indicative with respect to downstream finetuning performance. Still, the results show that the different pretraining strategies yield representations that encode different information.

5.6 Conclusions and future work

Aligning feature representations across views and across modalities is a common approach in self-supervised learning. Yet, the effects on the learned visual representations are often not well understood. The quantitative results showed that, in contrast to cross-view representation alignment, cross-modal representation alignment can lead to discarding complementary information of the individual modalities. In the context of cross-modal learning on images and point clouds, texture is considered complementary information of the visual data and depth is redundant information. Pretraining by cross-modal alignment was especially useful for tasks that require a notion of the spatial layout, such as depth prediction, and also instance segmentation and object detection. This effect is similar to the observations of previous work that shape-biased networks are more robust than texture-biased networks. Thus, pretraining with emphasis on the redundant information achieved the most robust performance when finetuning the visual backbone networks on various downstream tasks and datasets.

Merging complementary information is a fundamental goal of sensor fusion and discarding that information may harm the overall performance on such a task. While the presented results have indicated that visual representations become more robust by reducing the texture-bias through cross-modal alignment, future work should investigate whether these results still hold for self-supervised approaches that fuse different sensing modalities.

5.7 References

[1] Ji Hou, Saining Xie, Benjamin Graham, Angela Dai, and Matthias Nießner. Pri3d: Can 3d priors help 2d representation learning? In *Proc. of the IEEE/CVF International Conference on Computer Vision*, pages 5693–5702, 2021.

[2] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[3] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking ImageNet pre-training. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019.

[4] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[5] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[7] Xin Yuan, Zhe Lin, Jason Kuen, Jianming Zhang, Yilin Wang, Michael Maire, Ajinkya Kale, and Baldo Faieta. Multimodal Contrastive Training for Visual Representation Learning. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6995–7004, 2021.

[8] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *European Conference on Computer Vision*, pages 776–794. Springer, 2020.

[9] Daniel Trosten, Sigurd Lokse, Robert Jenssen, and Michael Kampffmeyer. Reconsidering Representation Alignment for Multi-view Clustering. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1255–1265, 2021.

[10] Maximilian Jaritz, Tuan-Hung Vu, Raoul de Charette, Emilie Wirbel, and Patrick Pérez. xMUDA: Cross-modal unsupervised domain adaptation for 3d semantic segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12605–12614, 2020.

[11] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5828–5839, 2017.

[12] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, May 2019.

[13] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[14] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Remi Munos, and Michal Valko. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems*, 33:21271–21284, 2020.

[15] Xinlong Wang, Rufeng Zhang, Chunhua Shen, Tao Kong, and Lei Li. Dense Contrastive Learning for Self-Supervised Visual Pre-Training. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[16] Zhenda Xie, Yutong Lin, Zheng Zhang, Yue Cao, Stephen Lin, and Han Hu. Propagate Yourself: Exploring Pixel-Level Consistency for Unsupervised Visual Representation Learning. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 16684–16693, June 2021.

[17] Ji Hou, Benjamin Graham, Matthias Nießner, and Saining Xie. Exploring data-efficient 3d scene understanding with contrastive scene contexts. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 15587–15597, 2021.

[18] Saining Xie, Jiatao Gu, Demi Guo, Charles R Qi, Leonidas Guibas, and Or Litany. PointContrast: Unsupervised pre-training for 3d point cloud understanding. In *European Conference on Computer Vision*, pages 574–591. Springer, 2020.

[19] Zaiwei Zhang, Rohit Girdhar, Armand Joulin, and Ishan Misra. Self-supervised pretraining of 3d features on any point-cloud. *arXiv preprint arXiv:2101.02691*, 2021.

[20] Yunze Liu, Qingnan Fan, Shanghang Zhang, Hao Dong, Thomas Funkhouser, and Li Yi. Contrastive multimodal fusion with TupleInfoNCE. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, pages 754–763, 2021.

[21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[22] Zhenyu Li, Zehui Chen, Ang Li, Liangji Fang, Qinhong Jiang, Xianming Liu, Junjun Jiang, Bolei Zhou, and Hang Zhao. SimIPU: Simple 2D Image and 3D Point Cloud Unsupervised Pre-Training for Spatial-Aware Visual Representations. *arXiv preprint arXiv:2112.04680*, 2022.

[23] Longlong Jing, Ling Zhang, and Yingli Tian. Self-supervised feature learning by cross-modality and cross-view correspondences. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 1581–1891. IEEE, 2021.

[24] Nawid Sayed, Biagio Brattoli, and Björn Ommer. Cross and learn: Cross-modal self-supervision. In *German Conference on Pattern Recognition*, pages 228–243. Springer, 2018.

[25] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2827–2836, 2016.

[26] Adrian Benton, Huda Khayrallah, Biman Gujral, Dee Ann Reisinger, Sheng Zhang, and Raman Arora. Deep generalized canonical correlation analysis. *arXiv preprint arXiv:1702.02519*, 2017.

[27] Michelle A. Lee, Matthew Tan, Yuke Zhu, and Jeannette Bohg. Detect, Reject, Correct: Crossmodal Compensation of Corrupted Sensors. In *Proc. of the IEEE International Conference on Robotics and Automation*, Xi'an, China, 2021.

[28] Yunze Liu, Li Yi, Shanghang Zhang, Qingnan Fan, Thomas Funkhouser, and Hao Dong. P4Contrast: Contrastive Learning with Pairs of Point-Pixel Pairs for RGB-D Scene Understanding. *arXiv preprint arXiv:2012.13089*, 2020.

[29] Ji Hou. Pri3d. `https://github.com/Sekunde/Pri3D` (commit 9086edd), 2021.

[30] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012.

[31] Hassan Alhaija, Siva Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes. *International Journal of Computer Vision*, 2018.

[32] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[33] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, pages 2961–2969, 2017.

[34] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[35] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[36] Junjie Hu, Mete Ozay, Yan Zhang, and Takayuki Okatani. Revisiting single image depth estimation: Toward higher resolution maps with accurate object boundaries. In *IEEE Winter Conference on Applications of Computer Vision*, pages 1043–1051, 2019.

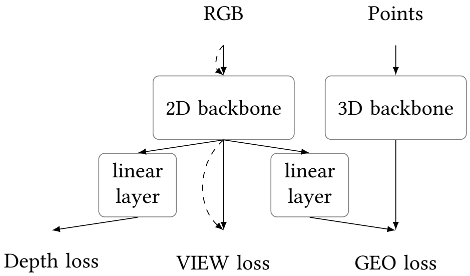


Figure 5.5: This graph shows how the depth proxy loss is included in the Pri3D model. The dashed arrow indicates a second RGB view of the same scene that overlaps with the first view. Note that *VIEW+GEO+d (lin)* was evaluated throughout the Pri3D paper.

5.8 Appendix

5.8.1 Relation to results presented in Pri3D

All models in Pri3D [1] were pretrained with a depth loss in addition to the *VIEW* and *GEO* loss. This *proxy* depth loss follows the formulation presented in [36] and details are found in the official Pri3D implementation [29]. Note that this loss is also preceded by a linear per pixel layer that reduces the backbone output channels to a single channel (see figure 5.5). In this work, this proxy loss is not included, because the goal was to focus on the effects of cross-view and cross-modal representation alignment. table 5.3 show the results of the Pri3D variations once the proxy depth loss is used. The relative performance change is given with respect to the corresponding model without the depth loss, which is not listed explicitly. While the depth loss does have positive effects on semantic segmentation of *ScanNet* and *NYUv2*, it is not beneficial on the object detection and instance segmentation tasks on the same datasets. On the other datasets and tasks it leads to mixed results.

Further, some models in [1] were pretrained with a learning rate of 0.1 and others with 0.01. table 5.4 shows the effect of the learning rate change. Within the *ScanNet* dataset, the higer learning rate outperforms the lower learning rate, and vice versa on the other datasets. The pretraining learning rate thus seems to be a crucial factor that should be taken into account when evaluating models within or across datasets.

	COCO Object Det. AP	COCO Instance Segm. AP	CityScapes Semantic Segm. mIoU	KITTI Semantic Segm. mIoU	NYUv2 Object Det. AP	NYUv2 Instance Segm. AP	NYUv2 Semantic Segm. mIoU	ScanNet Object Det. AP	ScanNet Instance Segm. AP	ScanNet Semantic Segm. mIoU
GEO+d (lin)	40.10 (-1.0%)	36.28 (-0.7%)	50.24 (-2.8%)	30.85 (-1.8%)	17.57 (-11.8%)	13.84 (-12.1%)	52.46 (0.9%)	26.15 (-1.8%)	18.72 (-1.2%)	60.61 (1.0%)
VIEW+d	40.12 (0.3%)	36.23 (0.3%)	51.75 (0.9%)	30.28 (0.5%)	18.03 (-5.0%)	14.33 (-5.2%)	53.00 (2.7%)	26.34 (-1.2%)	19.05 (0.2%)	61.08 (2.8%)
VIEW+GEO+d	39.58 (-0.5%)	35.82 (-0.6%)	50.38 (-0.2%)	28.80 (-1.6%)	18.89 (-5.2%)	15.17 (-2.9%)	51.40 (2.1%)	26.41 (-2.2%)	19.20 (0.4%)	61.32 (1.0%)
VIEW+GEO+d (lin)	39.79 (0.4%)	36.02 (0.3%)	49.39 (-1.0%)	30.13 (1.2%)	18.42 (-0.2%)	14.34 (-3.0%)	51.43 (0.5%)	26.44 (-1.4%)	19.08 (-0.7%)	61.42 (1.4%)

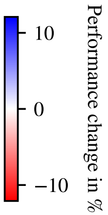


Table 5.3: Comparison of pretraining with and without the depth loss. The baseline for the performance change of per row is the same model but without the depth loss. The baseline performance is not listed explicitly here.

	COCO		Cityscapes	KITTI	NYUv2			ScanNet		
	Instance Segm. AP	Object Det. AP	Semantic Segm. mIoU	Semantic Segm. mIoU	Instance Segm. AP	Object Det. AP	Semantic Segm. mIoU	Instance Segm. AP	Object Det. AP	Semantic Segm. mIoU
VIEW+GEO+d (lin)	36.02	39.79	49.39	30.13	14.34	18.42	51.43	19.08	26.44	61.42
VIEW+GEO+d (lin, lr001)	36.72	40.69	53.72	30.54	15.21	19.13	54.90	18.67	25.98	59.70

Performance change in %
5
0
−5

Table 5.4: *VIEW+GEO+d (lin)* with learning rate 0.1 compared to learning rate 0.01. The relative performance is shown with respect to the baseline model *VIEW+GEO+d (lin)*.

6 Conclusions and future work

Self-driving vehicles and advanced driver assistance systems are becoming ubiquitous. Therefore, it is crucial to make self-driving vehicles as safe as possible and aim for even higher reliability than human drivers can provide. The research presented in this thesis contributes toward this goal by increasing the efficiency of an essential aspect of autonomous driving: the vehicle's perception of its environment. To make the environment perception more efficient, the research presented here improved *sensor*, *algorithm*, and *representation efficiency*.

6.1 Sensor efficiency

By capturing more cues of the environment, which may even be unrecognized by human drivers, novel sensors can increase the efficiency of environment perception. Vehicles approaching behind blind corners can be acoustically detected from their wall reflections by a car-mounted microphone array, as demonstrated in chapter 2. The method achieved an accuracy of 0.92 on the 4-class hidden car classification task for a static ego-vehicle, and up to 0.84 in some environments while driving. An approaching vehicle was detected with the same accuracy as the visual baseline already more than one second ahead, a crucial advantage in such critical situations.

While these initial findings are encouraging, the presented results have several limitations. The experiments included only a few locations and different oncoming vehicles, and while the method performed well in one environment, it had difficulties in the other and did not perform reliably in unseen test environments. To expand the applicability, more representative data is needed to capture a broad variety of environments, vehicle positions, and velocities, and the presence of multiple sound sources. Rather than generalizing across environments, additional input from map data or other sensor measurements could help to discriminate acoustic environments and to classify the reflection patterns accordingly. More data also enables end-to-end learning of low-level features, potentially capturing cues the Direction-of-Arrival-based approach currently ignores (e.g. Doppler, sound volume), and performing multi-source detection and classification in one pass [1]. Ideally, a suitable self-supervised learning scheme is developed similar to [2], though a key challenge is that actual occluded sources cannot immediately be visually detected.

This work on acoustic vehicle detection around corners shows how novel sensors can help to extend the perception range in situations where conventional sensors are lacking. Of course, microphones are not the only sensors that have the potential to achieve this. Other efforts, for example on thermal infrared cameras [3], support that it is worth exploring novel sensing modalities. The microphone array setup, despite being inspired by human experience in urban traffic, has capabilities beyond the human audition. The spatial layout of the microphone array offers a higher resolution than a binaural setup and microphones can capture frequencies beyond the range of human ears. What is often missing are algorithms to process the data and finding the relevant use cases where novel sensors can bring significant safety benefits for self-driving vehicles. While industrial product development focuses on tuning well-known sensors (e.g. camera, LiDAR, and radar), it is a great opportunity for academic research to think “outside-the-box” and explore the possibilities and applications of a novel or so far overlooked sensing techniques.

6.2 Algorithm efficiency

Processing of sensor data requires efficient algorithms to minimize the latency between capturing, understanding, and eventually reacting to the environment. Chapters 3 and 4 described how existing approaches can be extended for a favorable trade-off of accuracy and processing speed.

Chapter 3 presented a new approach to train deterministic decision trees with gradient-based optimization in an end-to-end manner. The approach uses a probabilistic tree formulation during training to facilitate back-propagation and optimize all splits of a tree jointly. By adjusting the steepness of the decision boundaries in an annealing scheme, the method learns increasingly more crisp trees that capture uncertainty as distributions at the leaf nodes, rather than as distributions over multiple paths. The resulting optimized trees are therefore deterministic rather than probabilistic, and run efficiently at test time as only a single path through the tree is evaluated. This approach outperforms previous training algorithms for oblique decision trees. In a forest ensemble, the method shows competitive or superior results to the state-of-the-art *sNDF* [4], even though the trees only evaluate a fraction of the split functions at test time. Unlike the *ADF* method [5], the presented method is not restricted to only using oblique split functions, thanks to gradient-based optimization. It is straightforward to include more complex split features, such as convolutional neural networks, or to add spatial regularization constraints. Another demonstrated benefit is that the learned decision tree can also help interpret how the decision of a visual classification task is constructed from a sequence of simpler tests on visual features. Overall, the presented approach provides high flexibility and the potential for accurate models that maintain interpretability and efficiency due to the conditional data flow. In perception for autonomous vehicles, this efficiency through conditional data flow could lead to specialized models that perform efficient scene parsing, for example by early exits, which avoid further computations once a model is certain of its prediction.

Chapter 4 introduced Instance Stixels to improve stixel segmentation by considering instance information from a CNN, and performing a subsequent stixel clustering step. The experiments showed multiple benefits of including the instance information already in the segmentation step, as opposed to clustering Semantic Stixels. First, quantitative and qualitative analysis shows that Instance Stixels adhere better to object boundaries. Second, Instance Stixels provide more accurate instance segmentation than Semantic Stixels augmented with instance information from a pixel-level instance segmentation network. Third, Instance stixels still preserve the favorable stixel characteristics in terms of compactness of the segmentation representation (on average less than 2673 stixels per image) and computational efficiency (up to 28 FPS at a resolution of 1792x784). Potential future research directions are the integration of additional sensor modalities as shown in [6] and temporal information to enforce consistency.

In the meantime, improvements in neural network architecture have led to pure deep learning approaches that outperform Instance Stixels in terms of processing throughput and potentially accuracy [7]. Furthermore, algorithmic improvements to achieve faster processing speed often compete with hardware improvements that essentially speed up implementations without any changes to the algorithms. Consequently, the benefit of the algorithmic improvements is often diminished. Yet, there are robotic applications that require specific hardware that is cheaper and physically smaller than GPU-accelerated desktop PCs. These cost and space benefits usually come at the expense of computational power. In such cases, conditional computation models such as decision trees can offer low computational footprints by skipping irrelevant computations. These traits make conditional models attractive for applications that require the models to run on less powerful hardware, and hence conditional models will remain an ongoing topic of research.

6.3 Representation efficiency

Instance Stixels, presented in chapter 4, offer a more compact and abstract representation than the per-pixel data of a stereo camera. Instead of more than a million pixels in an RGB-D image with each containing segmentation results, stixels compress this information to only a few thousand stixels per image. As such, stixels can provide a more compact interface to communicate the segmented scene to other algorithms than per-pixel data. Yet, integration with a path planning algorithm for obstacle avoidance in a research vehicle showed that the representation obtained from stixels is still often too fine-grained for planning and, due to its susceptibility to noise, needs additional filtering. This shows that the value of such a compressed representation is often determined by the subsequent modules and their requirements. A 2D path planning algorithm, for example, rarely benefits from height information, but often simply requires linear boundary constraints. It is therefore imperative to think about representations from an application perspective: what is the actual task at hand that needs to be solved, and does the representation fulfill the requirements of the subsequent processing steps? The original stixels have done this superbly, as a simple way to detect cars on highways ahead of the ego-vehicle using only disparity data. They enabled vehicles to prevent collisions with traffic ahead, and have done so in times when annotated data of traffic scenes were limited and end-to-end learning had not yet become the dominant approach to computer vision. In urban traffic, however, the scenarios are more complex than on highways [8], and meanwhile annotated data, as well as more powerful hardware, have enabled new, more accurate, and more versatile approaches. In conclusion, for intermediate representations such as stixels, it is important to thoroughly evaluate their applicability and value in subsequent processing steps.

The fusion of sensor data for a shared objective is a research problem that could benefit from intermediate representations. Advances in self-supervised learning have enabled learning expressive representations from unlabeled multi-modal data. Such data can be continuously collected by fleets of sensor-equipped vehicles which offers a large potential for self-supervised learning. In self-supervised learning, aligning feature representations across views and across modalities is a common approach. Yet, the effects on the learned representations are often not well understood. The quantitative results in chapter 5 showed that, in contrast to cross-view representation alignment, cross-modal representation alignment can lead to discarding complementary information of the individual modalities. In the context of cross-modal learning on images and point clouds, the texture is considered complementary information of the visual data and depth is redundant information. Pretraining by cross-modal alignment was especially useful for tasks that require a notion of the spatial layout, such as depth prediction, and also instance segmentation and object detection. This effect is similar to the observations of previous work that shape-biased networks are more robust than texture-biased networks. Thus, pretraining with emphasis on the redundant information achieved the most robust performance when finetuning the visual backbone networks on various downstream tasks and datasets.

Merging complementary information is a fundamental goal of sensor fusion and discarding that information may harm the overall performance on such a task. While the presented results have indicated that visual representations become more robust by reducing the texture-bias through cross-modal alignment, future work should investigate whether these results still hold for self-supervised approaches for sensor fusion.

6.4 Outlook

This thesis presented approaches to improve the efficiency of environment perception in self-driving vehicles. Improving the efficiency of sensors, algorithms, and representations all equally contribute to the overarching goal of faster perception that increases the time in which a self-driving vehicle can react to dangerous situations.

Newer hardware that enables faster processing can, in some cases, render efficiency improvements of algorithms irrelevant. Therefore, computational speed improvements by resorting to approximate solutions that harm the accuracy, such as stixels, may be useful for a short time but will likely be outperformed in the long run. Likewise, vehicle-to-everything (V2X) communication may also reduce the need for novel sensors. Communication with other cars or infrastructure, for example, may alarm the ego-vehicle of oncoming visually occluded traffic. This, however, makes the ego-vehicle dependent on other devices, while novel sensors contribute to the autonomy of self-driving vehicles. Research on novel sensing modalities may benefit from advances in efficient representations of data. Once correspondences of data of different sensors are well established, their data may be exploited to aid learning approaches on novel sensor data. For these reasons, multi-modal self-supervised learning is currently gaining momentum and acoustic sensing for visually occluded objects should be explored further in the future.

To improve the overall performance of self-driving vehicles, regardless whether efficiency, accuracy, or almost any other aspect, it is necessary to pursue a holistic evaluation approach for the full system, from perception all the way to control. This will be essential to exploit the full potential of self-driving vehicles in the pursuit of increased road safety.

6.5 References

[1] Weipeng He, Petr Motlicek, and Jean-Marc Odobez. Deep neural networks for multiple speaker detection and localization. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 74–79. IEEE, 2018.

[2] Chuang Gan, Hang Zhao, Peihao Chen, David Cox, and Antonio Torralba. Self-supervised moving vehicle tracking with stereo sound. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2019.

[3] Meng Ding, Wen-Hua Chen, and Yunfeng Cao. Thermal infrared single-pedestrian tracking for advanced driver assistance system. *IEEE Transactions on Intelligent Vehicles*, 2022.

[4] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulò. Deep neural decision forests. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2015.

[5] S. Schulter, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. Alternating decision forests. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 508–515, June 2013.

[6] Florian Piewak, Peter Pinggera, Markus Enzweiler, David Pfeiffer, and Marius Zöllner. Improved semantic stixels via multimodal sensor fusion. In *German Conference on Pattern Recognition*, pages 447–458. Springer, 2018.

[7] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proc. of the IEEE/CVF International Conference on Computer Vision*, pages 9157–9166, 2019.

[8] Andras Palffy, Ewoud Pool, Srimannarayana Baratam, Julian F. P. Kooij, and Dariu M. Gavrila. Multi-class road user detection with 3+1d radar in the view-of-delft dataset. *IEEE Robotics and Automation Letters*, 7(2):4961–4968, 2022.

Acknowledgments

First and foremost, I would like to express my gratitude to my promotor and advisor Prof. dr. Dariu M. Gavrila. Without your support and critical feedback, this thesis would not have become what it is today.

Further, I would like to thank my daily supervisor Dr. Julian F.P. Kooij. Your feedback on paper drafts and ideas was invaluable and I admire your scientific curiosity that has led to hours of discussions when I showed you results at the end of a day.

I further want to extend my sincere thanks to the committee members, Assist. Prof. dr. Valada, Prof. dr. Gevers, Prof. dr. ir. Wisse, Prof. dr. Babuska, and Dr. Caesar for their feedback and for joining me on the final part of this journey.

It was a great pleasure to work with all the members of the Intelligent Vehicles group at the TU Delft that have made it easy for me to integrate from day one. I would like to thank András Pálffy, Ewoud Pool, Joris Domhof, Zimin Xia, Hidde Boekema, Mubariz Zaffar, Jetze Schuurmans, Christoph Rist, Markus Roth, and Sebastian Krebs for technical discussions on papers, research ideas, devising lecture assignments and all the great fun! I had the great pleasure to supervise the M.Sc. theses of Patrick van Laar, Sietse van Schouwenburg, Yannick Schulz, and Avinash Mattar. Thank you for your trust, the fun, and all the lessons I learned from you. Frank Everdij, Ronald Ensing, and Oscar de Groot have been great colleagues and were invaluable for the great combined effort on our live obstacle evasion demonstration in 2020. Of course, thank you Tugrul Irmak, Yanggu Zheng, and Jork Stapel for being amazing office mates with great off-topic discussions, as well as Hanneke Hustinx, Karin van Tongeren, Dr. Barys Shyrokau, and Prof. dr. Riender Happee for their support and nice chats that always made me feel welcome. Thank you, Andras, Alberto, Zimin, Vishrut, and Jetze, for proofreading this thesis and for your helpful feedback and suggestions.

I greatly appreciate the support of Prof. Dr. Fred Hamprecht and Dr. Gijs Dubbelman for me personally and in our collaborations, for example on the i-CAVE project.

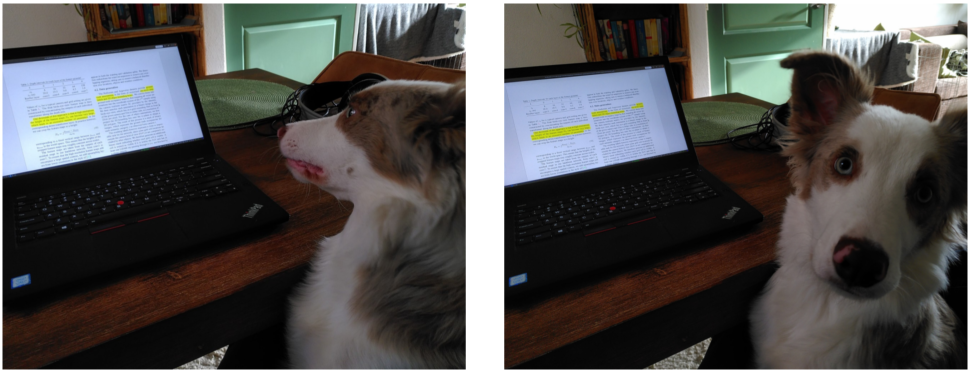
I am extremely grateful to my family, most importantly, my parents Katja and Roland, and my siblings Alexandra and Andreas, for their unconditional trust, support, and advice throughout my entire life which have always kept me grounded and made me feel safe.

Special thanks to Sabrina for her advice, encouragement, empathy, and patience during all the times when I struggled, as well as the fun and joy during the good times. Of course, I would also like to thank Dr. "Bobby" Bob for all the helpful pair-coding sessions that made the COVID-19 lockdowns more pleasant.

Lots and lots of thanks to all my friends for all the great times we had, that were there for me even when they haven't heard from me in a while and helped me refresh my mind whenever necessary.

Finally, I wish to extend my thanks to everyone else who supported me during my time as a Ph.D. student and I would like to thank you, the reader, for reading my thesis.

Delft, October 2022



(a) “Let me see...”

(b) “Seriously?!”

Figure 6.1: Dr. “Bobby” Bob clearly disagrees with my text highlighting.

Curriculum Vitæ

Thomas Markus Hehn

02/2018 – 01/2022 Ph.D. candidate, Intelligent Vehicles section, Cognitive Robotics Department, 3mE faculty, Delft University of Technology, The Netherlands.

08/2017 – 12/2017 Research assistant, Image Analysis and Learning, Heidelberg Collaboratory for Image Processing (HCI), Heidelberg University, Germany.

2014/10 – 2017/07 Graduate studies in physics (Master of Science), Heidelberg University, Germany.

2011/10 – 2014/10 Undergraduate studies in physics (Bachelor of Science), Heidelberg University, Germany.

2002/09 – 2011/06 Abitur, Gymnasium in der Glemsaue, Ditzingen, Germany.

1991/07/09 Date of birth in Stuttgart, Germany.

List of Publications

4. Yannick Schulz, Avinash Kini Mattar, Thomas M. Hehn, and Julian F.P. Kooij. “Hearing what you cannot see: Acoustic vehicle detection around corners”, IEEE Robotics and Automation Letters, 6, 2, p. 2587-2594, 2021, Institute of Electrical and Electronic Engineers.
   Author contributions: Equal share of authorship between first three authors. Yannick Schulz and Avinash Kini Mattar contributed to the data collection, prepared the data, implemented the acoustic detection method, performed the majority of the experiments, and contributed to writing. Thomas M. Hehn coordinated and contributed to the data collection, advised on the data preparation, implemented the visual baseline, took the lead in writing and presentation, and provided guidance and supervision. Julian F.P. Kooij devised the approach, contributed to writing, provided guidance and supervision.

3. Thomas M. Hehn, Julian F.P. Kooij, and Dariu M. Gavrila. “Fast and Compact Image Segmentation using Instance Stixels”, IEEE Transactions on Intelligent Vehicles, 7, 1, p. 45-56, 2022, Institute of Electrical and Electronic Engineers.
   Author contributions: Thomas M. Hehn devised the approach, implemented the GPU-optimized algorithm, conducted the experiments, and took the lead in writing. Julian F.P. Kooij advised on optimizing the algorithm, and provided guidance and supervision. Dariu M. Gavrila proposed the direction, provided guidance and supervision.

2. Thomas M. Hehn, Julian F.P. Kooij, and Fred A. Hamprecht. “End-to-end learning of decision trees and forests”, International Journal of Computer Vision, 128, 4, p. 997-1011, 2020, Springer US.
   Author contributions: Thomas M. Hehn devised the approach, implemented the GPU-optimized algorithm, performed the experiments, and took the lead in writing. Julian F.P. Kooij contributed to writing and provided guidance and supervision. Fred A. Hamprecht proposed the direction and provided guidance and supervision.

1. Thomas M. Hehn, Julian F.P. Kooij, and Dariu M. Gavrila. “Instance stixels: Segmenting and grouping stixels into objects”, IEEE Intelligent Vehicles Symposium (IV), p. 2542-2549, 2019, Institute of Electrical and Electronic Engineers.
   Author contributions: Thomas M. Hehn implemented the GPU algorithm, performed the experiments, and took the lead in writing and presentation. Julian F.P. Kooij contributed to writing, provided guidance and supervision. Dariu M. Gavrila provided guidance and supervision.