

Machine Learning for Grid-Based Sensor Fusion in Automotive Applications

Gabor Balazs

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Gerhard Rigoll

Prüfer der Dissertation:

1. apl. Prof. Dr.-Ing. Walter Stechele
2. Prof. Dr.-Ing. habil. Erwin Biebl

Die Dissertation wurde am 21.07.2021 bei der Technischen Universität München
eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am
13.12.2021 angenommen.

Abstract

Advanced driver assistance systems (ADAS) and automated cars fundamentally require a reliable perception of the vehicle’s environment. The robustness can only be achieved by sensor fusion, as the weaknesses of each sensor modality can be compensated by the other modalities. For ADAS of L2/2+, a full 360-degree perception combined with an estimate of the drivable space is required.

The current system setup, however, is extensively distributed, with the major pre-processing steps being performed within the sensor. Thus, only pre-processed, feature-level data is available for a central fusion control unit. Object- and target-based sensor data are merely sufficient to create such an environmental model with standard fusion algorithms.

The problem of fusing feature-level sensor data into a sensor-agnostic environmental model, where free-space estimation is available, raises the questions, whether the fusion can be performed by applying neural networks, and how this novel approach compares to classical fusion algorithms in terms of quality and costs.

In this work, the family of **Grid Fusion Networks** (**GridFuN**) is proposed. These models are based on the architecture of convolutional auto-encoders and are trained with a novel training scheme that facilitates data from distributed smart sensors.

To make the network models applicable on real-world automotive-grade microcontrollers, they are optimized for multiple objectives with a neural architecture search (NAS), in particular fusion quality and computational costs.

Results show that the discovered neural network models outperform the classical Bayesian grid fusion approach in fusion quality. Thus, **GridFuN** pose an alternative sensor fusion approach to standard Bayesian grid or Dempster-Shafer fusion, as they are capable of recognizing and interpreting the car’s surroundings. Further, an instantaneous situational awareness is possible—no map integration over time is needed to assess the situation in contrast to standard free-space mapping. In the course of the NAS, encoding and decoding blocks were created that excel similar blocks from literature (*AmoebaNet*, *Auto-DeepLab*, *Auto-DispNet*) in the grid fusion task.

The approach proposed in this work offers additional advantages for the mass-market by putting forward a cost-effective alternative to expensive and hard-to-integrate lidar sensors. Results of equivalent quality to lidar can be received from cheap camera and radar sensors.

Acknowledgments

At this point, I would like to express my gratitude to all the people who helped me on my journey throughout my dissertation. Many people have backed me up on small details or on big choices, with little hints or large motivational support.

First and foremost, my special thanks go to Prof. Walter Stechele for his respectful supervision and support. His helpful advice and concession to carry out my study freely without restrictions made the success of this work possible. Also, using his network and connecting me to the right people eased my daily work.

Besides that I would like to thank my technical supervisor at Infineon Technologies AG, Andre Roger, for pointing to the research topic from early on and helping me guide my study into the successful direction with the right boundaries.

Further, I would like to thank the automotive system engineering department at Infineon Technologies AG and my team leader Goran Keser for the proactive support in terms of infrastructure, legal support, and back-up against off-topic workloads that might have extended my dissertation timeline. I also have to thank Nicolas Leteinturier for supporting my works in dataset generation and neural network evaluations and Patrick Leteinturier for the insights into high level automotive system engineering.

My honest and deep gratitude also goes to the PhD network at Infineon Technologies AG for enabling a joyful time outside the dissertation. In the three years, new friendships evolved and numerous professional, social, and sport events lead to unforgettable memories, such as the *Campeon Innovation Weeks* or the *Automotive PhD Cup 2019!* The support from peers in a similar situation helped me in countless situations and resulted in a strong network that will last long after.

Lastly, but most importantly, I want to thank my parents for the unconditional and encouraging support, especially during the toughest times of my journey. My mother sensitized me to the demands on scientific writing leading to my two awarded papers. My father showed relentless interest in the technical backgrounds, always challenging me to investigate further and stick to project. Without the motivational and emotional support of my family, it would have been impossible to succeed.

Thank you!

Contents

1	Introduction	1
1.1	Problem Statement and Motivation	1
1.1.1	Levels of Automated Driving	1
1.1.2	Centralized vs. Distributed Processing	5
1.1.3	Applications of Machine Learning in Automotive	6
1.2	Structure of the Work	8
2	Preliminary Signal Processing	11
2.1	Raw Sensor Data Processing	12
2.1.1	Radar Signal Processing	12
2.1.2	Track-level Data Processing	19
2.1.3	Camera Sensor Processing	23
2.1.4	Other Sensor Preparations	25
2.2	Environmental Modeling Schemes	26
2.2.1	Object-Based Environmental Models	26
2.2.2	Grid-Based Environmental Models	28
2.2.3	Sensor Modeling	31
2.3	Sensor Fusion for Automotive Environmental Modeling	33
2.3.1	Levels of Sensor Fusion	33
2.3.2	Grid-Based Bayesian Fusion	34
2.3.3	Data Synchronization	37
3	Neural Networks for Sensor Fusion	41
3.1	Introduction and Motivation for the Usage of Convolutional Neural Networks and Auto-Encoders	41
3.2	State of the Art Neural Networks for Occupancy Grid Processing and Sensor Fusion	42
3.3	Neural Networks for Fusion of Camera and Radar Data	43
3.3.1	Structure, Building Blocks and Working Principle of Sensor Fusion Networks	44
3.3.2	Necessary Data and Available Datasets	53
3.3.3	Data Preparation	58
3.3.4	Training Configuration and Training Process	63
3.4	Fusion Quality and Hardware Requirements Analysis	65
3.4.1	Experimental Evaluation	65

Contents

3.4.2	MAC Counts and Memory Requirements	68
3.4.3	Optimization and Compression Techniques	74
3.5	Conclusion	77
4	Multi-objective Model Optimization with Neural Architecture Search	79
4.1	State-of-the-Art Neural Architecture Search Algorithms and Embedded CNN Accelerators	81
4.1.1	NAS Algorithms in Literature	81
4.1.2	State-of-the-Art Hardware Accelerators	82
4.2	Application of NAS Method to Sensor Fusion Networks	84
4.2.1	Macro-architectural Presets	85
4.2.2	Micro-architectural Search Space	86
4.2.3	NAS Method Details	88
4.3	Analysis of Automatically Designed Grid Fusion Networks	96
4.3.1	Search Progress and Evolution Overview	96
4.3.2	Results of the Search	99
4.3.3	Discussion of Results	103
4.4	Conclusion	106
5	Additional Application for Sensor Fusion Neural Networks	109
5.1	Use-case: Gesture Recognition for Multimedia Controls	109
5.1.1	Spatio-temporal Classification in Literature	110
5.2	Dataset Description	110
5.2.1	Data Gathering	110
5.2.2	Data Augmentation	113
5.3	Neural Network Architectures	114
5.3.1	Transformations for Dimensionality Reduction and Network Architectures	115
5.3.2	Network Scaling Factor	118
5.3.3	Embedded Deployment	118
5.4	Experimental Evaluation	118
5.4.1	Training Results and Embedded Implementation	118
5.4.2	Comparison to Literature	122
5.5	Conclusion	123
6	Conclusion	125
6.1	Summary	125
6.2	Publications	127
6.3	Outlook	127

List of Tables

1.1	Estimated number of sensors per modality per SAE automation level.	3
2.1	Comparison of track fusion algorithms	23
3.1	Overview of automotive perception datasets.	55
3.2	Parameter list of a typical radar target list.	56
3.3	Parameter list of a typical camera object list.	57
3.4	Parameter list of a typical Lidar point cloud.	57
3.5	Data format of the nuScenes dataset.	58
3.6	Confidence levels of finding the correct plane with RANSAC.	62
3.7	Occupancy grid maps obtained from the nuScenes dataset.	63
3.8	Results of Experimental Evaluation of Fusion Networks	67
3.9	Inference times of AE-FN and AEB-FN on Synopsys EV61	73
3.10	Inference times of AESQ-FN on Synopsys EV61	76
4.1	Model requirements for low-power neural accelerators	84
4.2	OPs of the NAS search space.	87
4.3	Comparative results of NAS experiments	104
5.1	Dataset of hand-gestures recorded with radar and ToF sensors.	111
5.2	GCN performances on GPU.	120
5.3	Performances of the proposed GCN architectures on the GPU.	120
5.4	Comparison of results based on the classification of multi-modal input (Radar+ToF) and the single modalities (Radar, ToF), all evaluated for network model $\mathcal{F}_{32}^{\text{ToF}}$.	121
5.5	Performances of the proposed GCN architectures on various embedded CNN accelerators.	121

List of Figures

1.1	Autonomous driving grades according to SAE J3016 standard.	2
1.2	Centralized and distributed processing in automotive.	6
2.1	Data processing levels in automotive.	11
2.2	Pulsed radar principle.	13
2.3	Frequency ramps of FMCW signals.	14
2.4	FMCW radar waveforms.	15
2.5	Radar angle estimation.	17
2.6	CFAR thresholding.	18
2.7	Typical radar processing steps.	19
2.8	Basic structure of common sensor data fusion architectures for ADAS.	22
2.9	Track-level fusion methods.	23
2.10	Feature extraction in camera imagery.	24
2.11	Generation of an environmental model.	27
2.12	Object-based world model.	27
2.13	Different variants of occupancy grid positions.	30
2.14	Inverse sensor models of range sensors.	31
2.15	Grid-based sensor fusion block diagram.	33
2.16	Grid-based sensor fusion.	36
2.17	Temporal alignment problem for multi-modal sensor systems.	38
3.1	Typical CNN architecture for image classification tasks.	46
3.2	Fully convolutional fusion network architecture.	49
3.3	Auto-encoder fusion network architecture.	50
3.4	Simple skipped connection in ResNet [50].	52
3.5	Auto-encoder with bypass fusion network architecture.	53
3.6	Combined occupancy grids of radar x_{OG}^{rad} and camera sensors x_{OG}^{cam}	59
3.7	Lidar point cloud used for RANSAC-based ground plane estimation.	59
3.8	RANSAC-based ground plane estimation.	61
3.9	One exemplary frame of the training data	63
3.10	Comparison of grid fusion results.	69
3.11	Computational requirements and metrics of the grid fusion networks.	70
3.12	Fusion performances over the architectural parameters.	71
3.13	Computational costs of architectural design parameters.	72
3.14	Initial and optimized version of the encoding block.	75
3.15	Initial and optimized version of the decoding block.	76

List of Figures

3.16	Computational requirements and metrics of the optimized grid fusion networks.	77
4.1	Different variants of <i>ResNet</i> -blocks.	79
4.2	State-of-the-art network models for image classification.	80
4.3	Overview of AI accelerators and processors.	83
4.4	NAS principle.	84
4.5	Macro-architectural concept of grid fusion networks.	86
4.6	The initialization of nodes.	88
4.7	Translation of adjacency matrix to the computational graph	89
4.8	Evolution of accuracies throughout the NAS.	97
4.9	Evolution of model fitness throughout the NAS.	98
4.10	Evolution of inference times and model sizes throughout the NAS.	99
4.11	Mean and standard deviation of NAS.	100
4.12	Trade-off between model size and accuracy.	101
4.13	Connection graphs of the best cell architectures	102
4.14	Comparison of discovered blocks to ones from the literature.	105
5.1	Exemplary visualization of the gesture classes.	112
5.2	Sequence lengths in the gesture dataset.	113
5.3	t-SNE representation of the GCN dataset.	114
5.4	Various GCN architectures.	117
5.5	Accuracy vs. model size comparison.	122

1 Introduction

1.1 Problem Statement and Motivation

Automated driving and advanced driver assistance systems are an ever growing topic in the automotive industry. From OEMs like BMW, Toyota and Tesla, down to suppliers, the market is trying to shape ideas of how a driverless car could look like. Yet, both technical and legal challenges remain unsolved in an economic environment of high pressure on the companies. After the first enthusiastic, young startup-like companies pushed towards fully automated driving with a large response/echo from media, the legal frameworks were developed for categorizing automated driving. Since then, even fast companies like Tesla have not achieved a level of automation where the driver is not needed anymore. The systems still have to be monitored by a human driver who is held responsible in case of an accident.

In the following, the levels of automated driving are explained with the current position of market participants.

1.1.1 Levels of Automated Driving

In the grading system of the J3016 standard of the international engineering and automotive industry association SAE, driving functions are classified into five levels of ever increasing functionality [114]. The different levels of autonomous driving are structured into combinations of what parts of dynamic driving are controlled by the driver or the vehicle. The parts of dynamic driving are the execution of longitudinal or lateral control, the monitoring of the driving environment and the fallback responsibility in case of an unforeseen, critical situation. Figure 1.1 shows the roles of driver and vehicle distributed over the SAE levels.

In the following listing, the number of sensors per automation level is derived from multiple market assumptions. The numbers are shown in a range from lowest to highest estimated amount, and they are taken from industrial players and market research institutes (IHS Markit, Infineon, Yole Developpement [130]).

Level 0: No Automation

The first grade of automation is no form of automation, as all driving tasks and the responsibility are on the human driver. Up to this date, roughly half of the car sales are vehicles with zero autonomy [130].

1 Introduction



SAE Level	SAE Name	SAE Narrative Definition	Execution of Steering/ Acceleration/ Deceleration	Monitoring of Driving Environment	Fallback Performance of Dynamic Driving Task	System capability (driving modes)	BAST Level 	NHTSA Level 
Human Driver monitors the driving environment								
0	No Automation	the full-time performance by the human driver of all aspects of the dynamic driving task	Human Driver	Human Driver	Human Driver	N/A	Driver only	0
1	Driver Assistance	the driving mode-specific execution by a driver assistance system of either steering or acceleration/deceleration	Human Driver and Systems	Human Driver	Human Driver	Some Driving Modes	Assisted	1
2	Partial Automation	Part-time or driving mode-dependent execution by one or more driver assistance systems of both steering and acceleration/deceleration. Human driver performs all other aspects of the dynamic driving task	System	Human Driver	Human Driver	Some Driving Modes	Partially Automated	2
Automated driving system ("system") monitors the driving environment								
3	Conditional Automation	driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task - human driver does respond appropriately to a request to intervene	System	System	Human Driver	Some Driving Modes	Highly Automated	3
4	High Automation	driving mode-specific performance by an automated driving system of all aspects of the dynamic driving task - human driver does not respond appropriately to a request to intervene	System	System	System	Some Driving Modes	Fully Automated	3/4
5	Full Automation	full-time performance by an automated driving system of all aspects of the dynamic driving task under all roadway and environmental conditions that can be managed by a human driver	System	System	System	Some Driving Modes		

Figure 1.1: Autonomous driving grades according to SAE J3016 standard (SAE Federal Highway Research Institute). Source <https://www.automotive-iq.com/autonomous-drive/articles/first-level-3-automated-vehicle-road-iso-functional-safety-and-analysis>, accessed 08. Feb. 2021.

Only because a car is rated without any autonomy, it does not mean that there are no advanced functionalities built-in. For example the vehicle may assist the driver with a traffic sign recognition system, which scans the scenery for traffic signs and shows the current signals and speed regulations to the driver, e.g. in the instrument display or in a head-up display.

Almost every car manufacturer has vehicles without any automation level in their fleet, but with decreasing tendency, as regulations require ever more assistance systems in a rising number of regions worldwide. The regional new car assessment programs (e.g. Euro NCAP, U.S. NCAP) are proposing a regulatory roadmap of ever more AD functions required in new vehicles in order to reduce fatal road accidents towards a minimum [94].

Level 1: Driver Assistance

In assisted driving, assistance systems take over control in either lateral or longitudinal acceleration in constrained situations. An example for a lateral control system is the lane keep assist (LKA) and for a longitudinal control system the adaptive cruise control (ACC). In both cases, the system assists in controlling one domain, but the human driver remains in the control loop to steer the other one, and to control the driving environment.

Each manufacturer has its own nomenclature of the driving functions. For exam-

1.1 Problem Statement and Motivation

Table 1.1: Estimated number of sensors per modality per SAE automation level. The columns show the various sensor modalities and types. LRR: long range radar, SRR/MRR: short or mid range radars, Int. Cam: Interior camera. ● denotes the low-end of the estimations, ●○ together denote the upper-end.

SAE	LRR	SRR/MRR	Camera	Int. Cam	Lidar	Ultrasonic
L1	● 1		●○ 1-2			●●●●●● 6
L2	● 1	○○ 0-2	●○○○○ 1-7			●●●●●●●● 8
L3	●○ 1-2	●●●● 4	●●●●○○ 4-7	○ 0-1	●○○ 1-3	●●●●●●●● 8
L4	●○ 1-2	●●●●○○○○ 4-8	●●●●●●●● 8	● 1	●○○○○○ 1-5	●●●●●●●● 8
L5	●○ 1-2	●●●●○○○○ 4-8	●●●●●●●● 8	● 1	●○○○○○ 1-6	●●●●●●●● 8

ple, the L1-system *adaptive cruise control* (ACC) is named *Adaptive Cruise Control* by Fiat, Ford, GM, VW, Volvo and Peugeot, but *Intelligent Cruise Control* by Nissan, *Active Cruise Control* by Citroen and BMW, and *DISTRONIC* by Mercedes [52].

Level 2: Partial Automation

Currently, most effort of car manufacturers is put into the development of partial automation. At this level of automation, both longitudinal and lateral control is performed by the system simultaneously. Nevertheless, the human driver still has to monitor the driving environment, as existing advanced driver assistance systems (ADAS) are heavily constrained in their operational design domain (ODD). The driver is required to perform tactical maneuvers, such as changing lanes or reacting to traffic signs correctly. Once the situation is outside the ODD, the driver has to take over control immediately and be able to react at all times.

Most car manufacturers remain within the limits of Level 2 automation, as the next step is a continuous, reliable monitoring of the driving environment, implying severe technical and legal challenges. For the enabling a higher automation level, the environmental perception has to be flawlessly redundant and robust. Most market researchers assume that all possible sensor modalities have to be employed, including the controversial Lidar technology. In contrast to other automakers like Audi or technology companies like Mobileye or Waymo, Tesla’s approach is to omit laser scanners completely. Tesla promises to enable vision sensors with sophisticated post-processing in form of deep learning algorithms, finally reaching the accuracy of

1 Introduction

Lidar sensors, thus, making those dispensable [24]. Whether this bet is turning out good for Tesla will be seen in the upcoming years. Yet, most OEM's automation systems are on SAE level 2, such as *Super Cruise* of General Motor, *Distronic Plus* of Mercedes- Benz, Nissan's *ProPilot Assist* or Tesla's *Autopilot*. As Waymo's CEO says, improving on that is only possible with Lidar technology [125].

Level 3: Conditional Automation

Here, the system can cope with most aspects of driving, also with situations that require immediate response, such as automatic emergency braking. The system recognizes situations, where it will run out of its ODD and warns the driver in due time, so they can prepare for the handover and intervene.

First big, publicly announced advances into the field of conditional automation systems were done by Audi in 2017 with the introduction of their *Audi AI traffic jam pilot*. It allows the Audi A8 to drive autonomously in the ODD of slow-moving highway traffic jams up to a velocity of 60 km/h without any input from the driver [48]. The Level 3 conditional automation system relies on a central computation platform (*zFAS*) employing four processing ICs: 1) NVIDIA Tegra K1 for 360 surround view processing of the four cameras, 2) Mobileye EyeQ3 for the stereo front camera processing, 3) Intel¹ Cyclone V chip performs the sensor fusion, calculations, 4) Infineon AURIX ensures safe, ASIL-D calculations for trajectory planning and decision making.

As the monitoring of the environment is performed by the system, it needs a fail-safe architecture. Being fail-silent by simply deactivating a function in case of a failure is not enough, as it may not result in a safe state of the vehicle. Thus, commonly redundant components are employed, be it overlapping sensors or multiple computational cores performing the same computations, as seen in the lockstep of the Infineon AURIX microcontroller.

The typical sensors used for this level of automation are at least six radars for a full 360 degree coverage, at least four cameras and at least one Lidar sensor. Additionally to the ultra-sonic sensors, also an interior camera is assumed for monitoring the driver's state.

Level 4: High Automation

Level 4 corresponds to a level of full automation, where the car can automatically drive in a wide ODD and react to some dangerous situations. Whereas it can cope with various safe traffic conditions and even react to hazardous situations, it still has the basic architecture of a vehicle designed for a human driver. Thus, the activation of Level 4 features is the choice of the driver, or else the system is only passively monitoring the situation and intervenes just in case of danger. The system is capable

¹Former Altera.

of driving independently in most environments, with some exceptions for weather or irregular environments, where the human has to take over control.

Level 5: Full Automation

Fully automated cars allow a completely different design of the complete vehicle. As the system is working reliably in all possible situations within its ODD, a human driver is not needed anymore as a fallback. Thus, any interface or method for interaction with the car's steering is dispensable; so is the complete cockpit including the steering wheel. From the user's point of view, the vehicle is becoming a simple vessel for transporting goods automatically from A to B. Whereas fully automated vehicles have been developed for decades, only very restricted ODDs were possible in the first automated highway systems in 1997 [72]. Expanding the field of operation of automated vehicles to more different and challenging traffic scenarios is the key problem of SAE Level 5 cars. Estimates predict that full automation in urban areas will not be possible before the 2050s, and full automation in all areas of life will most likely never occur [72].

1.1.2 Centralized vs. Distributed Processing

Over the years, the automotive market has evolved to a state, where suppliers of different tiers specialize on components, modules and systems. Each system's players pushed towards solutions incorporating more and more functionalities; thus, the systems were growing complex within their boundaries, regardless of potential synergies with other systems.

As an example, the radar sensor systems are designed to produce increasingly accurate and reliable track data containing a growing number of individual tracks in parallel. In this perspective, radar system suppliers, such as Bosch, Continental or Hella, have implemented sensors that incorporate extensive radar signal processing on special automotive microcontrollers. The signals except the tracking outputs are kept confidential as a business secret, not allowing the customer to access raw or intermediate states of the signal.

This is applicable to other domains, such as camera sensor systems, too. Intel's Mobileye has become the dominant player of the automotive vision market with its smart camera systems. The output of the Mobileye EyeQ3 is not the raw camera image, but a stream of pre-processed feature-level data, such as information about objects and lane markings.

The advancing levels of autonomous drive require an ever more intelligent way of combining the multi-modal information of the vehicle's sensors in order to create a reliable, robust and safe environmental model. As long as the fusion of track- and object-level data is sufficient, e.g. for ACC, the system architecture of distributed smart sensors is fine. Once the ADAS require an environmental model that sur-

1 Introduction

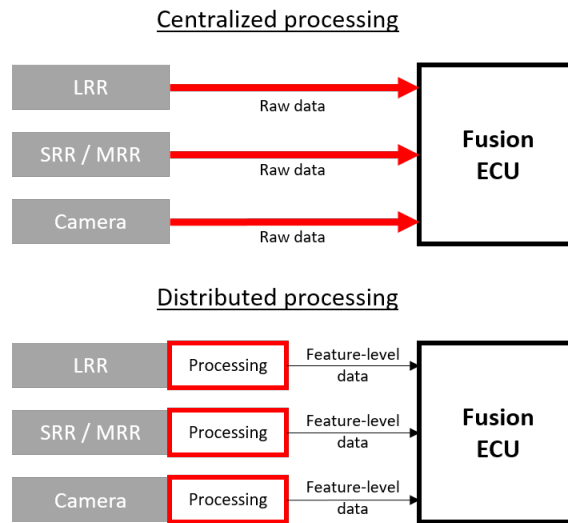


Figure 1.2: Centralized and distributed processing in automotive. [37]

passes the level of detail possible to achieve with track- and object-level data, the distributed architecture comes to its limits. In situations with dense urban traffic, the system needs knowledge about drivable, obstacle-free space rather than information about the track of other traffic participants. As the evolved landscape of suppliers and manufacturers is moving slowly, and market research shows that cars with and above Level 3 autonomous drive will not become dominant in the sales until the late 2030s, it can be expected that Level 2/2+ cars will have to rely on a distributed system architecture with smart sensor systems [130]. This explains the tendency of vehicles with high autonomy levels to have a central fusion ECU that has access to raw sensor data. A central fusion of raw data requires high-bandwidth interfaces to the sensors to stream the data of the multitude of sensors (Fig. 1.2), and it needs a high-performance compute unit that is able to process all the sensor streams in real-time.

1.1.3 Applications of Machine Learning in Automotive

In general, automated drive (SAE Level 3/4) can be seen in three layers that describe the logical steps of the automation process. Each of these layers inhibits areas where machine learning algorithms may enhance the functionality in future vehicles.

Three Layers of Automated Drive

The three layers *Sense-Plan-Act* are derived from nature, where each living being perceives its environment first, then it considers how to move through the environment, and finally actuates its muscles in order to make it happen. The same pattern

1.1 Problem Statement and Motivation

is followed in the automotive domain:

1. *The Perception Layer (Understand the present)*: The onboard sensors scan the environment and aggregate the real-time data to estimate the surroundings. By fusing the multi-modal sensors to one representation, advanced techniques such as self-localization and mapping (SLAM) or free-space estimation enable a robust perception of the vehicle.

2. *The Prediction Layer (Predict potential future states)*: The environmental representation with all the fused data of the sensors from the first layer is used to aggregate information about how the scene has changed over time. From this knowledge, it is possible to anticipate how traffic participants will behave in the near future. Certain behavior models are assigned to detected objects of interest, depending on their classification from the perception layer.

3. *Trajectory Control Layer (Planning for the future)*: This final layer ensures the proposal and execution of the trajectories. Based on the knowledge from the second layer, a safe trajectory is calculated to maneuver through the environment in local proximity of the car. At the same time, the global planner finds a route on the macro-scale to the desired destination. The local planner continuously performs safety verifications, so that the vehicle is definitely following the planned trajectory.

Application Areas of Machine Learning and Typical Computational Requirements

Machine learning algorithms can effectively enhance classical software solutions in various parts of the three-layer model for automated drive. In the *Perception Layer*, sensor-processing algorithms can be extended by the use of machine learning models such as convolutional neural networks (CNN) to extract features and detect objects in camera images or in radar data. The processing of raw sensor data with neural networks is estimated to have a data throughput of 100 to 200 GBit/s of information. Assuming a simple sensor setup of a front camera and a front radar, the processing of given data streams takes about 1 to 4 TOPs for a Level 2 car with lane keep steer (LKS). A highly automated vehicle with a full sensor suite requires around 100 TOPs for processing the first layer.

In the *Prediction Layer*, the prediction of future states of the environmental model boils down to the task of a time-series forecast—which is typically done with recurrent neural networks (RNN). The estimated data throughput in this layer is lower than the raw data processing, at around 10 GBit/s.

The *Trajectory Control Layer* outputs smooth trajectories, steering safely through the environment. Whereas plain control software produces rather abrupt steering angle corrections, ML-based trajectory generation work is superior, as seen in Tesla's LKS compared to control-software-based lane keep assist (LKA) of various OEMs. The higher the abstraction level, the less data needs to be processed. Thus, the data throughput in the third layer is estimated to lie in the order of magnitude of

1 Introduction

several 100 MBit/s. In case of a Level 2 car with LKS, 0.1 to 0.5 GOPs are required to be calculated by the hardware. In case of a Level 4 automated drive, no more than 2 TOPs are estimated.

Automotive Target Hardware

It is clear that for the raw signal processing of the multiple camera streams with hundreds of TOPs, only a number-cruncher in form of a GPU is a viable option. As GPUs themselves achieve only low ASIL¹ levels, a parallel fail-safe computational companion with ASIL-D is needed [64]. For the processing of pre-processed, thus, compressed data streams with less throughput, it is likely that automotive-grade microcontrollers will be employed for the mass market. Especially cars with Level 2/2+ automation and a distributed sensor setup will be produced in large volumes until the late 2030s. These baselines lead to the following research question.

Research Question

How well do advanced machine learning methods perform in the fusion of pre-processed sensor data in comparison to the standard Bayesian fusion technique? And how well are automated network optimization techniques, such as neural architecture search (NAS), suited for optimizing deep learning models, with the objective of the algorithms being able to perform in real-time on embedded accelerators of automotive MCUs?

1.2 Structure of the Work

Equipped with the introduction and motivation towards the topic, the following structure of the work has been outlined. In Chapter 2, the preliminary signal processing of the individual sensor modalities for autonomous drive is explained. The chapter follows the streamline from raw data up to the data format that is typically used by the successive application level (Section 2.1) and typical environmental models for autonomous robots (Section 2.2). In Section 2.3, traditional sensor fusion approaches in two major categories are described. These are the track fusion using track-level data and the grid-based fusion using raw and feature-level data.

Chapter 3 introduces the sensor fusion neural networks, which are the proposed method of fusing the feature-level data that is accessible in level 2/2+ cars. The model architecture, the required data format and the utilized dataset, the training procedure and the evaluation results are discussed in Section 3.3. As the networks are meant to be run on automotive microcontrollers, they have to be optimized accordingly. The proposed optimization process is discussed in Chapter 4, where

¹ASIL: Automotive Safety Integrity Level

1.2 Structure of the Work

a custom neural architecture search method is applied on the sensor fusion neural networks of the chapter before. Finally, the results of the optimized grid fusion networks are compared to results from literature in Section 4.3.

Additionally, in Chapter 5, the paradigm of fusing sensor data with neural networks is applied to another automotive use-case, namely the gesture recognition for multimedia controls. Again, special care is taken for the embedded optimization of the network models, as the target hardware is a low-power embedded hardware platform.

The thesis ends with the conclusive Chapter 6, where the results of the work are summarized and used for an outlook to upcoming trends in the automotive industry.

2 Preliminary Signal Processing

Raw sensor data comes in various forms, depending of the sensor type at hand. Cameras produce images in form of two-dimensional matrices with multiple color channels, radars measure the amplitude of radio-waves and convert the analog signal into a discrete array with the internal ADC¹, and lidar sensors output a point-clouds (PC), which are arrays of distance–angle relations of a laser beam. Other sensors in the automotive context, such as ultra-sonic range finders, or accelerometers and rotary encoders, each modality has its specific raw sensor format; and typically all sensors have in common that the data is processed, before it is used for an application or function.

In the context of automotive environmental perception, the processing chain for sensory data starts with the raw data being processed to feature-level data (Fig. 2.1). Typically, this step happens within the sensor, where a micro-controller prepares the signal in a way that is fundamentally needed for all applications. Once parts of the processing are done on the sensor itself, the module is labeled as a *smart sensor*. The common pre-processing steps of automotive radar and camera sensors are discussed in Section 2.1, as they are the prerequisite for all environmental models of this work.

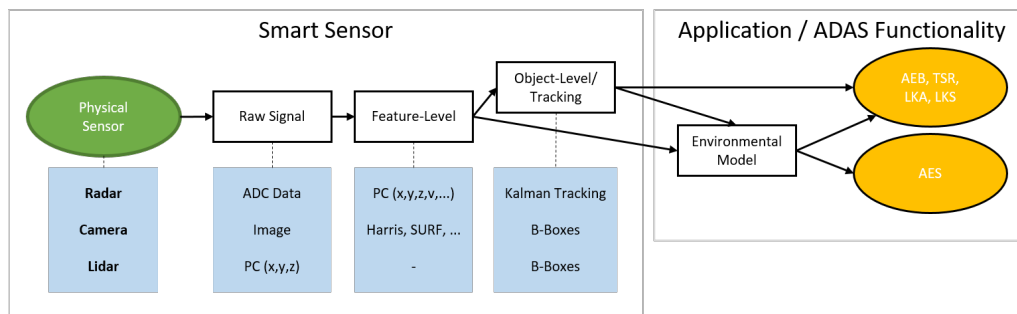


Figure 2.1: Typical data processing levels in automotive, from raw sensor signals over feature-/object-level data to ADAS functionality. Raw signals: ADC data, images, point-clouds (PC); feature-/object-level data: PC, image feature descriptors (e.g. SURF²), tracks, B-Boxes (bounding boxes).

¹ADC: analog-to-digital converters. Often raw radar data is referred to the data coming directly from the ADC.

²Speeded-up robust features (SURF) discussed in Subsection 2.1.3.

2 Preliminary Signal Processing

Today’s advanced driver assistance systems (ADAS) range from ease-of-use functions, such as traffic sign recognition (TSR) or adaptive cruise control (ACC), to important safety-enhancing functions, such as automatic emergency braking (AEB), lane keep assist and steer (LKA, LKS) or automatic emergency steer (AES). ADAS rely on a perception of the surroundings, which is granted by the typically smart sensors on the vehicle. Based on the pre-processed data, an integrated model of the environment is calculated, which is then used in the application layer. Different types of environmental representations are discussed in Section 2.2, together with the steps needed to calculate them.

Research about automotive environmental perception mostly focuses on processing raw sensor signals. These works assume the access to raw sensor data, which is valid for only a subset of vehicles available on the automotive market. Access to raw data is needed for functionalities and routines in cars with automation levels 3, 4 and finally 5. The amount of vehicles capable of these levels of AD is vanishing, when compared to today’s cars on the road and the cars anticipated to be built in the next 15 to 20 years [130]. But ADAS such as lane keep assist, lane departure warning, or automated emergency braking, in cars with automation levels up to 2 have to rely on a sensor suite of mass-produced, serial cars in a highly cost-competitive market.

2.1 Raw Sensor Data Processing

This section covers the signal processing of raw sensor data up to the level at which a central fusion ECU receives the data. First, the processing chain of automotive frequency-modulated continuous-wave (FMCW) radars is described; from ADC data to target lists containing range, angle, and velocity. Object-level tracking, which is today’s standard usage of radar data, is covered briefly in order to complement the novel, lower-level processing of radar. Then, the working principle of automotive vision systems is described in form of feature extraction and detection with traditional and novel DL-based methods. Finally, basic working principles of other sensors are sketched.

2.1.1 Radar Signal Processing

Pulsed Radar Principles

The basic operation mode of radar sensors is the pulsed radar. The radar is capable of identifying the range and velocity of reflecting targets by sending (Tx) electromagnetic waves at regular time intervals and receiving (Rx) the echos of these pulses. The waves can be described as harmonics with an amplitude A , frequency f and phase ϕ according to

$$s(t) = \text{rect}\left(\frac{t}{T_p}\right) \cdot A \cdot \cos(2\pi ft + \phi). \quad (2.1)$$

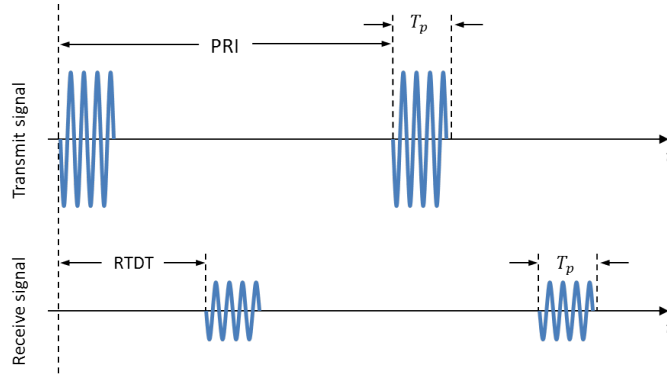


Figure 2.2: The working principle of pulsed radar.

The period between transmitting and receiving the signal is the round trip delay time (RTDT) τ . The range r to the reflecting target can be calculated easily with

$$r = \frac{c \cdot \tau}{2}, \quad (2.2)$$

by knowing that the radio-waves have to travel the distance twice with the speed of light c . In the pulsed radar, the pulse duration T_p and the pulse repetition interval (PRI) plays an important role in the precision of the radar system. The range resolution ρ_r is defined by

$$\rho_r = \frac{c \cdot T_p}{2}, \quad (2.3)$$

which describes the ability of the system to separate two individual targets, instead of detecting one. Two targets can be resolved only, if the received signal is longer than twice the pulse duration T_p . All received shorter than $2T_p$ will be recognized as one [1]. The range of operation of a pulsed radar system (r_{min} , r_{max}) is defined by the PRI and T_p . The RTDT may not be smaller than T_p ; else, the transmitted and received signal are interfering. On the other side, it may not be larger than the PRI; else, the received signal loses reference to the transmitted signal, resulting in range ambiguities.

$$r_{min} = \frac{c \cdot T_p}{2} \quad (2.4)$$

$$r_{max} = \frac{c \cdot PRI}{2} \quad (2.5)$$

To encounter range ambiguities, it is common to change the PRI over multiple measurements. This way, the correct range remains the same, but the ambiguous ranges vary over time and, thus, can be discarded.

2 Preliminary Signal Processing

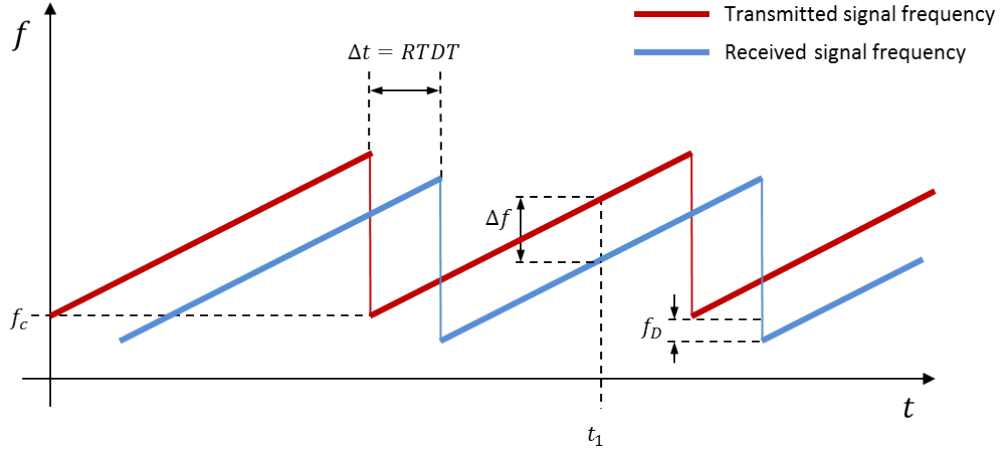


Figure 2.3: Frequency ramps of transmitted and received FMCW signals.

Automotive Frequency-modulated Continuous-wave Radar

For short range automotive scenarios, continuous-wave (CW) radars are more suitable, as short distances require short pulses with limited energy [102]. A CW radar operates on a known frequency, transmitting and receiving simultaneously. Without signal modulation, the CW radars are not able to resolve the range, but the velocity of a reflecting target by its Doppler frequency shift. The change in frequency tells whether the object is moving towards the radar or away from it—providing the radial velocity.

The frequency-modulated continuous-wave radar is capable of measuring both, the distance and the velocity to objects. The signal frequency of a linear FMCW radar follows a predefined ramp (Fig. 2.3), where the modulating frequencies are added on a carrier frequency f_c . Each object that is illuminated by the radar reflects a signal that contains information about the object's distance and velocity. The distance is calculated according to Eq. 2.2, but with the delay time τ as the time difference between Tx and Rx ramps (Fig. 2.3). A positive Doppler-shift f_D indicates a relative velocity towards the radar; a negative f_D is the result of a movement away from the radar.

The crucial part of FMCW radars is the standing wave between sensor and object, which results from the interference of the continuously transmitted waves. This signal is described by the intermediate frequency (IF), which can be calculated for a specific time t_1 by mixing the Tx and Rx signals. The result can be decomposed into the frequency Δf for static objects and $\Delta f - f_D$ for moving objects; and it can be separated in a true range-related *beat frequency* f_{beat} , depending only on the RTDT and to the *Doppler frequency* f_D . Thus, the IF is a combination of Doppler frequency shift and the frequency shift based on the RTDT. By mixing of the high-frequency transmit and receive signals, the IF is converted to baseband, resulting in

2.1 Raw Sensor Data Processing

a much lower frequency. Thus, the signal processing is easier to handle afterwards. The mixing process is called *dechirp-on-receive*.

The waveform s_{Tx} of one frequency ramp is represented by its analytical equation

$$s_{Tx}(t) = \text{rect}\left(\frac{t}{T_c}\right) \exp\left[2\pi j\left(f_c t + \frac{1}{2}\alpha t^2\right)\right], \quad (2.6)$$

where the FM sweep rate α describes the change of the signal frequency over time in Hz/s, t is the time¹ and T_c is the duration of the chirp [27]. As the signal travels to the target and back, the waveform stays the same, but a shift by the RTDT occurs:

$$s_{Rx}(t) = \text{rect}\left(\frac{t - \tau}{T_c}\right) \exp\left[2\pi j\left(f_c(t - \tau) + \frac{1}{2}\alpha(t - \tau)^2\right)\right] \quad (2.7)$$

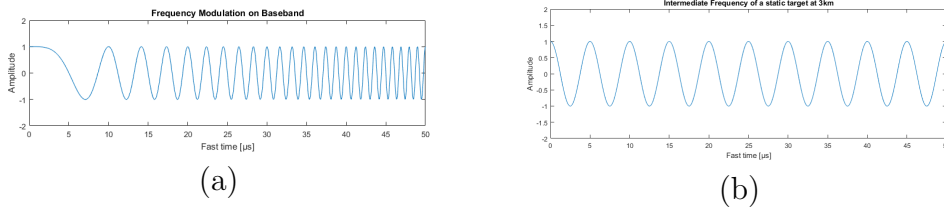


Figure 2.4: FMCW radar waveforms. (a) Frequency sweep of one chirp described by Eq. 2.6. (b) Resulting beat frequency.

After the dechirp-on-receive of s_{Tx} and s_{Rx} , the intermediate frequency is calculated by applying the simple multiplicative model of a mixer² and discarding the components of higher frequency³ [86]:

$$s_{IF}(t) = s_{Rx}(t) \cdot s_{Tx}(t) \quad (2.8)$$

$$s_{IF}(t) = \exp\left[2\pi j\left(f_c t + \frac{1}{2}\alpha t^2\right) - 2\pi j\left(f_c(t - \tau) + \frac{1}{2}\alpha(t - \tau)^2\right)\right] \quad (2.9)$$

$$s_{IF}(t) = \exp\left[2\pi j\left(f_c \tau + \alpha \tau t - \frac{1}{2}\alpha \tau^2\right)\right] \quad (2.10)$$

Typically, the raw radar signal is a discrete version of the IF-signal, which is sampled by an ADC. As one can see, s_{IF} is a function of the fast-time and inhibits only a

¹Time for one chirp is usually denoted as the fast-time, whereas the time over multiple chirps is denoted as the slow-time.

²Mixing of two harmonics: $\cos(x) \cdot \cos(y) = \frac{1}{2} [\cos(x - y) + \cos(x + y)]$ with $e^{ix} = \cos(x) + i \cdot \sin(x)$

³The hardware in the receiver is designed for the frequencies of the IF signal. Thus, the high frequency components are dampened and can be discarded. Also, the $\text{rect}()$ -function is discarded for a better readability of the equations.

2 Preliminary Signal Processing

single linear term. Thus, for a static environment, the IF-signal is a sinusoid function with a fixed frequency. However, if the target is moving relative to the radar, the RTDT changes during the s_{IF} sampling and causes in phase and frequency shifts. The range-dependent beat frequency f_{beat} for a fixed time instance t is

$$f_{beat} = \alpha \cdot \tau = \frac{2\alpha}{c}r, \quad (2.11)$$

with r as the slant range. As τ can be translated directly into the slant range of the target, also the range resolution ρ_r is directly proportional to the frequency resolution of the beat frequency δf_{beat} [86]:

$$\rho_r = \frac{\delta f_{beat} c}{2\alpha} = \frac{c}{2\alpha \cdot PRI} = \frac{c}{2B}, \quad (2.12)$$

with B as the frequency bandwidth of one chirp. Eq. 2.12 indicates that the range resolution is just dependent of the bandwidth of the transmitted frequency ramp [103]. Fig. 2.4b shows the intermediate frequency of a target located at a distance of 3 km ($\tau = 10$ s), interrogated by a radar with $T_c = 50 \mu s$ and $B = 1$ MHz.

The Range-Doppler Domain

Eq. 2.11 shows that the beat frequency of static objects is only depending on their slant range to the sensor. For multiple objects, the beat signal is the interference of the individual tones of each reflector, with each tone's frequency encodes the slant ranges. The range information s_r is extracted from Eq. 2.10 by applying the Fourier transform. Typically, the transformation is done with the fast Fourier transform (FFT) algorithm. This first FFT is also denoted as *range FFT* or *1st-stage FFT*.

In order to determine the velocity information of an object, its phase change has to be observed over the slow-time. This is done by sending a rapid sequence of N_c successive frequency chirps (Fig. 2.3) and accumulating the received range information s_r over the time of one frame of N_c chirps. The frequency and phase of the beat signal of the n -th chirp can be calculated as [106]:

$$\phi_0 - 2\pi \frac{2v}{c} f_c n T_c - 2\pi \left[\frac{2v}{c} (f_c + n\alpha) + \frac{\alpha}{T_c} \right] t, \quad (2.13)$$

where v is the radial velocity and $2\pi \frac{2v}{c} f_c n T_c$ corresponds to the velocity-dependent phase change from chirp-to-chirp.

The velocity information is also extracted with a Fourier transform, but across the dimension of the chirps in slow-time. This FFT is also denoted as the *Doppler FFT* or *2nd-stage FFT*. The resulting spectrum after the two FFTs corresponds to the range-Doppler image (RDI), which incorporates the velocity information over the ranges.

Angle Estimation

The extraction of range and velocity from the ADC data is covered until now. The third aspect of automotive radar processing is to gain knowledge about the direction of arrival Θ of the reflected waves. In order to estimate Θ , an array of receive antennas is necessary, where the incoming wavefront arrives at different times at the individual antennas, i.e. has to travel an additional distance Δ (Fig. 2.5). The delta is depending on the antenna spacing d and the angle of arrival according to $\Delta = d \sin(\Theta)$. It leads to a phase change ω_n for the n -th antenna according to [106]:

$$\omega_n = \frac{2\pi}{\lambda} n d \sin(\Theta). \quad (2.14)$$

Eq. 2.14 shows that—for a given amount of Rx antennas with a fixed spacing d —the angle of arrival $\hat{\Theta}$ can be derived from the phase changes over the antenna elements by estimating $\hat{\omega}$:

$$\hat{\Theta} = \sin^{-1} \left(\frac{\hat{\omega} \lambda}{2\pi d} \right). \quad (2.15)$$

The phase change $\hat{\omega}$ is typically estimated using another Fourier transform into the third dimensions over the antenna elements. This is the *angle FFT* or *3rd-stage FFT*.

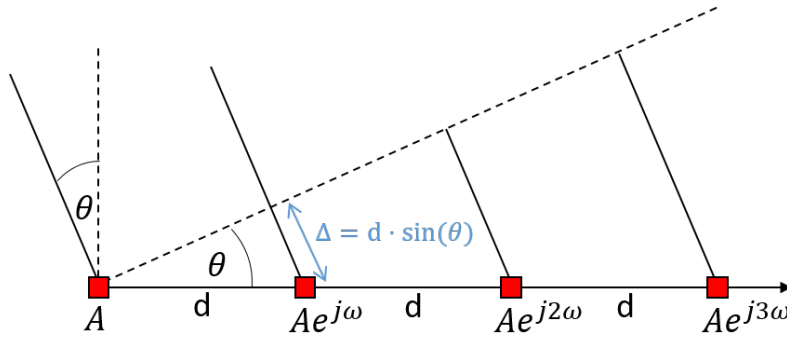


Figure 2.5: Radar angle of arrival estimation using multiple Rx antennas in a uniform linear array.

Thresholding Targets

In order to extract the desired targets from the noisy range-Doppler images, thresholding is applied to the data. In real-world signals the noise level is not constant spatially and temporally due to random clutter, an adaptive thresholding approach is needed. The *constant false alarm rate* (CFAR) thresholding is the state-of-the-art method for radar signal processing. Its principle is to compare the data cell under test (CUT) with the neighboring cells (Fig. 2.6). The CUT (central cell in Fig. 2.6)

2 Preliminary Signal Processing

is surrounded by the *training cells* x , which are used for the thresholding value z for the CUT. The training cells are shifted symmetrically apart from the CUT for signals with a high resolution and large array size. The cells between CUT and training cells are the *guard cells*. The fundamental design parameters of a CFAR thresholding filter are the number of training cells N , the number of guard cells and the detection scheme.

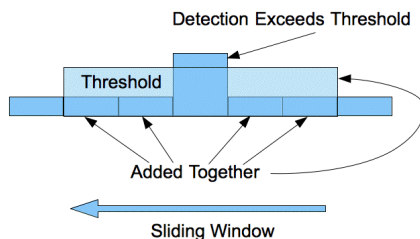


Figure 2.6: CFAR thresholding. Source: https://commons.wikimedia.org/wiki/File:Constant_false_alarm_rate.png accessed on 09. Feb. 2021.

There are various detection schemes for CFAR, which describe, how the thresholding value is determined. The most simple scheme is the *cell averaging* CA-CFAR, where the values of the training cells are averaged according to $z = \frac{1}{N} \sum_{i=1}^N x_i$. However, high-valued outliers may disturb the mean value of the training cells. Thus, the *ordered statistic* OS-CFAR first ranks the training cells by value and then discards a fixed amount of the highest values before calculating the mean with the remaining.

The CFAR principle is displayed as a filter for a one-dimensional array in Fig. 2.6, but it can also be extended to arbitrary dimensions. In case of a 2D CFAR, a band of training cells around the CUT are taken into consideration for z . Naturally, the higher-dimensional thresholding filters are more accurate, but come to the cost of increased computational load. A common trade-off is to separate the 2D filtering into two orthogonal 1D tasks.

Putting It All Together

The processing steps of an automotive FMCW radar system are depicted in Fig. 2.7. Multiple receive antennas record a three-dimensional radar cube consisting of ADC data with the three dimensions: fast-time of single chirps, the slow-time over the chirp sequence and the dimension of the antenna elements. The 1st-stage FFT transforms the fast-time to range information, the 2nd-stage FFT extracts the velocity information from the phase change over the slow-time and the 3rd-stage FFT estimates the angle of arrival from the phase change over the antenna elements. Thus, the known features are range r , radial velocity v_r and angle Θ . When converting polar to Cartesian coordinates, one can compile these information to a target description in form of $\mathbf{t} = [x \ y \ v_x \ v_y]$. Typically, the radar sensor is outputting

a target list with N entries in form of $\mathbf{T} = [\mathbf{t}_1, \mathbf{t}_1, \dots, \mathbf{t}_N]^T$. Current automotive radar systems generate hundreds of targets per cycle, which, plotted in Cartesian coordinates, are interpreted as a point cloud. Especially, when the array of antennas allows to receive elevation information, thus, generating three-dimensional point clouds with $\mathbf{t} = [x \ y \ z \ v_x \ v_y \ v_z]$. The raw point cloud is used for tracking, which is explained in the following section (Sec. 2.1.2).

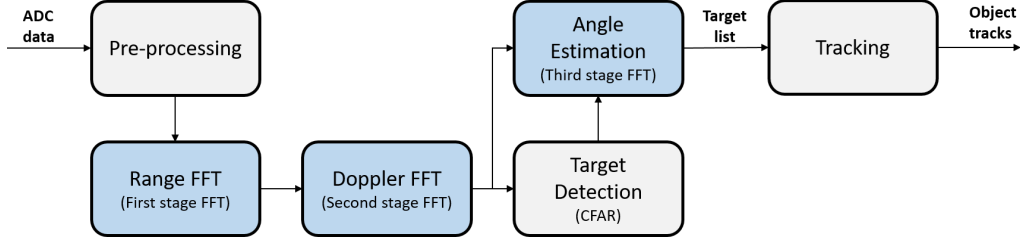


Figure 2.7: Typical radar processing steps utilizing three FFTs until tracking.

2.1.2 Track-level Data Processing

Kalman Filter

With the estimated parameters of an object (Sec. 2.1.1) in form of range, velocity, and angle, the Kalman filter describes the underlying object's motion by estimating its state. The state can range from one, up to three dimensions, describing the objects location in space. Additionally to the location, a motion model is assumed, typically a constant velocity or a constant acceleration. The linear Kalman filter is described by the linear stochastic equation

$$x_{k+1} = F_k x_k + G_k u_k + v_k, \quad (2.16)$$

where x_k is the object's state at timestep k , F_k is the state transition model matrix, G_k is the control model matrix, u_k are the known controls acting on the object and v_k is an additive random noise disturbing the motion [69, 78]. The state can be restricted to two or three dimensions to describe the spatial location, but also extended to a full description of the object with

$$x = [x \ y \ v_x \ v_y \ a_x \ a_y]^T, \quad (2.17)$$

where the location, velocity v and acceleration a of the object is incorporated. The Kalman filter assumes measurements z_k as a noisy function of the true state x_k , thus, also the sensor measurements are described by linear functions of the state

$$z_k = H_k x_k + w_k, \quad (2.18)$$

2 Preliminary Signal Processing

where H_k is the measurement model matrix and w_k the additive noise [69, 78]. The measurements can consist of any part of the measurement model matrix, so that multi-modal sensor fusion can be done by designing H_k in a way that the different sensor modalities' properties are incorporated in it. For example, the location can come from the camera and the radar, and additionally the velocity and acceleration can be provided only by the radar.

Kalman Filter Loop

The Kalman filter is initialized with the best estimate of the state, $x_{0|0}$, and the state covariance, $P_{0|0}$. Starting with a bad or random guess is also feasible, but the filter requires some time to adapt to the situation. After the initialization, the following continual loop is performed.

1. The next state is determined by propagating the current state using the motion equations:

$$x_{k+1|k} = F_k x_{k|k} + G_k u_k. \quad (2.19)$$

Analogically, the covariance matrix is propagated as well:

$$P_{k+1|k} = F_k P_{k|k} F_k^T + Q_k. \quad (2.20)$$

The subscript notation $k + 1|k$ indicates that the quantity is the optimum estimate at the $k + 1$ step propagated from step k . This estimate is often called the *a priori* estimate [17].

The measurement prediction at the updated time is calculated from the propagated state:

$$z_{k+1|k} = H_{k+1} x_{k+1|k}. \quad (2.21)$$

2. The difference between the predicted measurement and the actual measurement is used to correct the propagated state at the updated time. The correction requires computing the Kalman gain, which, in turn, first needs the measurement prediction covariance (innovation). The Kalman gain is calculated with the innovation

$$S_{k+1} = H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1}, \quad (2.22)$$

according to:

$$K_{k+1} = P_{k+1|k} H_{k+1}^T S_{k+1}^{-1}. \quad (2.23)$$

3. Next, the predicted estimate is corrected with the measurement¹. The corrected state estimate uses the subscript notation $k + 1|k + 1$, and it is computed from

$$x_{k+1|k+1} = x_{k+1|k} + K_{k+1} (z_{k+1} - z_{k+1|k}). \quad (2.24)$$

¹On the assumption that the estimate is a linear combination of the predicted state and the measurement [69].

The corrected state is often called the a posteriori estimate of the state because it is derived after the measurement is included.

The covariance matrix is corrected with the innovation and the Kalman gain:

$$P_{k+1|k+1} = P_{k+1|k} - K_{k+1}S_{k+1}K'_{k+1}. \quad (2.25)$$

Finally, the measurement is computed based upon the corrected state. This is not a correction to the measurement, but it is the best estimate of what the measurement would be based upon the best estimate of the state.

Extensions of the Kalman Filter

The Kalman filter is based upon the assumption that the errors in the system are Gaussian. Also, the above mentioned linear Kalman filter works well only on linear systems. Several extensions of the linear Kalman filter were introduced to cope with this limitations. The extended Kalman filter (EKF) and the unscented Kalman filter (UKF) are the most prominent successors, as they are applicable on nonlinear systems, too.

The EKF linearizes about an estimate of the current mean and covariance with the Jacobian, approximating the nonlinearity to a linear estimate. The linearization requires the functions to be differentiable [67]. The UKF uses a deterministic sampling technique (unscented transformation) to pick a minimal set of sample points around the mean. This set of sample points propagated through the nonlinear functions and form a new mean and covariance estimate. With the UKF, the computation of the Jacobian is omitted, thus, complex functions or not differentiable functions can be estimated, too [67].

Multiple and Extended Target Tracking

With the rapid advance in radar and lidar sensor technology, angular accuracies improve and, in equal measure, the number of reflection points per object increase. Thus, the assumption that each object is represented by one track and each track is maintained by one target does not hold anymore. Tracking an object that is perceived by the sensors with multiple, individual target detections, is called *extended object tracking* or *extended target tracking*. Popular extended object tracking approaches are the random matrix approaches and the star-convex shape approaches [45], focusing only on a single extended target.

Similar to point-target models, the tracking of multiple extended objects is performed with *Random Finite Sets* (RFS) [110] or *Probability Hypothesis Density* (PHD) filters [36]. However, the data association of targets to objects and objects to tracks has factorial scaling, thus, becomes intractable rapidly [45]. Several approaches exist to ease the complexity with approximations, such as gating, clustering, or distance partitioning.

2 Preliminary Signal Processing

Still, novel solutions are needed to tackle this complex assignment problem. Besides non-probabilistic methods (*Greedy Randomized Search*, *Lagrangian Relaxation*), machine learning based methods (*Conditional Random Fields*, *Belief Propagation*, *Markov Chain Monte Carlo*, RNN-based Deep Neural Networks) are investigated [35].

Track Fusion Techniques

Assuming N sensors of any kinds of sensor modalities, the sensor data can be fused in various stages of processing for tracking. The fusion stage can be placed after each block of typical pre-processing steps from raw data, feature-extraction to tracking (Sec. 2.1.1). Depending on which data is used, the fusion is denoted as *low-level* (raw data), *feature-level* (pre-processed data) or *track-level* (track data) fusion [5].

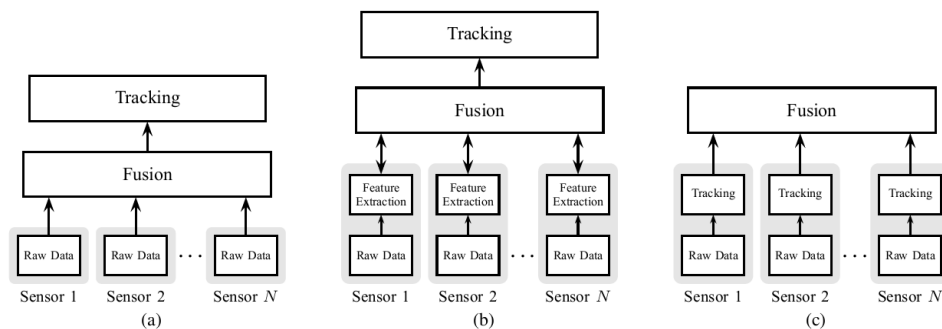


Figure 2.8: Basic structure of common sensor data fusion architectures for ADAS. (a) low-level fusion, (b) feature-level fusion and (c) high-level track fusion. Source: [5].

Recall Chapter 1.1.2, where it is pointed out that current mass-produced cars are equipped with distributed smart sensor systems, which incorporate large portions of the processing chain. Radar sensors, for example, produce a list of tracks as an output. Also, assuming low-level sensor fusion, the temporal synchronization of this data forms a critical task, which is eased in the track domain, as by using the Kalman filter, states can be predicted to a uniform timestep. Consequently, nowadays fusion ECU employ track-level fusion. Well-known track-level fusion techniques are either a Kalman filter or the following covariance-based approaches [85]: *Covariance Intersection* (CI), *Cross-Covariance* (CC), *Covariance Union* (CU).

A comparative research of the listed track-level fusion algorithms compares the computational times for one cycle of update and prediction [85]. The results of CI, CC and CU are compared to the usage of an asynchronous Kalman filter. The measured computational costs show that CC and CI need less resources than the Kalman filter, but the CU exceeds it by a factor of 2.62 (Tab. 2.1). The track fusion

Table 2.1: Comparison of track fusion algorithm. Performances shown in RMSE for input tracks with low and high correlation. Source: [85]

Method	Rel. cost	Performance (low corr.)	Performance (high corr.)
Kalman filter	1.00	0.181	0.369
CC	0.69	0.131	0.167
CI	0.87	0.178	0.199
CU	2.62	0.137	0.172

algorithms are also compared in terms of their mean RMSE¹. In terms of fusion quality, the Kalman filter performs well for uncorrelated input tracks, but with correlated tracks to fuse, it loses its ability to fuse robustly. The covariance-based approaches perform equally well with only slight differences (Tab. 2.1). Conclusively, the cross-covariance approach has best performance for the lowest costs, in contrast to the Kalman filter, which loses quality at highly correlated tracks.

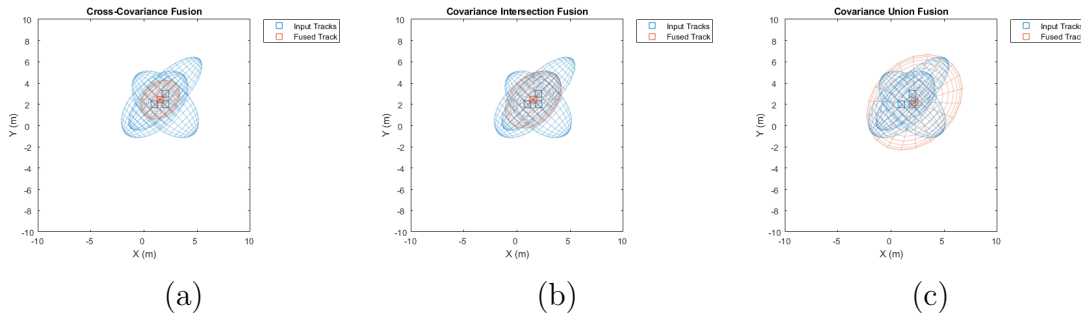


Figure 2.9: The track-level fusion techniques applied for the fusion of three tracks within one timestep. (a) Cross-covariance, (b) covariance intersection, (c) covariance union. Source: Mathworks MATLAB Documentation².

2.1.3 Camera Sensor Processing

The market trend for automotive camera systems is clearly growing in the upcoming years, as their demand for ADAS is rising, based on the increasing legal safety requirements for new cars [63]. The trend invokes the production of low-cost, mass-

¹RMSE: root mean squared error.

²fusexcov: <https://www.mathworks.com/help/fusion/ref/fusexcov.html>,
 fusecovint: <https://www.mathworks.com/help/fusion/ref/fusecovint.html>,
 fusecovunion: <https://www.mathworks.com/help/fusion/ref/fusecovunion.html>.

2 Preliminary Signal Processing

produced systems that consist typically of two parts: the remote sensor itself, collecting light from the environment and the processing computer. The processing computer pre-processes the signals in order to send the data to other vehicle subsystems, such as the fusion ECU. It enhances image quality and even performs feature extraction of various extent.

In this subsection, a brief overview about the camera signal processing is given. A focus is particularly on the feature extraction of images, but not on the optics nor low-level processing, such as image/pixel generation or epipolar geometry.

The first step of extracting features is applying filters to the raw images in order to extract some handcrafted features, such as edges or significant landmarks. A very basic, but yet common edge detector is the Sobel filter. Every pixel p_i in the image (Fig. 2.10a) with its directly surrounding pixels is multiplied with a specific matrix S and produces the output pixel g_i . For the complete image A , the Sobel filter is used to extract horizontal (in x-direction) edges

$$G_x = S_x * A = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A, \quad (2.26)$$

and vertical (in y-direction) edges

$$G_y = S_y * A = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A, \quad (2.27)$$

which both are combined to the output image (Fig. 2.10b) with highlighted edges $G = \sqrt{G_x^2 + G_y^2}$. According to this principle, many other filters are designed in order to extract certain features from the raw signal, such as the Prewitt filter, the Canny edge detector or the Laplace operator. Fig. 2.10c shows the result of the Harris corner detector. Fig. 2.10d shows the result of the Harris corner detector.



Figure 2.10: Feature extraction in camera imagery.

A step further, in tasks like tracking an object in an image or the optical flow of the image, basic keypoints have to be tracked over multiple frames, thus, have to be re-identified. Once distinct low-level features are discovered in an image, they are described by a feature descriptor that uniquely identifies that feature. Typically,

each keypoint incorporates various information about the surrounding. In case of the highly influential SIFT descriptor (scale-invariant feature transform), each keypoint is described by scale- and rotational-invariant features (Fig. 2.10d) [82]. This is achieved with the use of the Difference of Gaussians of various down-scaled versions of the input and calculating the Laplacian of Gaussian approximations. These are scale-invariant, but require a high amount of computations. This problem is addressed by successors of SIFT, such as SURF (speeded-up robust features) [14].

A real breakthrough in the camera sensor processing with the rise of deep learning, particularly in the field of convolutional neural networks (CNN). A main property of CNNs is the ability to learn filters from the training data automatically, in contrast to the handcrafted filters (Sobel, Prewitt, etc.). Those filters are convolved with the input and produce an output that has highly distinctive features, if the training of the filter was done correctly. A CNN uses multiple, successive layers of filters, ideally each with a different level of abstraction of features, starting from basic, low-level features as corners and edges, to ever more detailed representations (e.g. in case of face recognition: eyes, nose, mouth).

2.1.4 Other Sensor Preparations

Whereas camera and radar sensors offer the widest area of applications, other exteroceptive sensors are also present in automotive perception systems. Ultrasonic sensors are used to sense in the vicinity of the vehicle and are typically the base of today's parking assistance systems. Yet, lidar sensors are uncommon on the streets, but they are expected to significantly grow as the technology evolves and regulatory rules require more and more sensor systems.

Lidar Sensors

Analogically to radar, the lidar (light detection and ranging) is measuring the time-of-flight of an electromagnetic signal, but in contrast to the FMCW radar, it uses short pulses of directed laser with a much shorter wavelength. Common wavelengths in the automotive domain range from 850 nm to 1 μm [139]. There are two types of lidar sensors, the rotating and the solid-state ones. In case of the rotating lidars, multiple (few dozens) lasers are stacked vertically and this array is rotated around the vertical axis, covering a sweep-angle and elevation. The solid-state lidars have less active lasers, but these are directed through micro-electro-mechanical system (MEMS) mirrors within the semiconductor. With the help of two rotating MEMS mirrors, the rays are steered along two axis, covering azimuth and elevation. The solid-state lidars contain no mechanical moving parts, thus, the lifetime is longer, the production costs lower and the package is smaller.

Lidars are viable option for environmental perception, mainly because of the precise angular resolution. The MEMS-based Continental HRL131 lidar achieves an-

2 Preliminary Signal Processing

gular resolutions of 0.05° in azimuth and 0.075° in elevation. Typically, the lidar output has thousands of reflections in the point cloud per measurement cycle.

A general trend is observable in the comparison of radar and lidar sensors: the superior angular accuracy of the laser scanners are approximated and targeted by next generation radar sensors. Those high-resolution radars (HRR) employ large MIMO antenna arrays with every higher amounts of virtual antenna elements. The outputted radar point cloud contains ever more entries, now in the range of hundreds of targets per cycle, but this number will rise significantly. The goal of the HRR is to achieve an imaging radar system, able to produce lidar-like fine-grained outputs. In contrary, the lidar technology is investigating the application of modulated laser signals, such as FMCW. It is expected to enhance the capability of detecting moving objects, as direct velocity measurements are possible with FMCW. It can be concluded that the radar and lidar sensor technologies are getting closer to each other.

Ultrasonic Sensors

Ultrasonic sensors are the most inexpensive sensor for automotive environmental perception, and yet, the most wide spread ones. As their range is limited to just a few meters, the dominant use-cases are parking assistance applications. The output of an ultrasonic sensor is an analog voltage that is correlated to the distance of the closest object. The range is determined by sending modulated ultrasonic signals and receiving the echo. As one sensor provides only one distance value per timestep, typically multiple ultrasonic sensors are placed around the bumpers of the vehicle in order to calculate the angle to the echoing target by triangulation.

2.2 Environmental Modeling Schemes

Deriving an environmental model directly from the surrounding world is not possible without sensors. Those sensors interrogate the world and generate measurements z (Fig. 2.11a) The world model M is created from those measurements z_0, z_1, \dots, z_t over time t (Fig. 2.11b). Because there are multiple sources of inaccuracies and errors in the environmental model generation, the reality is approximated with probabilistic approaches. In the following sections, the object-based and the grid-based environmental model are described.

2.2.1 Object-Based Environmental Models

In object-based environmental models, the world only consists of dynamic objects with an associated motion model. This approach allows precise predictions about the dynamic behaviors of traffic participants based on the track history. It can

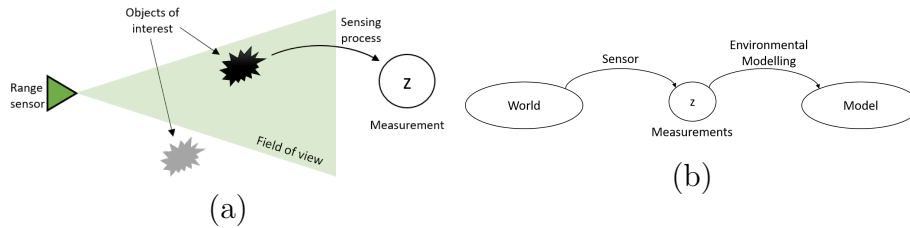


Figure 2.11: (a) Range sensor measurements. (b) Generation of an environmental model of the world with measurements from a sensor.

incorporate multi-modal sensor information in form of sensor-agnostic targets¹ and generate a model that spans over vast dimensions, e.g. ranging hundreds of meters in front of the car. On the downside, this environmental model does not incorporate information about the static environment, such as the drivable object-free space, which is needed for advanced maneuver planning.

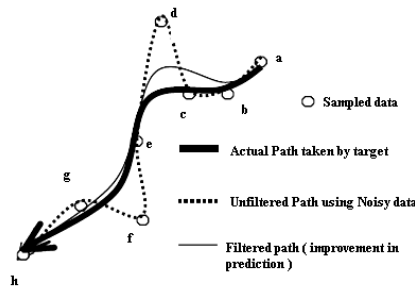


Figure 2.12: Object-based world model. Source: [133].

Track Management

The typical cycle for generating and maintaining the object-based environmental model in each measurement readout are the following steps [17]:

1. *Data association*: Starting from the target list (Sec. 2.1.2), each target at this timestep is associated to an existing track. The association task can be done with a simple thresholding by selecting the track that is closest to the given target or within a given distance threshold. Other approaches are of statistical nature, such as the Probabilistic Data Association Filter (PDAF) [12] or in case of multiple targets, the Joint Probabilistic Data Association Filter (JPDAF) [28].

¹Targets in this context are seen as a collection of properties bound to a spatial location and usually come as part of a target list.

2 Preliminary Signal Processing

The set of targets that are successfully assigned to an existing track are used for updating the tracks in the second step. But, not all targets are associated to an existing track in this stage. The remaining ones are likely to belong to a new object that is not tracked yet, thus, a new track will be initialized (third step). Similarly, tracks without any assigned targets will be removed from the track list, as the underlying object is likely to have moved out of the field of view (fourth step).

2. *Track update:* Confirmed existing tracks are updated either with their associated targets in each measurement cycle, or if none is available, the prediction of the track into the current timestep based on its known motion model. Incorporating a new measurement into a track can be done with various algorithms, such as the multiple hypothesis tracker (MHT) or the alpha-beta tracker, but most tracking systems apply the Kalman filter [69] (Sec. 2.1.2), or one of its generalized successors that are capable of handling nonlinear motion models, too (extended Kalman filter (EKF), unscented Kalman filter (UKF), particle filter).
3. *Track initialization:* All unassociated targets are generating a new track. When the object-based environment itself is initialized, all targets are creating new tracks. Once the environment is set up, only those are used for spawning tracks that are not used in the track update (second step). In order to avoid false alarms, tracks have two states, the *tentative* and *confirmed* state. A track is tentative, if over m successive measurement cycles n times a track has been assigned to it with $m > n$, and typically $m = 5$ and $n = 3$.
4. *Track deletion:* If a track is not associated with targets, it is subject to deletion. Typically a condition has to be fulfilled to remove this track from the track list, e.g. in the past m measurement cycles, too few (n out of m) or no targets at all have been associated, or the uncertainty in form of the covariance matrix is too large.

2.2.2 Grid-Based Environmental Models

Grid-based models represent the world by subdividing it into small areas, where each of the areas are described with properties of interest. Depending on the application, those areas can be of different size, but typically, they are of uniform size and spacing, describing a checkerboard-like grid in the two-dimensional case. In 3D, the world is subdivided into small, uniformly sized cubes. Whereas the cells can contain many features, the most important one is the probability of occupancy, hence the name *occupancy grid* (OG).

Classical and Continuous Occupancy Mapping

The classical way of obtaining the occupancy probabilities is by estimating the unknown underlying probability density function (PDF) with a Bayes filter recursively over time. The occupancy probability is calculated for each grid cell $m_i, i \in [0, I]$ of that map $m = [m_0, m_1, \dots, m_I]^T$ individually by approximating the posterior probability $p(m|z_{1:t}, l_{1:t})$, where $z_{1:t}$ is the measurement history, and $l_{1:t}$ the past known locations of the vehicle in the map, from the beginning of the mapping until the current time t [32]. The OG of a single measurement cycle without the map history is $p(m|z_t)$. Under the assumption that the map is static and all ego locations are known, the map can be updated over time with the Bayes filter applied for each individual grid cell m_i .

The probability values of individual cells are computed in a Bayesian framework, thus, are assumed to be stochastically independent and the calculation of the posterior is done for all cells separately. This assumption of course is not valid, as underlying geometries are correlated over multiple cells. Still, the Bayesian framework for OGs is used with great success, as it approximates well. To mitigate this discretization, continuous mapping methods evolved, such as Gaussian process occupancy maps (GPOM) or Hilbert maps. Both approaches treat sensor measurements as a form of sampling a continuous distribution forming the real environment. The idea of GPOM is to place a Gaussian prior over the space of functions mapping locations to the occupancy class [98,99]. The final, continuous Gaussian process classifier model is sampled in the discrete locations of the grid map. As the complexity of the representation is growing with the number of data points, a computationally less expensive, continuous approach was introduced with Hilbert maps. Hilbert maps represent the occupancy property of the world with a linear discriminative model operating on a high dimensional feature vector that projects observations into a reproducing kernel Hilbert space (RKHS) [107].

Occupancy Grid Placement

Depending on the sensor setup and the type of application, these occupancy grids can be designed to model the environment in front of, or surrounding the car. Recall the typical sensor suites on vehicles with low SAE levels in Chapter 1.1.1 and Table 1.1, where SAE L1 and L2 are estimated mostly incorporate front facing radar and camera sensors. In this case, the placement of the occupancy grid is naturally in front of the car (Fig. 2.13a). Few vehicles with partial automation are assembled with an enhanced sensor suite, e.g. Tesla Model S, but most likely all vehicles with SAE Level 3 and above will have surround perception enabled by a full sensor cocoon around the car (multiple radar, camera and ultrasonic sensors). The resulting sensor readings can be mapped into an occupancy grid that spans an area around the car (Fig. 2.13b).

2 Preliminary Signal Processing

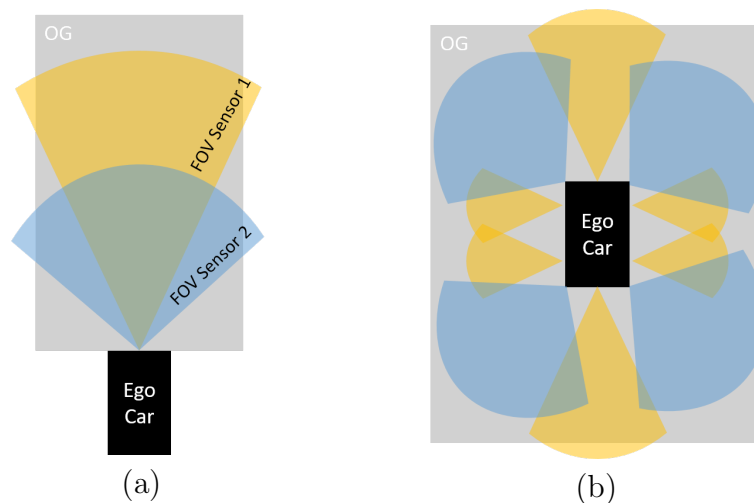


Figure 2.13: Different placements of occupancy grids relative to the ego vehicle, dependent on hardware setup and application. (a) OG in front of, (b) surrounding the car.

Occupancy Grid Generation

The generation of OG starts similarly as the world generation for the object-based environmental model: by receiving measurements z from the range sensors. Depending on the sensor modality, z can come in various forms, as usually lower-level data is employed for the occupancy grid generation than for the object tracking. The earliest works in OG generation with range sensors were done with ultrasonic sensors, where the detected range of each sensor was integrated into a two-dimensional grid with the help of an inverse sensor model (ISM) [32, 33]. Most of the works in literature employ laser range finders for OG generation, as these sensors offer high angular resolution and a large number of detections per measurement cycle [96, 132]. Occupancy grid mapping was also investigated based on radar and camera technology and proven to work well, also for freespace estimation [2, 25, 76, 115].

Typically, the state s_i of a grid cell m_i is a random variable with the value of s_i describing the probability of the cell being occupied $P(m_i|z) = 1$ or free $P(m_i|z) = 0$.

Numerically, an occupancy grid can be interpreted as a special case of a tensor, where the grid is spanned over typically two or three dimensions, depending on if it is a 2D or a 3D occupancy grid. Each cell has certain properties that are described in the additional dimensions of the OG tensor. For example, a simple two-dimensional OG with the occupancy probability as the only property of the cells is a three-dimensional tensor. In this work, all occupancy grids are assumed to be two-dimensional and following Def. 2.2.1.

Definition 2.2.1 (Occupancy Grid) *An occupancy grid is a special case of a tensor (3.3.1), where the first two dimensions describe the rectangular area (x, y) and*

the rest of the dimensions describe the underlying features $f_1 \dots f_n$ of the occupancy grid. $O \in \mathbb{R}^{x \times y \times f_1 \times \dots \times f_n}$

Convention: The feature occupancy probability $p(o_i)$ is defined as a random variable describing the probability of the cell i being occupied.

- $p(o_i) = 0.0 \rightarrow$ the cell i is most likely not occupied.
- $p(o_i) = 0.5 \rightarrow$ the occupancy state of the cell i is unknown.
- $p(o_i) = 1.0 \rightarrow$ the cell i is most likely occupied.

2.2.3 Sensor Modeling

New sensor measurements z typically come in form of target or object lists. These arrays of data need to be translated into a format that can be integrated into the occupancy grid framework with the Bayes filter. Each sensor modality perceives the environment differently, thus, generates a list of information from the surroundings. Therefore, the process of perceiving is described by the PDF of the sensor model $p(z|l)$. Oppositely, reasoning from sensor measurements z to a map $m(z)$ requires a sensor-specific modeling, the inverse sensor model (ISM), and the exact location l . The ISM is applied for transforming data from range sensors to spatial occupancy probabilities, e.g. sonar, lidar or radar data. Each sensor modality inhibits its advantages and drawbacks, such as the radar is precise in estimating the target's range, but is weak in determining the angle. The ISM takes this into account by modeling the transform accordingly (Fig. 2.14).

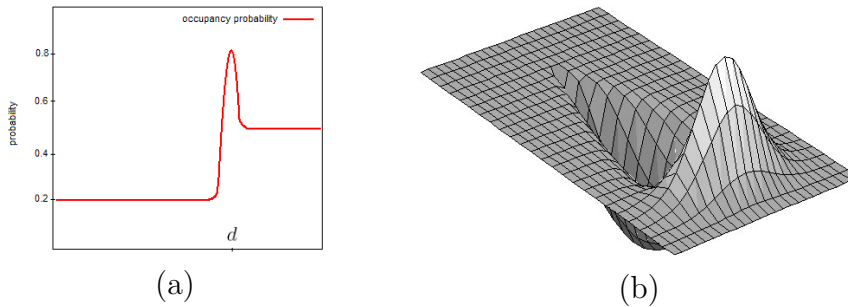


Figure 2.14: (a) One-dimensional ISM of a lidar ray with a target at distance d . The occupancy probability is low in front of the target, high at the location of the target and unknown behind ($p(r) = 50\%$, $r > d$). Image taken from [41]. (b) Two-dimensional ISM of a range sensor with the sensor positioned on the left end of the dip and the target located at the peak. The surface models the probability of occupancy. Image taken from [93].

2 *Preliminary Signal Processing*

Recent approaches use deep learning algorithms to generate an inherently probabilistic ISM from raw radar observations [138] and sonar scans [84], which is further discussed in Chapter 3.2.

2.3 Sensor Fusion for Automotive Environmental Modeling

In this section, methods for combining the data from multiple ($K > 1$) sensors of different pre-processing levels into a uniform representation is discussed. Here, only the fusion of the measurements z_1, \dots, z_K of one timestep is considered. On a high-level, two groups of approaches can be distinguished for fusing sensor information: The ones that integrate the measurements z_1, \dots, z_K by assuming a conditional dependence and those that assume the measurements to be conditionally independent. The first approach integrates the sensor data z_1, \dots, z_K simultaneously into the final representation, which can be the object tracking or a combined occupancy grid, based on the *forward sensor model* described in [128]. In case of the forward sensor model, for a given set measurements, a map is constructed in a way that considers all inter-cell dependencies. Thus, the mapping task becomes an optimization problem that tries to find a map that maximizes the data likelihood [128].

In turn, the approach that relies on the assumption of independence between individual grid cells and measurements, has been adopted in the numerous works regarding mapping tasks [41, 93, 96, 132, 142]. This hypothesis allows to generate a fused representation in separate sub-steps, as depicted in Fig. 2.15. The independence is leveraged by building an individual OG for each of the k sensors independently ($O(z_1), \dots, O(z_K)$). The next step is to integrate the sensor-specific occupancy grids into a fused representation O_{fused} .

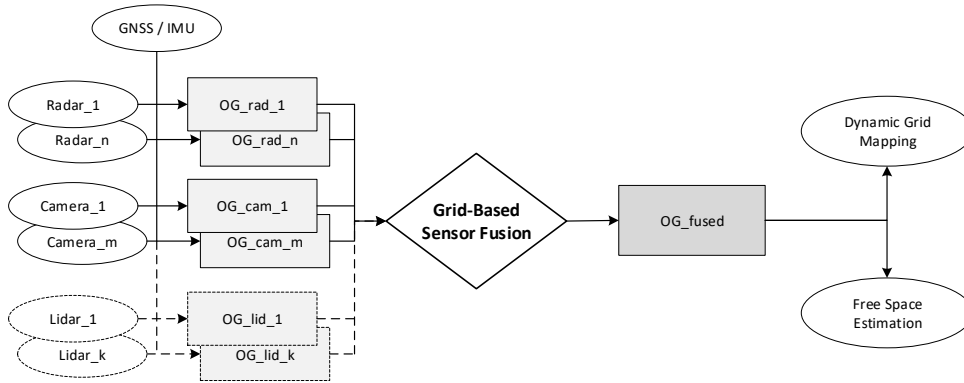


Figure 2.15: Grid-based sensor fusion block diagram.

2.3.1 Levels of Sensor Fusion

Out of the many categories and concepts of sensor fusion algorithms, the most commonly used ones are compiled into six levels with the JDL Data Fusion Model [121].

2 Preliminary Signal Processing

They range from very high level information fusion on the user level, down to fine-grained data alignments. Three of those are best applicable to the sensor fusion task in automotive:

- *Decision-level or late fusion* represents the highest level of abstraction and uses sensor information that has already been extensively processed and possibly used for classification. The decision-level fusion combines high-level information from various, possibly conflicting nodes and tries to find the hypothesis of the sensors that fits best to the situation. An example for decision-level fusion is the estimation of the traffic state (low traffic, medium flow, traffic jam) based on data from acoustic and image sensors [66].
- *Feature-level fusion* takes data from the sensor nodes that have already been processed. As a consequence of the pre-processing, the information contained in the signals is smaller, lightens the bandwidth and computational requirements of the system. The selection of significant and descriptive features for each sensor modality is fundamental for the design of the feature-level fusion system. For simple automotive assistance systems, such as ACC, this level of fusion is performed in an object-based environment, typically with Kalman filters. Details to the feature-level sensor fusion are described in Section 2.1.2 by the use of radar track data.
- *Data-level or early fusion* is a fusion technique at the lowest level of abstraction. Usually, it aims to combine raw data from multiple sources of the same modality, as the data format is the same. It has to most capabilities and information in the result, but comes to the cost of the largest bandwidth and computational requirements. Typically, this kind of fusion is needed for information-rich environmental models for high automation levels, as outlined in Section 1.1.1.

In the main part of this work, feature-level data is the basis for the fusion process. An information-rich representation similar to the raw, data-level fusion is investigated to be achieved by enhancing the feature-level fusion with machine learning techniques.

2.3.2 Grid-Based Bayesian Fusion

Whereas in reality, objects often span over the area of multiple grid cells, thus, are stochastically dependent of each other, they are treated like if they were independent in order to apply the Bayesian fusion. This was first done by Moravec in 1988 [92]. It is deducted from the Bayes' theorem:

Theorem 2.3.1 (Bayes)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \quad (2.28)$$

where A and B are events and $P(B) \neq 0$.

Bayesian Fusion Principle

Given a system of two sensors with their measurements z_1, z_2 , the occupancy grid $O(z_1, z_2)$ is generated by calculating the cell-wise occupancy probability $P(o_i|z_1 \wedge z_2)$. Moravec replaced the occupancy probability with the odds ratio of the two events (o_i : cell i occupied, e_i : cell i empty):

$$\frac{P(o_i|z_1 \wedge z_2)}{1 - P(o_i|z_1 \wedge z_2)} = \frac{P(o_i|z_1 \wedge z_2)}{P(e_i|z_1 \wedge z_2)}, \quad (2.29)$$

together with Bayes' theorem and for each occupancy state separately,

$$P(o_i|z_1 \wedge z_2) = \frac{p(z_2|o_i \wedge z_1)P(o_i|z_1)}{P(z_2|z_1)}, \quad (2.30)$$

$$P(e_i|z_1 \wedge z_2) = \frac{p(z_2|e_i \wedge z_1)P(e_i|z_1)}{P(z_2|z_1)}, \quad (2.31)$$

becomes

$$\frac{P(o_i|z_1 \wedge z_2)}{1 - P(o_i|z_1 \wedge z_2)} = \frac{P(o_i|z_1)p(z_2|o_i \wedge z_1)}{P(e_i|z_1)p(z_2|e_i \wedge z_1)}. \quad (2.32)$$

Following the deduction from Rakotovao [105], the terms $p(z_2|o_i)$ and $p(z_2|e_i)$ can be derived from the Bayes' theorem:

$$p(z_2|o_i) = \frac{P(o_i|z_2)p(z_2)}{P(o_i)}, \quad (2.33)$$

$$p(z_2|e_i) = \frac{P(e_i|z_2)p(z_2)}{P(e_i)}, \quad (2.34)$$

and inserted into Eq. 2.32 (Note: $P(e_i|z_1) = 1 - P(o_i|z_1)$ and $P(e_i) = 1 - P(o_i)$) results in

$$\frac{P(o_i|z_1 \wedge z_2)}{1 - P(o_i|z_1 \wedge z_2)} = \frac{P(o_i|z_1)}{1 - P(o_i|z_1)} \cdot \frac{P(o_i|z_2)}{1 - P(o_i|z_2)} \cdot \frac{1 - P(o_i)}{P(o_i)}. \quad (2.35)$$

Solving to the odds of this cell being occupied considering the two measurements z_1 and z_2 , the final equation is

$$P(o_i|z_1 \wedge z_2) = \frac{P(o_i|z_1) \cdot P(o_i|z_2) \cdot [P(o_i) - 1]}{P(o_i) \cdot [P(o_i|z_1) + P(o_i|z_2) - 1] - P(o_i|z_1) \cdot P(o_i|z_2)}. \quad (2.36)$$

2 Preliminary Signal Processing

Eq. 2.36 describes the Bayesian fusion of two sensor readings in one cell of an occupancy grid. The hypothesis of independent measurements allows to compute $P(o_i|z_1 \wedge z_2)$ from the ISMs $P(o_i|z_1)$ and $P(o_i|z_2)$. The complete occupancy grid can be generated by computing Eq. 2.36 for each cell in the grid.

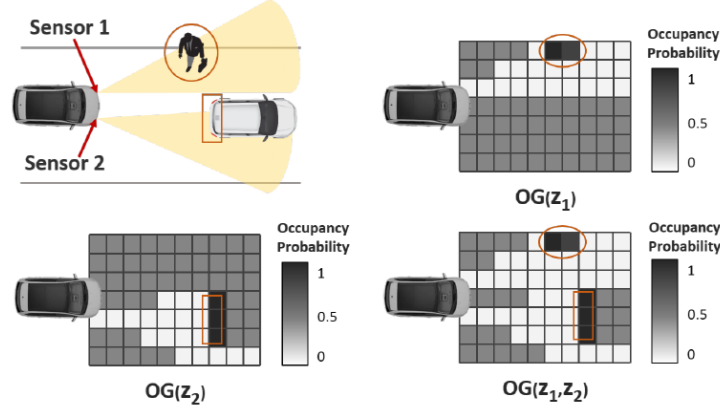


Figure 2.16: Exemplary scenario for grid-based sensor fusion of two sensor-specific occupancy grids $OG(Z_1)$, $OG(Z_2)$ and the resulting fused representation $OG(Z_1, Z_2)$. Image taken from [105].

With an Independent Opinion Pool to the Log-Odds

Assume that non-informative priors ($P(o_i) = 0.5$, $P(e_i) = 0.5$) are known in Eq. 2.36. Under this hypothesis, the Bayesian fusion can be formulated as a form of the Independent Opinion Pool [16, 105] by bringing it to the form of

$$F(x, y) = \frac{xy}{xy + (1-x)(1-y)}. \quad (2.37)$$

In other words:

$$P(o_i|z_1 \wedge z_2) = F(P(o_i|z_1), P(o_i|z_2)), \quad \text{if } P(o_i) = P(e_i) = 0.5 \quad (2.38)$$

The benefit of formulating the problem as a Independent Opinion Pool task is twofold. Contradicting sensor readings are treated in a way that the uncertainty about the occupancy state of the cell is increasing and the occupancy probability is moving towards 0.5. On the other hand, matching sensor measurements reinforce their values and output the fused occupancy state with a higher certainty [32].

However, the property of reinforcing the occupancy probabilities leads to numerical problems, as once the values reach either 0 or 1, the equation becomes agnostic to any new sensor readings. Elfes proposed a solution in form of the log-odds

$$l(o_i|z_1 \wedge z_2) = l(o_i|z_1) + l(o_i|z_2), \quad (2.39)$$

2.3 Sensor Fusion for Automotive Environmental Modeling

where $l(x) = \log\left(\frac{P(x)}{1-P(x)}\right)$ [32]. The logarithmic form solves the numerical instability by relying on simple additions instead of multiplications and divisions in Eq. 2.38.

$$P(o_i|z_1 \wedge z_2) = 1 - \frac{1}{1 + \exp(l(o_i|z_1 \wedge z_2))} \quad (2.40)$$

translates the log-odds values back to occupancy probabilities.

Dempster-Shafer Fusion

Recall Definition 2.2.1, where the main feature of each cell of the occupancy grid is the occupancy probability. Boiling down the sensor measurements of a series of sensors into one single feature may lead to results that seem to be certain, but underlie conflicting sensor measurements. As an example, two sensors generate contradicting data ($P(o_i|z_1) = 0$, $P(o_i|z_2) = 1$), each with high certainty. However, the result of the Bayesian fusion (Eq. 2.40) is 0.5, which is equal to a situation with no sensor measurements at all. To cope with this loss of information, the Dempster-Shafer theory of evidence explicitly models the uncertainty coming from contradicting sensor signals [29].

According to the Dempster-Shafer fusion (DSF), a probability mass is assigned for all combinations of possible states. Given the possible states in occupancy grids $\Theta = \{o, e\}$, the combinations are $2^\Theta = \{\emptyset, o, e, \{o, e\}\}$. In the DSF, the following probability masses arise as features of the occupancy grids:

- $p(\emptyset) = 0 \rightarrow$ the empty set is always impossible.
- $p(o_i) \rightarrow$ the probability of cell i being occupied.
- $p(e_i) \rightarrow$ the probability of cell i being empty.
- $p(\{o_i e_i\}) \rightarrow$ cell i has conflicting sensor measurements.

The application of the DSF proves to be useful in many works [89, 142]. Others argue that it is not needed, because automotive sensors have to be intrinsically reliable and contradicting sensor measurements only come from wrong sensor configurations or bad calibrations [105].

2.3.3 Data Synchronization

In real-world scenarios with a realistic sensor setup, additional important aspects have to be considered before performing the sensor fusion.

Spatial Alignment

Due to the highly diverse sensor placements and orientations, a spatial alignment is required for a uniform representation (see Section 2.2.2). A thorough calibration is needed for determining the exact areas of the fields of views (FOV) of the sensors. The overlapping parts of the FOV can be used for instantaneous¹ sensor fusion. When inserting a sensor measurement into a partially built map, the FOV of the sensor has to be placed into the correct location of the given map. For that step, the vehicle's position and orientation within the map has to be known exactly, typically by self-localization in parallel to the mapping task (simultaneous localization and mapping SLAM). SLAM uses a combination of IMU/GNSS data smoothed by a Kalman filter and position estimations based on the range sensor data matched to the known map. This way, the map integrity is ensured and the spatial alignment done.

Temporal Alignment

Further, the temporal synchronization of the sensor readings is crucial for a robust high-quality system. The measurement latency t_L between the actual acquisition time and the sensor output time is not negligible, especially, when the pre-processing is time consuming, as in the case of visual detection systems. Further, in multi-modal systems with sensors s_i with $i > 1$, the measurement frequencies vary from sensor to sensor. Thus, the sensors start the acquiring in different measurement cycle times t_C . As a natural consequence, the signals arrive asynchronously to the fusion ECU.

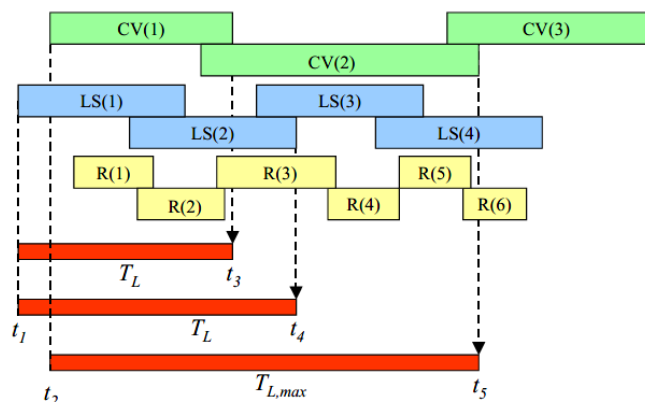


Figure 2.17: Temporal alignment problem for multi-modal sensor systems. Image taken from [68].

If the latency $t_L^{s_i}$ is constant for each sensor, a deterministic schedule can be derived for integrating the sensor readings at different timesteps. On the other hand, if the

¹Within one timestep.

2.3 Sensor Fusion for Automotive Environmental Modeling

latency $t_L^{s_i}$ is varying for at least one sensor, such a schedule can be disturbed; it is non-deterministic [68]. The temporal alignment is done based on timestamps $t_{TS}^{s_i}$ created at the beginning of each data acquisition, and the order of which sensor measurement to integrate next has to be decided in each timestep. This is to the fact that the Kalman filter can only predict future states of a system, not the past states. If a signal from $t_{TS} = 0$ arrives later than a signal from $t_{TS} = 2$, it can not be integrated. To cope with the non-deterministic way of the input data, the fusion latency $t_{L,\max}$ for integrating the signals is set to the maximal worst-case latency:

$$t_{L,\max} = \max(t_L^{s_i}) + \max(t_C^{s_i}) \quad (2.41)$$

In the remainder of this work, it is assumed that the data from the sensors arrives synchronously to the fusion system.

3 Neural Networks for Sensor Fusion

In this chapter the problem of grid fusion is tackled with the application of neural networks. First, the choice of neural network type is explained for the given hardware setup of Level 2/2+ cars. Recall Chapter 1, where the constraints and challenges for the fusion ECU are introduced. Namely, the typical hardware setup for partially automated cars and the limited, or even no access to raw sensor data. As the computations of complex neural networks pose a large problem to resource-constrained automotive embedded hardware, the computational costs of the investigated models are analyzed. Finally, a suite of state-of-the-art hardware accelerators for neural computations are introduced, on which the proposed models will be evaluated.

3.1 Introduction and Motivation for the Usage of Convolutional Neural Networks and Auto-Encoders

With the perception system built upon the requirements and capabilities of Level 2/2+ cars, only feature-level sensor data are available for the fusion ECU. In order to perform a centralized fusion with the help of machine learning, the available data has to be prepared for being fed into the machine learning models. One promising method of preparing the available data for neural network processing is to transform the data lists to occupancy grids. As occupancy grids are two-dimensional matrices with a third dimension describing the features of each pixel, they can naturally be interpreted similarly as image data. Once the inputs to the model are image-like, convolutional neural networks are an ideal option for processing, as they are well-known and studied widely throughout computer vision literature.

Still, there are several differences between image processing and occupancy grid processing. In the classical computer vision task of image classification, the neural network receives one input image, performs several manipulations throughout the network in order to calculate the output, a one-dimensional classification vector. The first two dimensions of the input are the spatial sizes of the image in pixels. The third dimension are the features of the image. In case of a black/white image, the only feature is denoting the grayscale of this pixel. RGB images have three values for each pixel stored, describing the red, green and blue color values of this pixel. Recall, that occupancy grids describe occupancy states of spatial locations in

a grid. Each grid cell is similar to the image pixels, an area with underlying features. In contrast to images, grid cells can have more than one feature about the specific area. The features can be, but are not limited to, the probability of existence of an obstacle, its underlying object's class, or information about its motion model, such as its velocity vector.

Nevertheless, the basic nature of images and occupancy grids is the same, allowing to process both with CNNs robustly and efficiently with a large variety of processing schemes that even allow manipulation of the OGs. For example, convolutional auto-encoders are used for the semantic segmentation of images. In this case, the input to the network is an image and the output is usually an image of the same dimensions, but the content of the image is manipulated, or new features are added to the initial image. This property of convolutional auto-encoders can be employed for the processing of occupancy grids, too.

In the following, current approaches to perform various usecases of sensor fusion with machine learning techniques are explored. The related work paves the way towards the application of CNNs for grid-based sensor fusion.

3.2 State of the Art Neural Networks for Occupancy Grid Processing and Sensor Fusion

The applications of neural networks for the different stages of occupancy grid processing were investigated in literature. Three main stages can be distinguished: The generation, the manipulation and the fusion of occupancy grids.

Generation of OGs

Not only is the processing of occupancy grids enhanced by neural networks—the map generation can be leveraged, too. Recall, that the sensor data in form of object and target lists may first be translated to spatial occupancy probabilities. This translation is done with the sensor-specific inverse sensor models, which vary not only from sensor type to sensor type, but also from individual sensors of a same type (eg. two ultrasonic sensors of the same type). Because the inverse sensor model is different for each sensor, an adaptive inverse sensor model comes at hand. The learning of inverse sensor models of range sensors was investigated for sonar sensors in [47, 84].

Radar signals can be processed by neural networks in various stages of the typical signal processing chain (Chapter 2.1.1). Semantic segmentation is applied to radar range-angle maps in order to receive the distribution (μ, Σ) of occupancy [138]. A convolutional auto-encoder is employed with a transformer from polar to Cartesian coordinates, which is placed after the encoder. Both, mean μ and standard deviation Σ are the outputs of the network. This distribution is used to reconstruct the spatial

3.3 Neural Networks for Fusion of Camera and Radar Data

occupancy probabilities describing an occupancy grid.

Manipulation of OGs

As stated in the introduction before, the idea of processing occupancy grids with convolutional neural networks known from computer vision tasks stands to reason, and thus, it has been worked on in the recent years.

Deep convolutional neural networks are employed to detect objects in occupancy grids based on 3D point clouds coming from lidar sensors [140]. Further, a fully convolutional architecture is used to enhance the quality of occupancy grids that were computed with the standard Dempster-Shafer fusion [89]. Typically, networks that output occupancy grids themselves, rely on the basic architecture of the U-Net [113].

Fusion of OGs

In the field of occupancy grid fusion with neural networks, no literature is known to the author. To complete this lack of research, the following section is dedicated to deduct neural networks for the fusion of occupancy grids in general—and specifically for the sensor setup of Level 2/2+ cars defined in Subsection 1.1.1.

3.3 Neural Networks for Fusion of Camera and Radar Data

The generation and manipulation of occupancy grids with neural networks are two stages of occupancy grid processing, which have been worked on. Yet, the grid-based fusion with neural networks has not been investigated and, thus, forms an open topic for research. The contents of this research and analysis are described in this section and published in [9, 10] as conference papers^{1 2}.

The grid fusion networks have to fulfill the following aspects:

- Fusion of multiple two-dimensional occupancy grids into one combined environmental model.
- Fusion of one timestep at a time, thus, no temporal fusion or mapping over time.
- The fusion has to rely only on data originating from sensors used in Level 2/2+ cars.

¹Paper [10]: IEEE ICCVE 2019 (Best Student Paper Award).

²Paper [9]: ewC 2020.

3 Neural Networks for Sensor Fusion

- The output occupancy grid map incorporates the structural components of the surroundings and the moving objects, based on the fused sensor information.
- Create an instantaneous free space estimation within the fused occupancy grid for further trajectory planning tasks.

System Overview

The context of grid fusion networks in the general "sense-plan-act" framework of ADAS and AD is the "sense"-block. After the sensors sense the environment and compute their individual features and object detections, the data is transmitted to a central fusion ECU. Within this ECU, the target and object lists are mapped to sensor-specific occupancy grids, which form the inputs to the grid fusion networks. The output of the fusion module is the fused representation of all inputs and shows a comprehensive environmental model of the car's surroundings.

This model is an input to the "plan"-block and is used as the basis for further algorithms. The main use of the fused, uniform representation is the path planning of the cars future trajectories. A correct placement of the obstacles and objects is crucial for collision avoidance, and the robust free space estimate is important for the trajectory planning.

Structure of this Chapter

In the following sections, viable architectures of grid fusion networks are deduced. Special requirements towards the training data are highlighted and put into perspective of current publicly available datasets. With the models and the correct data at hand, the training process is defined. Finally, the trained models are analyzed in terms of fusion quality, hardware requirements and the applicability on state-of-the-art hardware accelerators.

3.3.1 Structure, Building Blocks and Working Principle of Sensor Fusion Networks

In the system overview, the position of the sensor fusion networks is defined in between the various sensors and the path planning, thus, the type of inputs and outputs are determined. An array of sensors is assumed and each with its sensor data \mathcal{D}_i , which can be a point cloud, a target list, an object list or of similar type. In the pre-processor stage, the various types of sensor readings are transformed to the uniform representation of occupancy grids, but incorporating possibly different areas of observation and different kinds of features.

In the following, it is assumed that the occupancy grids are transformed in a way, so that all describe the same area of coverage. If there is no sensor measurement about an area, it is treated with the same uncertainty as if it is outside the field of

3.3 Neural Networks for Fusion of Camera and Radar Data

view of the specific sensor. Areas outside the field of view of a sensor are padded with the same values as areas with no measurements.

Definition 3.3.1 (Tensor) *An n -dimensional tensor is defined as $\mathbf{T} \in \mathbb{R}^{d_1 \times d_2 \times \dots \times d_n}$ and is denoted with an upper-case, bold letter.*

An occupancy grid (Def. 2.2.1) is a special case of a two-dimensional matrix, incorporating one or more feature layers, which is—in general—a tensor. In the course of this work, the notation of a tensor will be used for occupancy grids, when neural processing is involved, as the dimensions are changing often throughout the networks.

Macro-Architectural Building Blocks

As most of the state-of-the-art neural networks, the grid fusion networks are constructed out of configurable blocks that can appear repeatedly in the network [50]. The repeatability allows the generation of complex, deep neural networks without losing track of both, decisions on the low-level, and design choices on the high-level architecture. How these blocks are arranged with regards to each other is defined by the macro-architecture of the model. In the macro-architecture, the number and type of blocks and their interconnections are configured, which in consequence determines the type of neural network in total. Colloquially speaking, the macro-architecture forms the skeleton or spine of the network, and the operations within the blocks are the flesh of the model. The blocks can either change the passed tensors by enlarging or reducing the spatial dimensions or leave them unchanged.

Definition 3.3.2 (Operation) *An operation \mathbf{o}_c takes a tensor $\mathbf{T}_0 \in \mathbb{R}^{d_0}$ as an input, performs a calculation c and transforms the input to the output tensor $\mathbf{T} \in \mathbb{R}^d$ according to $\mathbf{T} = \mathbf{o}_c(\mathbf{T}_0)$.*

- $\mathbf{o}_{conv}^{f\ s}$: Standard two-dimensional convolution. The filter size s and the amount of filters f are denoted in the superscript, e.g. $\mathbf{o}_{conv}^{8\ (3 \times 3)}$.
- $\mathbf{o}_{deconv}^{f\ s}$: Transposed convolution. Strides of 2, or equivalently doubling of image dimensions by default. The filter size is denoted in the superscript, e.g. $\mathbf{o}_{deconv}^{16\ (5 \times 5)}$.
- \mathbf{o}_{ups} : Spatial upsampling. Bilinear interpolation to the double image dimensions by default.
- \mathbf{o}_{relu} : ReLU activation function
- \mathbf{o}_{bn} : Batch normalization
- \mathbf{o}_{max} : 2D spatial max pooling

3 Neural Networks for Sensor Fusion

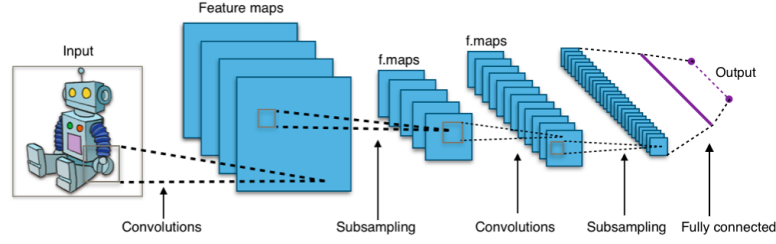


Figure 3.1: Typical CNN architecture for image classification tasks. Source: https://commons.wikimedia.org/wiki/File:Typical_cnn.png accessed on 10. Feb. 2021.

Blocks that reduce the size of a tensor are denoted as *reduction blocks*, and they usually inhibit downsampling operations, such as convolutions with strides or max pooling layers. Similarly, blocks that expand the tensor’s spatial dimensions are denoted as *expansion blocks*, and they come with upsampling operations, such as transposed convolutions¹, or bilinear interpolations. *Normal blocks* apply operations to tensors that remain unchanged in dimensions.

A typical, straight-forward classification network (Fig. 3.1) is built of several normal and reduction blocks. Once a desired, low-dimensional feature space is retrieved, a fully connected layer forms the tensor to a classification output.

Definition 3.3.3 (Blocks) A generic block $\mathcal{B} = (\mathcal{O}, \mathcal{G})$ is a confined set of n operations $\mathcal{O} = \{\mathbf{o}_1, \dots, \mathbf{o}_n\}$ that are arranged in a directed acyclic graph \mathcal{G} and transform the input $\mathbf{T}_0 \in \mathbb{R}^{d_0}$ to the output $\mathbf{T} \in \mathbb{R}^d$ by applying it to the input $\mathbf{T} = \mathcal{B}(\mathbf{T}_0)$.

The directed acyclic graph $\mathcal{G} = (\mathcal{E}, \mathcal{V})$ is based on edges \mathcal{E} (denoting Operations, Def. 3.3.2) and vertices \mathcal{V} (denoting Nodes, Def. 3.3.4). It is represented by its adjacency matrix \mathbf{A} of size $\mathcal{V} \times \mathcal{V}$. Each entry of \mathbf{A} describes an generating $\mathbf{o}^{a \rightarrow b}$ using tensor a and producing tensor b by a combination operation, such as addition.

$$\mathbf{A} = \begin{bmatrix} \mathbf{o}^{1 \rightarrow 1} & \mathbf{o}^{1 \rightarrow 2} & \mathbf{o}^{1 \rightarrow 3} \\ \mathbf{o}^{2 \rightarrow 1} & \mathbf{o}^{2 \rightarrow 2} & \mathbf{o}^{2 \rightarrow 3} \\ \mathbf{o}^{3 \rightarrow 1} & \mathbf{o}^{3 \rightarrow 2} & \mathbf{o}^{3 \rightarrow 3} \end{bmatrix} \quad (3.1)$$

As it is an acyclic graph, only the elements of the upper right triangular matrix are relevant ($\mathbf{o}^{a \rightarrow b} \forall a, b \in [1, \mathcal{V}] \wedge a < b$). Thus, for $\mathcal{V} = 3$, only 3 operations are possible within the block, as seen in its adjacency matrix

$$\mathbf{A} = \begin{bmatrix} & \mathbf{o}^{1 \rightarrow 2} & \mathbf{o}^{1 \rightarrow 3} \\ & & \mathbf{o}^{2 \rightarrow 3} \\ & & \end{bmatrix}. \quad (3.2)$$

¹Notes on nomenclature of this operation: Deconvolution, inverse convolution.

Micro-Architecture of Blocks

On the low-level inside of a block, the micro-architecture defines the contents of each building block, such as the type of operations, the number of layers and how they are connected to each other. The arrangement of layers within a block can be as simple, as the first two entities of *AlexNet*, which can be interpreted as reduction blocks containing four directly connected operations (convolution, ReLU activation, local response normalization, max pooling) [75]. Recent advances in automated optimization approaches, such as neural architecture search, show a tendency towards more complex micro-architectures [79,80,88,109,137,147]. Those blocks are usually defined by a directed acyclic graph, where the edges are the operations and the nodes are combinations of two or more tensors to one. The application of the neural architecture search on the grid fusion problem is discussed in Chapter 4.

The Fusion Step

In the definitions and discussions above, the building blocks for single-stream neural networks have been introduced. A single-stream architecture processes a single data source, such as images for classification, object detection or segmentation. A multi-stream model can combine multiple sources of information, such as single frames of a video and in parallel the optical flow for action recognition in videos [117]. If there are multiple streams of data involved, their fusion can be performed at different stages of the network, similarly to the position of the traditional fusion methods (*early*, *late* and *hybrid* fusion, Chapter 2.3).

A late fusion in classification networks is usually done in the fully connected layers. Either each stream has its own classification and the final decision of the network is done in a majority vote of all streams' weighted results, or each stream's last embeddings are concatenated before feeding them into the final classification layer. The late fusion concept does not only apply to neural networks. A prominent example for late fusion in the larger machine learning methodology is random forests that combine an arbitrary number of decision trees. In general, late fusion can be considered as an ensemble of multiple models that tends to generalize better than each single model. Ensembles perform best, when the components have a low bias and high variance, which can be ensured by choosing models with diverse structures and formats.

An early fusion of multiple input streams with convolutional neural networks can be done by combining the input tensors at the beginning of the architecture. Along various possibilities of early combinations for example of same-sized images, such as element-wise addition or extension along a spatial dimension, the concatenation of the features is most viable.

Hybrid fusion is, when the multiple streams are not fused at one specific point of the model, but when the fusion is performed over multiple stages. FuseNet combines RGB and depth images with a two-stream auto-encoder, where both modali-

3 Neural Networks for Sensor Fusion

ties are fused by successively adding the encoded features in the different encoding depths [49]. An other approach is to fuse during the decoding stage, as shown in ChiNet [65].

Either way of fusing input occupancy grids, embeddings or, in general, tensors, the combination is defined by its multiple inputs and the combination operation. Similarly, and in addition to the Operations (Def. 3.3.2), the nodes of this work are defined as follows.

Definition 3.3.4 (Node) *A node \mathcal{N} is an n -tuple that combines n tensors of same spatial dimensions $\mathbf{T}_1 \in \mathbb{R}^{x \times y \times d_1}$, $\mathbf{T}_2 \in \mathbb{R}^{x \times y \times d_2}$, \dots $\mathbf{T}_n \in \mathbb{R}^{x \times y \times d_n}$ to one tensor with new feature dimensions $\mathbf{T} \in \mathbb{R}^{x \times y \times d}$ according to the notation*

$$\mathbf{T} = \mathcal{N}(\mathbf{T}_1, \mathbf{T}_2, \dots \mathbf{T}_n). \quad (3.3)$$

The tensor fusion is done with a combination operation, such as

- \mathcal{N}_{add} : *Element-wise addition.*
- \mathcal{N}_{cat} : *Concatenation of features.*

Fully Convolutional Fusion Networks (FC-FN)

A straight-forward way to fuse the input occupancy grids is to concatenate the inputs as a first step and to stack simple convolutional blocks for processing the information. The number of convolutional blocks is defined by the depth δ . In the last block \mathcal{B}^δ , the output is formed by reducing the number of feature channels of the last embedding tensor $\mathbf{T}^{\delta-1}$ to an occupancy grid-like format $\mathbf{T}^\delta \in \mathbb{R}^{x \times y \times f_n}$ with f_n features.

The image dimensions of the tensors stay unchanged, as no upsampling, nor strides are used, and all convolutional layers use padded inputs in order to keep a fixed occupancy grid size. The receptive field of each convolutional filter is limited due to the constant tensor dimensions throughout this simple network design. Thus, the expected result of the FC-FN is the fusion of the inputs, but with a limited or even no viable free space estimation, as the filters are unable to recognize larger portions of the surroundings within the occupancy grids.

The architectural design parameters of FC-FN are the number of input occupancy grids, their grid dimensions, the number of convolutional blocks δ and their micro-architecture. In turn, the micro-architecture is defined by the type, amount and connection of single operations and the configuration of e.g. the convolutions. In order to keep the design of FC-FN simple, the normal block \mathcal{B}_{norm}^{FC} consists of a two-dimensional convolution \mathbf{o}_{conv} , a ReLU activation \mathbf{o}_{relu} and batch normalization \mathbf{o}_{bn} forming the set of operations $\mathcal{O}_{norm}^{FC} = \{\mathbf{o}_{conv}, \mathbf{o}_{relu}, \mathbf{o}_{bn}\}$. The block is interconnected

3.3 Neural Networks for Fusion of Camera and Radar Data

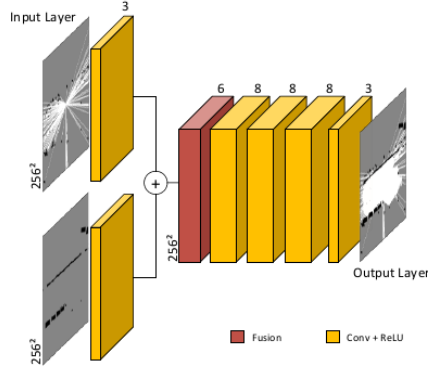


Figure 3.2: The architecture of fully convolutional fusion networks with $\delta = 4$ and $f = 8$. Only two input occupancy grids are depicted for better visualization.

without nodes, thus the adjacency matrix of the FC-FN normal block is

$$\mathbf{A}_{norm}^{FC} = \begin{bmatrix} \mathbf{O}_{conv} & & \\ & \mathbf{O}_{relu} & \\ & & \mathbf{O}_{bn} \end{bmatrix}. \quad (3.4)$$

The output tensor of the network is formed by applying the normal block to the input tensor δ times. The input tensor \mathbf{T}_{in}^{FC} is the concatenation of the raw input occupancy grids

$$\mathbf{T}_{in}^{FC} = \mathcal{N}_{cat}(\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_n). \quad (3.5)$$

Using this input, the output tensor is calculated by

$$\mathbf{T}_{out}^{FC} = \mathcal{B}_{norm}^{FC \langle \delta \rangle}(\mathbf{T}_{in}^{FC}) \quad (3.6)$$

and retains the same spatial dimensions. The configuration of the network boils down to two main architectural parameters, the number of convolutional filters f and the depth δ . Both parameters have a great influence on the fusion quality and computational complexity, as the analysis shows (Chapter 3.4.2).

Auto-Encoder Fusion Networks (AE-FN)

The above introduced FC-FN have a low complexity and a low model size, but they are also limited in terms of fusion quality and the ability of estimating free space. The limitation mainly comes from the constrained receptive field of the convolutional filters. To cope with this problem, the auto-encoder offers itself as a viable solution.

The auto-encoder fusion networks are structured into an encoding and a decoding part. In the encoder, each input occupancy grid is encoded separately by multiple

3 Neural Networks for Sensor Fusion

encoding blocks \mathcal{B}_{enc}^{AE} . Each consecutive encoding block is adding to the depth δ of the network.

Once the desired depth is reached, the tensors of the input streams are combined in a fusion node. A concatenation of the n encoded input tensors is forming the tensor in the deepest part of the network,

$$\mathbf{T}^\delta = \mathcal{N}_{cat}(\mathbf{T}_{in,1}^\delta, \mathbf{T}_{in,2}^\delta, \dots, \mathbf{T}_{in,n}^\delta). \quad (3.7)$$

Now, only a single stream is present in the decoder. The same number of decoding blocks \mathcal{B}_{dec}^{AE} is used to reconstruct the output tensor to the same dimensions as the input occupancy grids.

$$\mathbf{T}_{out}^{AE} = \mathcal{B}_{dec}^{AE \langle \delta \rangle}(\mathbf{T}^\delta) \quad (3.8)$$

With the input occupancy grids being the input tensors $x_{OG}^i = \mathbf{T}_{in,i}$, the complete formulation for a forward pass through the AE-FN can be described as

$$\mathbf{T}_{out}^{AE} = \mathcal{B}_{dec}^{AE \langle \delta \rangle}(\mathcal{N}_{cat}(\mathcal{B}_{enc}^{AE \langle \delta \rangle}(\mathbf{T}_{in,1}^\delta), \mathcal{B}_{enc}^{AE \langle \delta \rangle}(\mathbf{T}_{in,2}^\delta), \dots, \mathcal{B}_{enc}^{AE \langle \delta \rangle}(\mathbf{T}_{in,n}^\delta))) \quad (3.9)$$

or in a short notation for multiple input tensors to the concatenation node

$$\mathbf{T}_{out}^{AE} = \mathcal{B}_{dec}^{AE \langle \delta \rangle}(\mathcal{N}_{cat}^i(\mathcal{B}_{enc}^{AE \langle \delta \rangle}(\mathbf{T}_{in,i}^\delta))). \quad (3.10)$$

The equations 3.7, 3.8, 3.9 and 3.10 describe the macro-architecture of the AE-FN that is depicted in Fig. 3.3. In the following, the micro-architecture of the encoding and decoding blocks is discussed.

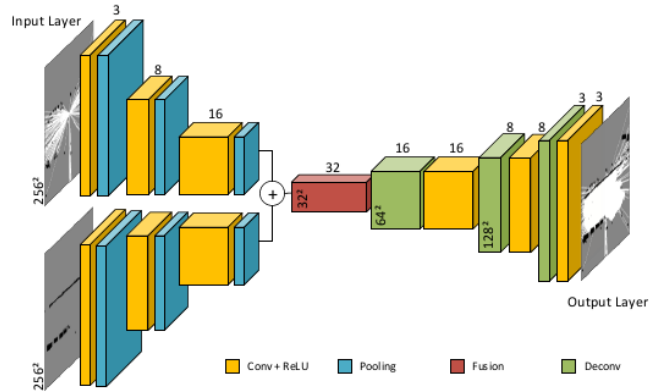


Figure 3.3: The architecture of an auto-encoder fusion network with $\delta = 2$ and $f = 8$. Only two input streams are shown for visualization purpose.

In contrast to the normal block of the FC-FN \mathcal{B}_{norm}^{FC} , the encoding block of the AE-FN $\mathcal{B}_{enc}^{AE-FN}$ reduces the spatial dimensions of the input tensors with reduction

3.3 Neural Networks for Fusion of Camera and Radar Data

operations, such as max pooling \mathbf{o}_{max} or convolutions \mathbf{o}_{conv} with strides. In research, the choice of reduction operation has changed over the last years. Early networks, such as ALL-CNN in 2015 [119] employ strided convolutions in order to have a simplified set of operations—ALL-CNN uses convolutions only. On the other hand, recent networks make use of pooling operations again in order to ease gradient propagation through very deep neural networks and to reduce the overall computations and parameters. In 2018, FishNet [123] replaces 1x1 convolutions in skip connections with pooling in order to fix the gradient propagation issue of ResNet [50]. Because no differences in quality are observed between \mathbf{o}_{max} and \mathbf{o}_{conv} with strides 2 according to [119] and a more stable gradient flow through max pooling layers, max pooling is chosen to reduce the spatial dimensions in reduction cells. The encoding block is now based on the adjacency matrix

$$\mathbf{A}_{enc}^{AE} = \begin{bmatrix} \mathbf{o}_{conv} & & & \\ & \mathbf{o}_{relu} & & \\ & & \mathbf{o}_{max}^{2 \times 2} & \\ & & & \mathbf{o}_{bn} \end{bmatrix}. \quad (3.11)$$

Whereas there are now four operations within the block, the interconnections still remain a simple sequence. The architectural parameters of the AE-FN are again the network depth δ and the configuration of the filters in the convolution operation.

The decoding block is based on the adjacency matrix

$$\mathbf{A}_{dec}^{AE} = \begin{bmatrix} \mathbf{o}_{deconv} & & & \\ & \mathbf{o}_{conv} & & \\ & & \mathbf{o}_{relu} & \\ & & & \mathbf{o}_{bn} \end{bmatrix}. \quad (3.12)$$

The data samples provided by the sensors can be interpreted as different features of a common distribution, namely, reality. The auto-encoder networks learn this underlying distribution of occupancy topology from the data and map an according free space estimate to the grid.

Results show that AE-FN are already capable of calculating a fused environmental representation based on the input streams. Additionally, the free space estimation that is built on the training on lidar-based ground truth data (Chapter 3.3.2) shows large portions of correctly predicted drivable road surface. On the downside, small details of either input branch are lost in the deep parts of the auto-encoder, and the resulting occupancy grid appears blurred. This effect becomes severe for large values of δ .

Auto-Encoder with Bypass Fusion Networks (AEB-FN)

In order to mitigate the blurring effect on the output image, small details need to be passed through the network without compression by the pooling layers. The

3 Neural Networks for Sensor Fusion

method applied to do so is skipping connections, or bypassing parts of the auto-encoder network, and was first introduced in ResNet [50] (Figure 3.4). Others refer to shortcuts to jump over larger portions of the network, demonstrated in HighwayNetworks [120]. By adding more bypassing connections, the densely connected network performance is robustly enhanced [58].

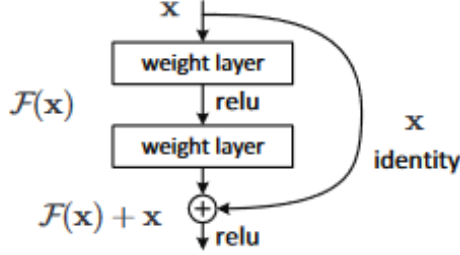


Figure 3.4: A skipped connection proposed in ResNet [50], where a neural computation $\mathcal{F}(\mathbf{x})$, e.g. two weight layers, are bypassed. The output of the skipped connection is formed by an addition $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. Image taken from [50].

The skipped connections are applied to the architecture of the previously discussed auto-encoder fusion networks AE-FN. In each input branch, each encoding block’s tensor is used twice: once for further encoding in the subsequent cell, and once for bypassing the network’s further encoding. The bypassed tensor is combined to the output of the decoding cell of the previous depth $\delta + 1$ with a combination node according to

$$\mathbf{T}_{dec}^\delta = \mathcal{N}_{cat}(\mathbf{T}_{dec}^{\delta+1}, \mathbf{T}_{enc,1}^\delta, \mathbf{T}_{enc,2}^\delta, \dots, \mathbf{T}_{enc,n}^\delta), \quad (3.13)$$

or in short form

$$\mathbf{T}_{dec}^\delta = \mathcal{N}_{cat}^i(\mathbf{T}_{dec}^{\delta+1}, \mathbf{T}_{enc,i}^\delta). \quad (3.14)$$

The first input tensor $\mathbf{T}_{in,i}$ is excluded from the bypassing, thus the output occupancy grid has no direct influence of the inputs. The encoding and decoding blocks remain the same as in AE-FN, so do their adjacency matrices

$$\mathbf{A}_{enc}^{AEB} = \mathbf{A}_{enc}^{AE}, \quad (3.15)$$

$$\mathbf{A}_{dec}^{AEB} = \mathbf{A}_{dec}^{AE}. \quad (3.16)$$

Additional to AE-FN, combination nodes in between the decoding cells are introduced as bypasses in the AE-B architecture.

Now, the formal description of the entire AEB-FN model is deducted. The tensor at the deepest part of the network remains \mathbf{T}^δ (Equation 3.7) and only the remaining

3.3 Neural Networks for Fusion of Camera and Radar Data

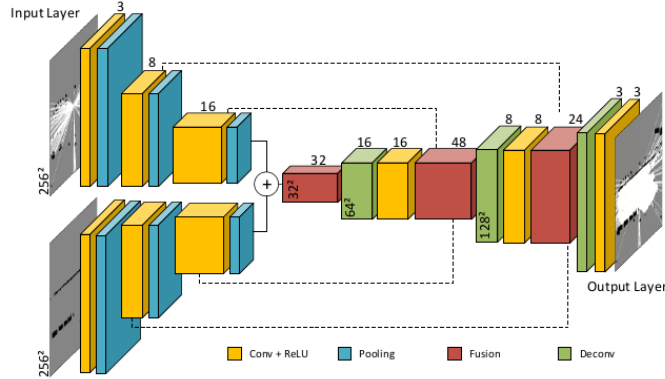


Figure 3.5: The architecture of an auto-encoder with bypass fusion network with $\delta = 2$ and $f = 8$. Only two input streams are shown for visualization purpose.

parts of the model change. The tensor after the first decoding block \mathcal{B}_{dec}^δ is

$$\mathbf{T}_{dec}^\delta = \mathcal{N}_{cat}(\mathcal{B}_{dec}^\delta(\mathbf{T}^\delta), \mathbf{T}_{enc}^\delta) \quad (3.17)$$

and the tensor after the second decoding block $\mathcal{B}_{dec}^{\delta-1}$ is

$$\mathbf{T}_{dec}^{\delta-1} = \mathcal{N}_{cat}(\mathbf{T}_{dec}^\delta, \mathbf{T}_{enc}^{\delta-1}) = \mathcal{N}_{cat}(\mathcal{B}_{dec}^{\delta-1}(\mathbf{T}^{\delta-1}), \mathbf{T}_{enc}^{\delta-1}). \quad (3.18)$$

The AEB-FN is helped in creating sharp class outlines by skipping encoding connections (Fig. 3.5). This allows small features of the input occupancy grids to propagate through the deep network more easily and produces a more detailed output.

3.3.2 Necessary Data and Available Datasets

In the previous subsections, neural networks for the fusion of camera and radar occupancy grids are introduced and the macro- and micro-architecture of the models are discussed. The content of this subsection is the data with which the networks are trained. More specifically, first, an overview of public datasets shows the main boundaries and tendency of research in the area of automotive perception. Then, the requirements for the sensor fusion problem of this work are outlined. The conclusion of this subsection is the choice of which dataset is applicable for training the sensor fusion networks.

Overview of Public Datasets

A manifold of datasets exists in the area of visual recognition and, more specifically, the automotive perception. Throughout the years, certain datasets were the quasi-gold standard for comparing state-of-the-art perception model performances. KITTI

3 Neural Networks for Sensor Fusion

[42], first introduced in 2013 and later further extended, was the first big benchmark of deep learning models specifically developed for the automotive context. The data was collected by front-facing, stereo grayscale and color cameras and a central lidar for ground truth, thus, allowing also depth estimations. With its focus on visual object detection, the annotations provided are 3D bounding boxes around the objects of interest.

Cityscapes [26] has been the next large benchmark in 2016, focusing on semantic segmentation of dense, urban scenarios relying on camera data only. The annotations for the 25 K images of the dataset are pixel-level classification into 30 different classes.

Based on these vision-based datasets, numerous deep learning approaches for image object detection and scene semantic segmentation arose. Still, the rising safety requirements for ADAS applications need redundancy and robustness to various kinds of environmental conditions, thus relying on camera sensors only is not sufficient. The usage of lidar sensors in automated cars is controversial. Most of the leading car manufacturers use laser technology, e.g. Uber, Waymo, and Toyota, but prominently, Tesla does not. Even though Tesla plans to replace lidar with extensive vision data processing [136], they still employ radar sensors. This reflects the general trend for automated cars of combining camera and radar sensors, hence, the recent datasets start including radar data, too.

The first large, public dataset with radar data is the nuScenes dataset [19] from 2019. Its focus is dedicated goal is leveraging sensor fusion algorithms for automotive use cases, thus a full sensor suite of six cameras, five radars and a central lidar sensors is employed. The radar data comes in form of detection lists.

There are four main tendencies in automotive perception datasets throughout recent years, which can be summarized as follows:

- *From camera-centered, towards multi-modal sensor systems:* In the beginning, the research came from computer vision, thus there was a need for standard computer vision algorithms and their respective datasets. More recently, the safety requirements of the automotive industry created a need of multi-modal, multi-sensor datasets.
- *Increasing dataset volumes:* On the one hand, new network models have higher computational requirements that are met with modern compute hardware, such as GPUs. On the other hand, large amounts of data is needed to train the complex models and cover a divers manifold of automotive scenarios. This need is met by ever growing datasets.
- *Shift from research institutes towards industrial, company-driven datasets:* The first effort of pushing perception model development with public, scientific datasets was done by universities and research institutes. Since the time when automated driving and ADAS appeared in public consciousness,

3.3 Neural Networks for Fusion of Camera and Radar Data

also car manufacturers and suppliers realized the need for datasets that are based on their requirements. As a result industrial, company-driven datasets are published, such as nuScenes [19], Waymo [122], Lyft Level 5 [54].

- *From semantic segmentation back to object detection and tracking:* The initial datasets have a focus on pixel-wise classification of imagery (*px* annotations in Tab. 3.1). Whereas semantic segmentation seemingly is trickier to master and potentially offers more information to the system, there is a trend towards datasets focusing on object detection (*obj* annotation in Tab. 3.1) and tracking again.

Table 3.1: Overview of automotive perception datasets [146].

Dataset, Year	Camera	Radar	Lidar	Annotations	Volume
KITTI, 2013 [42]	✓	-	✓	2D px, 3D obj	6 h
Cityscapes, 2016 [26]	✓	-	-	2D px	25 K images
RobotCar, 2016 [83]	✓	-	✓	-	1000 km
TorontoCity, 2016 [135]	✓	-	✓	2D px	56 K images
Mapillary, 2017 [95]	✓	-	-	2D px	20 K images
ApolloScape, 2018 [59]	✓	-	✓	2D px, 3D obj	100 K images
BDD100K, 2018 [145]	✓	-	-	2D px	100 K
H3D, 2019 [101]	✓	-	✓	3D obj	28 K frames
nuScenes, 2019 [19]	✓	✓	✓	2D px, 3D obj	40 K frames
Waymo, 2019 [122]	✓	-	✓	2D obj, 3D obj	10 h
Astyx, 2019 [87]	✓	✓	✓	3D obj	500 frames
RadarRobotCar, 2020 [13]	✓	✓	✓	-	280 km
Level 5, 2020 [54]	✓	-	✓	3D obj	400+ h

Data Prerequisites

The data fusion problem at hand has its roots in the sensor configuration of modern, mass-produced cars with a fragmented, distributed processing on the sensor side. Whereas the fusion is computed centrally in the fusion ECU, each sensor has certain processing steps onboard. This leads to the situation that the data for fusion is not available in raw format but in a digested form.

The same prerequisites apply for the training data: If the sensor-specific occupancy grids were computed from raw data, the amount of data would be great and the resulting environmental models rich in details and information. A radar sensor transmitting raw ADC data to the central compute unit would not only deliver target information about certain distinct points within the field of view of the sensor, but also tiny fragments or details about the continuous in between the detected

targets. Similarly, current camera sensor systems describe the environment with an object list, inhibiting particular pieces of details, such as object class, width, height or color. But if the raw image data was available, one could reconstruct the continuous 3D scene, e.g. on the basis of optical flow [53].

In the following paragraphs the required data from radar, camera and Lidar sensors and their necessary formats for feature-level fusion are discussed. The radar and camera sensors are mandatory, as they build the backbone for Level 2/2+ cars. Further, the Lidar data plays an important role in the training of the networks (Chapter 3.3.4), thus is part of this discussion, too.

Required Data - Radar

Recall the radar processing blocks (Chapter 2.1.1), where the raw ADC data is transformed to range-Doppler images by the first and second order FFT. The RDI is then used to detect targets together with their range, velocity and angle parameters. The typical automotive radar further computes the track information of objects from the target list.

Consequently, only the track list can be used for the environmental modeling, and—in recent developments of radar sensors—also the target lists are made available over the CAN bus. Thus, the required radar data is in form of a target list that describes dominant reflections within the field of view of the sensor. The basic, descriptive parameters of a radar target list are listed in Tab. 3.2.

Table 3.2: Parameter list of a typical radar target list.

Parameter	Description	Unit
Range	Distance from sensor to reflector	[m]
Angle (azimuth)	Horizontal angle	[deg]
Angle (elevation)	Vertical angle	[deg]
Velocity	Relative, radial velocity of the target	[m/s]
Intensity	Strength of the reflection	[dB]

Required Data - Camera

The format of a camera object list is more dependent on the camera system’s manufacturer than a radar target list on its supplier. This is based on the fundamental processing steps involved in radar processing, compared to the quasi freedom of design for a camera system.

Thus, the required camera sensor data is chosen based on the parameter list of the Mobileye 630 camera system. Mobileye is the dominant supplier of automotive vision modules with sales of over 10 million units per year since 2018, proving their system and data format to be well-established [7]. The set of parameters

3.3 Neural Networks for Fusion of Camera and Radar Data

contains information about the object’s semantics, such as its classification (e.g. *car*, *vulnerable road user (VRU)*, etc.), the object’s physical appearance and location, and other sparse information (traffic signs, lane markings, etc.).

Table 3.3: Parameter list of a typical camera object list.

Parameter	Description	Unit
Object class	Object classification result	
Position	Down- and cross-range	[m]
Size	Width and length of the object	[m]
Lane information	Current lane, lane equations	

Required Data - Lidar

As Lidar data is used for training purposes only (Chapter 3.3.4), the format requirements are not the same as for camera and radar data. Also, the typical Level 2/2+ car is not equipped with Lidar sensors, as the high price and poor packaging options hinder its large-scaled usage. The typical raw Lidar data is used for the ground truth occupancy grid generation. It comes in the format described in Tab. 3.4.

Table 3.4: Parameter list of a typical Lidar point cloud.

Parameter	Description	Unit
Range	Distance from sensor to reflector	[m]
Angle (azimuth)	Horizontal angle	[deg]
Angle (elevation)	Vertical angle	[deg]
Intensity	Normalized magnitude of return pulse	

Dataset of Choice

The data of the nuScenes dataset consists of raw camera imagery, processed radar detection lists and raw lidar point clouds. In each frame of the collection, all sensors have synchronized measurements at a rate of 2 Hz and provide six camera images, readings from five radars and one central lidar (Table 3.5). Camera images are available as raw 1600×900 px RGB images, lidar point clouds as 3D coordinates with respective reflectivity values. Radar detections come as 3D coordinates, too, but with more features than the lidar readings, such as velocities and uncertainties.

Because one of the main focuses of nuScenes is object detection, the annotations are in form of 3D bounding boxes around the objects of interest in the scene. The properties of these bounding boxes are the position, size, orientation and velocity in 3D Cartesian coordinates. As the the aim of sensor fusion neural networks is to

Table 3.5: Data format of the nuScenes dataset.

Sensor Modality	Provided Data	Data Format
Camera	Raw images	1600x900 px
Radar	Detection-level point cloud	$n_{pts} \times 16$
Lidar	Raw point cloud	$n_{pts} \times 4$

combine camera and radar detections for free space estimation, the raw data from the dataset has to be prepared first.

3.3.3 Data Preparation

In this subsection, the processes of getting to the final inputs and the ground truth for the training is described. First, the inputs, then the lidar-based ground truth is discussed.

Input Occupancy Grid Generation

Multiple sensor readings of the same modality can be treated in different ways. Either each of the i sensors is generating an own occupancy grid $x_{OG}^1 \dots x_{OG}^i$, where all x_{OG}^i form an individual input to the grid fusion network. Another way of forming the input occupancy grids is to group the sensor modalities to one occupancy grid each, thus having one grid per modality. The benefits of the second approach are a better adaptability to changes in the hardware setup, e.g. additional sensor or adjustment of the sensor placement, and, in consequence to the compressed input volumes, a smaller computational complexity of the neural network itself. Thus, each sensor modality is translated into an occupancy grid, namely x_{OG}^{rad} , x_{OG}^{cam} and x_{OG}^{lid} (Table 3.7, Fig. 3.6).

The radar grid is generated by first translating and rotating each of the five radar sensor’s detection lists into the uniform car-centered coordinate system that has the same origin as the lidar sensor. Typically, the radar sensors detect few tens of targets per cycle, resulting in around 100 entries in the combined target list. Once the global radar detection list is set, the individual detections are mapped into a car-centered occupancy grid that also has the lidar origin as the center. The mapping is done with the inverse sensor model from Chapter 2.2.3 that converts the detections into spatial occupancy probabilities. Depending on the type of ISM, the x_{OG}^{rad} can denote the obstacles only, or express a rudimentary free space estimation, too. In the case of the left figure in Fig. 3.6, the free space modeling is part of the ISM.

Since the data format of the 6 cameras is *raw images* in the nuScenes dataset, the data has to be further processed. One way to obtain an object list is to run object detection algorithms on the raw data. An other, more pragmatic way is to assume the object annotations, which are in the form of three-dimensional bounding boxes,

3.3 Neural Networks for Fusion of Camera and Radar Data

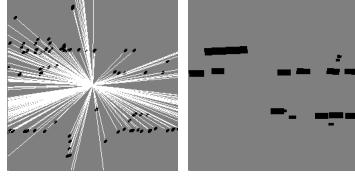


Figure 3.6: Combined occupancy grids of radar x_{OG}^{rad} and camera sensors x_{OG}^{cam} .

as ideal object detections of modern automotive vision systems, such as MobilEye ME630.

Once the camera object list is obtained, it is converted to an occupancy grid by projecting the three-dimensional bounding boxes to their bases in order to obtain the obstacle footprints. All cells of x_{OG}^{cam} within the boundaries of an obstacle are assumed to be occupied and the rest of the area is populated with the unknown state (Fig. 3.6, right).

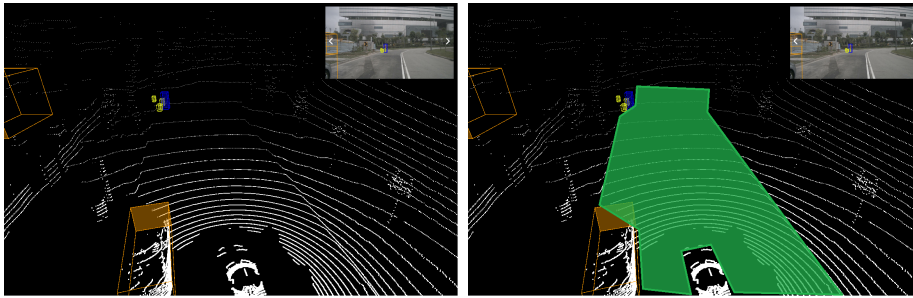


Figure 3.7: Lidar point cloud used for RANSAC-based ground plane estimation. Image taken and adapted from *nuScenes dataset explorer*¹.

Ground Plane Estimation for Ground Truth

In the previous paragraph, the generation of the network inputs are discussed. The feature-level sensor data are transformed into sensor-specific occupancy grids, which are to be fused by the fusion networks. In order to train the correct fusion of occupancy grids and to find a dense free space estimation within the result, the respective ground truth has to be constructed.

In the nuScenes dataset, a temporally synchronized lidar scan is provided to each radar and camera measurement. This data is used to generate a dense free space estimation within the two-dimensional occupancy grid resulting in the lidar grid x_{OG}^{lid} . Along with the information about drivable space, the radar and the camera data are fused with Bayesian fusion (Chapter 2.3.2) and combined with the lidar-

¹<https://www.nuscenes.org/nuscenes#explore> (scene-0011), accessed 16. Oct. 2020.

3 Neural Networks for Sensor Fusion

based occupancy grid x_{OG}^{lid} . The result is the ground truth grid x_{OG}^{gt} used for training the networks (Table 3.7).

The three-dimensional lidar point cloud needs to be processed with a ground plane estimation algorithm in order to obtain a valid free space estimation. The ground plane is assumed to be the street surface, where typically most of the lidar reflections are located. Usually, ground plane estimation methods are used to remove the reflection points of the ground in order to reveal the remaining obstacles. Here, the plane is used for generating the free space estimation. Different techniques are researched for ground plane estimation in point clouds:

- *Fixed height ground plane:* The easiest approach for receiving a ground plane is to take the mounting height z of the lidar as fixed and neglect the pitch and roll angles of typical car motion. All points of the lidar point cloud within a certain range below z are assumed to lie on the ground plane, thus, on the road surface.

This approach may be the fastest to compute, but delivers invalid ground plane estimates due to the pitch and roll angles of accelerating and turning the vehicle.

- *Neural network based estimation:* A neural network for estimating the ground plane of a three-dimensional point cloud is presented in [77], where the point cloud is seen from the bird’s eye view (BEV) and the height of the points is encoded in the feature channels of the resulting two-dimensional BEV image. This image is then processed with a model similar to U-Net [113] to estimate the road surface.

Additionally, special training needs to be done for the plane estimation network to work properly, before generating the x_{OG}^{lid} . Without a ground truth for training the plane estimation network, this is not viable for the given case.

- *RANSAC-based fitting:* This iterative approach finds the optimal ground plane, if run for enough iterations k [40]. First, randomly three points are selected, defining a plane P_1 . Now all points are counted that lie within a certain margin Δ_z to P_1 , resulting in a score s_1 . If a second plane P_2 has a larger score s_2 , it is assumed to be the best fit \hat{P} . After i iterations, the plane \hat{P} is the optimal plane fit for the given point cloud.

Limitations of this approach are the choice of the parameters Δ_z and k , and the possibly wrong estimations, caused by planes in the point cloud with more reflection points than the ground plane. This error can be ruled out by taking the normal of the plane into consideration, as seen in Figure 3.8b, where plane 1 and 3 have horizontal surface normals, but only plane 2 has the correct orientation.

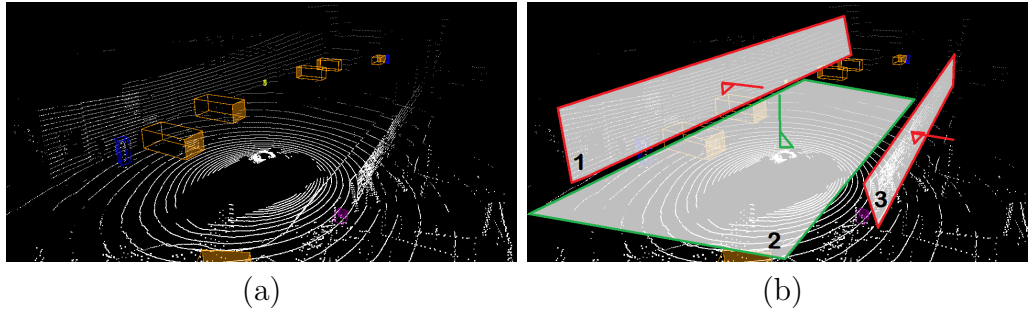


Figure 3.8: RANSAC-based ground plane estimation. (a) shows a typical lidar measurement combined with object annotations. (b) depicts possible plane estimates of the RANSAC algorithm. Only plane 2 has a valid orientation. Image taken and adapted from *nuScenes dataset explorer*¹.

The RANSAC-based plane fitting is used to extract the drivable road surface of the lidar point clouds. Whereas it might be computationally expensive compared to the other approaches, it is computed offline and only once before the actual training to obtain the dataset. In turn, it robustly finds the ground plane of the road independently of the roll and yaw angle of the car with respect to the surface plane.

In the nuScenes dataset, the Lidar generates an empirical estimate of around 150,000 detections per cycle, of which 20,000 - 80,000 are found on the ground plane, depending on the road type and traffic situation. What might cause a different amount of reflections on the ground plane, may be a large structure beside the car. For example a truck beside the ego vehicle casts a large shadow on the road surface and inhibits large portions of the lidar rays. If the surface normal of a proposed plane is not pointing upwards, the points lying on that plane are removed from the point cloud (Algorithm 1). In the end, roughly 50% of the reflection points are outliers of the ground plane ($\epsilon = 0.5$) and the plane is described by three sample points ($s = 3$). The amount of RANSAC iterations k is depending on the assurance p of finding the correct plane [40]:

$$k = \frac{\log(1 - p)}{\log(1 - (1 - \epsilon)^s)} \quad (3.19)$$

According to Equation 3.19, the confidence level of 99% of finding the correct plane with 50% outliers is reached after $k = 35$ iterations (Table. 3.6). This configuration of RANSAC was used to process each frame of the dataset and extract the free space estimate. All reflection points in the estimated plane are converted to occupancy probabilities and only cells with occupancy probability $P_{x,y}^{lid} < 0.5$ are taken into account. Additionally, the occupancy information of the radar and camera detections are added for an enhanced x_{OG}^{gt} .

¹<https://www.nuscenes.org/nuscenes#explore>, accessed 16. Oct. 2020.

Algorithm 1 RANSAC-based plane fitting algorithm.

Input: point cloud \mathbf{x}
Output: point cloud $\hat{\mathbf{s}}$
 $n = \text{count}(\mathbf{x})$
 $\Delta_z = 0.07\text{m}$
 $k = 35$
 $\hat{\mathbf{s}} = 0$
for $i = 1$ **to** k **do**
 $p_i = \text{randomSelection}(\mathbf{x}, 3)$
 $P_i = \text{createPlane}(p_i)$
 for q **in** \mathbf{x} **do**
 if $\text{calcDistance}(P_i, q) < \Delta_z$ **then**
 $\mathbf{s} = \text{appendPoint}(\mathbf{s}, q)$
 end if
 end for
 if $\text{count}(\hat{\mathbf{s}}) < \text{count}(\mathbf{s})$ **then**
 if $\text{normalVector}(\hat{\mathbf{s}}) \geq 0.95$ **then**
 $\hat{\mathbf{s}} = \mathbf{s}$
 else
 $\mathbf{x} = \text{remove}(\mathbf{x}, \mathbf{s})$
 end if
 end if
end for

Table 3.6: Confidence levels of finding the correct plane with RANSAC.

Outliers ϵ	10%	20%	30%	40%	50%	60%	70%
Iterations k	4	7	11	19	35	70	169

Training Data Overview

All the generated OGs are chosen to span over the same area around the car, with the car-centered coordinate system in the center of the grid (Fig. 3.9). The grid dimensions are $d_x \times d_y \times d_f$, with d_x , d_y as the spatial dimensions and d_f as the number of feature channels per grid cell. The features of the OGs are probabilities of three states (s_0 : free, s_1 : unknown, s_2 : occupied), thus $d_f = 3$ (Table 3.7).

The benefit of a reduced grid size is the fast processing. The required computations for one forward pass scale with the grid size as seen in Table 3.9. A doubling of the grid d_x and d_y to 512×512 causes four times the computational costs, and increasing the grid to 1024×1024 enlarges the costs by a factor of 16. On the other hand, the cell resolution has to be large enough to capture small details. However, low-dimensional grids are processed faster, resulting in potentially faster update

3.3 Neural Networks for Fusion of Camera and Radar Data

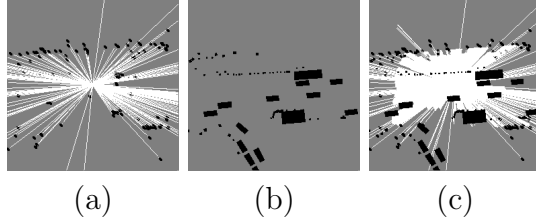


Figure 3.9: One exemplary frame of the training data: (a) x_{OG}^{rad} , (b) x_{OG}^{cam} , (c) x_{OG}^{gt} .

intervals. Empirically, a resolution of $\delta_x = 0.25$ m offers a well-balanced trade-off between quality and cost.

With a cell resolution δ_x of 0.25 m, the grid covers an area of $64\text{m} \times 64\text{m}$ around the car. The grid placement and extent is under the influence of the use-case of the ADAS system. In highway or rural scenarios, it is beneficial, if the center of the grid is in front of the car and the grid may not be quadratic, as high velocities and straight roads shift the area of interest there. In urban areas, dense traffic situations require a 360 degree awareness around the car (Fig. 2.13).

Table 3.7: Occupancy grid maps obtained from the nuScenes dataset.

Name	Type	Source	Dimensions
x_{OG}^{cam}	Input	Camera detections	$d_x \times d_y \times 3$
x_{OG}^{rad}	Input	Radar detections	$d_x \times d_y \times 3$
x_{OG}^{lid}	GT	Lidar detections	$d_x \times d_y \times 3$
x_{OG}^{gt}	GT	$x_{OG}^{rad} + x_{OG}^{lid} + \text{Annotations}$	$d_x \times d_y \times 3$

The dataset has a total of 1,000 scenes, each with 20s of measurements at 2 Hz (in average 40 frames per scene). Using the nuScenes *trainval* package containing 850 scenes, a total of 34,149 individual frames are available. Out of this set, frames with few or no detections from either the cameras or the radars were discarded. The resulting 5,595 selected frames are then augmented with mirroring and random rotations to a total of 44,760 training frames. Each training frame has all three occupancy grids (x_{OG}^{rad} , x_{OG}^{cam} , x_{OG}^{gt}) (Table 3.7).

3.3.4 Training Configuration and Training Process

The training is done with processed data coming from the nuScenes dataset. First, sensor-specific occupancy grids are calculated for camera and radar. Additionally, the information from all sensor modalities including a lidar are used for the ground truth occupancy grid. Then, the training is performed similar to other auto-encoders for semantic segmentation tasks, such as U-Net, but with two input branches.

Training Configuration

The loss function is a pixel-wise softmax classification of the output of the network's last layer. The classification is pixel-wise as each pixel's individual occupancy probability is estimated. Each occupancy state's error ϵ is computed according to

$$\epsilon_{(x,y)}^{s_i} = y_{(x,y)}^{s_i} - y_{0(x,y)}^{s_i}, \quad i \in \{0, 1, 2\}, \quad (3.20)$$

with $y_{(x,y)}$ and $y_{0(x,y)}$ being logits and ground truth of one pixel. All pixel-wise errors combined yield the pixel-wise L2 loss, which is summed up over all coordinates (x, y) in the grid to compute the total unweighted L2 loss [81]

$$L2 = \sum_{x,y} \sum_i (\epsilon_{x,y}^{s_i})^2. \quad (3.21)$$

During training, the ADAM optimizer is used for minimizing Equation 3.21 with an initial learning rate of 0.002, $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The learning rate is decaying by 2 every 1,000 iterations.

Besides the loss function, also the accuracy and the following evaluation metrics are logged during training for both, training and validation set.

Network Architecture Exploration

The combinations of architectural design and hyper parameters are plenty and finding an optimal neural architecture for a given task is not trivial. The set of parameters to search for in this work are the training parameters (learning rate, decay) and the architectural parameters (depth, number of initial feature channels). Different techniques are applied throughout literature to find an optimal set.

- The most basic way of finding a neural architecture is *handcrafting the model*. This requires a solid experience, careful analysis and plenty of time to find a right configuration by changing the parameters one by one.
- The *grid search* covers all combinations in a predefined set of parameters. This is the most time consuming method and potentially infeasible, if there are too many parameters to check. But typically, the grid search covers a large search space and gives insights in the trade-offs in between certain parameters.
- In order to speed up the grid search, the *random search* samples random parameter combinations and evaluates them. This method is viable, if computing all the potential combinations with a grid search is infeasible.
- The optimal set of training and architectural parameters can also be found with Bayesian optimization, which employs Gaussian processes. Those describe the posterior distribution of functions that describe the deep learning

3.4 Fusion Quality and Hardware Requirements Analysis

model. With an increasing number of samples (e.g. trained networks), the posterior distribution improves and the Gaussian process shows with increasing certainty, which combinations in the search space are of high importance (e.g. variance). The iterative search then focuses on the region of those combinations.

Due to the uniform coverage of the search space of parameter choices, the grid search is chosen to find an optimal set of the number of initial filter channels χ and network depth δ . All permutations of $\chi \in [4, 8, 16, 32]$ and $\delta \in [3, 4, 5, 6]$ are trained with equal training setups and data. Consequently, 16 network variants per network family (FC-FN, AE-FN, AEB-FN) are trained until convergence, resulting in a total of 48 networks. The short notation for a network variant is FN - δ - χ , e.g. the fully convolutional fusion network with a depth of 3 and 8 initial feature channels is denoted as FC-3-8.

3.4 Fusion Quality and Hardware Requirements Analysis

In this Section, the proposed networks are evaluated on the datasets introduced in Section 3.3.2 and compared according to well-known evaluation metrics for semantic segmentation. Additionally to the fusion quality assessment, the computational costs of these networks are calculated and checked for real time capabilities on state-of-the-art CNN accelerator hardware. Finally, an optimized variant of the networks is deduced in order to speed up the inferences and reduce the memory footprints of the networks.

In this analysis, the computational costs are defined as the number of operations that the hardware has to calculate and the number of parameters that have to be stored. Inference times and energy consumption vary for different hardware setups, thus, are not part of this analysis. The inference times play a role in the hardware-specific automated neural architecture searches conducted in Chapter 4.

3.4.1 Experimental Evaluation

Evaluation Metrics

The comparison of the results is based on the common evaluation metrics for semantic segmentation tasks. Variations on pixel accuracy and intersection over union (IoU) are evaluated for each trained network, based on the metrics of the PASCAL VOC challenge [38]. Following the nomenclature and calculations of [81], n_{ij} denotes the number of all pixels with a misclassification, where the network predicts class j , whereas class i is true. Similarly, n_{ii} stands for the number of all correctly classified pixels. n_{cl} denotes the number of classes, and here, $n_{cl} = 3$ for the three

3 Neural Networks for Sensor Fusion

occupancy states (free, unknown, occupied). The total number of pixels of class i is $t_i = \sum_j n_{ij}$ [81].

The evaluation metrics (pixel accuracy m_{pAcc} , mean pixel accuracy m_{mAcc} , mean IoU m_{mIoU} , frequency weighted IoU m_{fIoU}) are computed as follows. The pixel accuracy m_{pAcc} relates the correctly predicted pixels to the total number of pixels of each class

$$m_{\text{pAcc}} = \frac{\sum_i n_{ii}}{\sum_i t_i}. \quad (3.22)$$

The mean value over all classes

$$m_{\text{mAcc}} = \frac{\sum_i n_{ii}}{\sum_i t_i} \quad (3.23)$$

gives better insight over an overall performance of the network.

$$m_{\text{mIoU}} = \frac{\sum_i n_{ii}}{\sum_i t_i}. \quad (3.24)$$

$$m_{\text{fIoU}} = \frac{\sum_i n_{ii}}{\sum_i t_i}. \quad (3.25)$$

Evaluation Results

Table 3.8 shows the comparative results¹ based on the metrics defined above. Additionally to the PASCAL VOC metrics, also the validation accuracy and validation loss are compared. The training is based on equal settings: The same training, test and validation data, and with the same seed for random number generations. Each network family is evaluated on the variants based on the grid search described in Section 3.3.4. Each network is trained once on the raw dataset \mathcal{D} , and once on its augmentation \mathcal{D}^* .

The fusion networks performances are compared side-by-side with the traditional Bayesian grid fusion, described in Chapter 2.3.2. As the Bayesian fusion has no training phase, nor a model, the columns **Shape**, **val. Acc** and **val. Loss** are not applicable. To highlight the different architectural variants, the smallest and largest χ , and a selection of the most expressive encoding depths δ are shown.

The investigated fully convolutional fusion networks are only able to mimic the fusion quality of probabilistic grid fusion techniques. The performance gain in each of the metrics is minor, independent of the variant of the FC-FN. Also, the different architectural variants are performing similarly, regardless of increasing network size, especially for the more robust, augmented dataset \mathcal{D}^* . This behavior is based on the constant-sized network design that suppresses the learning of abstract features, and the limited receptive field of each filter that does not allow to catch contextual

¹Experimental results published in [10].

Table 3.8: Results of Experimental Evaluation of Fusion Networks. Comparison between the small subset \mathcal{D} and its augmentation \mathcal{D}^* .

Network	Shape δ χ	val. Acc		val. Loss		m_{IoU}		m_{IoU}		m_{pAcc}		m_{mAcc}	
		\mathcal{D}	\mathcal{D}^*	\mathcal{D}	\mathcal{D}^*	\mathcal{D}	\mathcal{D}^*	\mathcal{D}	\mathcal{D}^*	\mathcal{D}	\mathcal{D}^*	\mathcal{D}	\mathcal{D}^*
<i>Bayesian</i>	-	-	-	-	-	0.772	0.794	0.888	0.826				
FC-1-4	1 4	0.881	0.874	0.065	0.064	0.802	0.835	0.773	0.829	0.873	0.908	0.877	0.892
FC-1-16	1 16	0.883	0.875	0.060	0.062	0.829	0.827	0.815	0.804	0.900	0.893	0.887	0.887
FC-4-4	4 4	0.879	0.875	0.065	0.062	0.736	0.807	0.715	0.790	0.837	0.885	0.843	0.880
FC-4-16	4 16	0.884	0.881	0.058	0.058	0.851	0.844	0.847	0.819	0.917	0.901	0.909	0.909
AE-4-4	4 4	0.790	0.831	0.097	0.079	0.439	0.725	0.676	0.771	0.805	0.869	0.530	0.833
AE-4-16	4 16	0.800	0.838	0.091	0.075	0.510	0.715	0.678	0.769	0.804	0.866	0.611	0.823
AE-5-16	5 16	0.802	0.807	0.090	0.088	0.551	0.624	0.712	0.777	0.825	0.871	0.668	0.717
AE-6-16	6 16	0.641	0.786	0.165	0.101	0.235	0.453	0.502	0.683	0.703	0.815	0.330	0.544
AEB-4-4	4 4	0.862	0.888	0.067	0.058	0.798	0.840	0.841	0.846	0.912	0.914	0.876	0.911
AEB-4-16	4 16	0.895	0.907	0.056	0.049	0.884	0.910	0.892	0.914	0.941	0.955	0.941	0.949
AEB-5-4	5 4	0.878	0.887	0.062	0.058	0.825	0.843	0.846	0.835	0.915	0.908	0.898	0.914
AE-B-5-16	5 16	0.729	0.905	0.116	0.052	0.519	0.921	0.622	0.925	0.741	0.961	0.803	0.954
AEB-6-4	6 4	0.875	0.893	0.064	0.054	0.754	0.870	0.794	0.893	0.883	0.943	0.835	0.931
AEB-6-16	6 16	0.875	0.904	0.074	0.050	0.789	0.895	0.793	0.890	0.883	0.941	0.868	0.943

information. Thus, FC-FN learn to reproduce the cell-wise probabilistic fusion with similar performance as Bayesian fusion, according to image quality (Figure 3.10) and evaluation metrics (Table 3.8).

The auto-encoder architecture of the AE-FN allows to learn more complex correlations between the inputs, if the encoding procedure compresses the grid size to a certain level. A good compression rate for learning contextual information of the car’s surrounding is achieved, when the receptive field of the last convolutional layer spans over the complete input occupancy grids. In order to classify large unoccupied regions in the grid, deep encoding is needed, which causes extensive blurring of the classification results. The performance metrics highlight the degradation due to deep encoding: The pixel and mean accuracies decrease for high δ values. The Jaccard indices show a high dissimilarity between the ground truth and the AE-FN predictions, not even close to the quality of Bayesian or FC-FN results. The peak of AE-FN fusion quality is for $\delta = 4$, and deeper encoding leads to high blurring degradation. Still, the AE-FN output occupancy grids show the benefit of auto-encoders, compared to constant-sized network architectures: Large context recognition is observable, thus the free space estimation becomes viable.

On the other hand, small obstacles and objects of the input tensors are suppressed by the blurring effect. To cope with this, the skipped connections help to bypass deep encoding parts of the AEB-FN models. As a result, the AEB-FN generate a similar free space estimation, but a sharpened classification result for small obstacles and occupied regions. The respective performance metrics show the superiority of this approach to the traditional Bayesian fusion. Especially networks with a large value for χ show a high capability of context recognition, as more trainable weights are available. Also, compared to AE-FN, deeper encoding is possible due to the increased training stability. The fusion performance and robustness are further boosted with the augmented dataset \mathfrak{D}^* .

Training shows that these networks tend to overfit on dataset \mathfrak{D} , as most road scenarios have similar layouts with the car driving on a fairly straight road. Without data augmentation, overfitting occurs for the free space estimation ahead and behind the vehicle. Obstacles ahead of the vehicle are overwritten with the dominant estimate learned, which is zero occupancy ahead. Data augmentation in \mathfrak{D}^* with random rotations mitigates this behavior.

3.4.2 MAC Counts and Memory Requirements

Three network families and the two architectural parameters δ and χ are subject for comparison. Due to the inferior fusion performance of the FC-FN, this section focuses on the promising auto-encoder network families. Each network family’s dependency to the two dimensions δ and χ , the impact on computational costs and fusion quality is analyzed. This chapter shows the comparison of computational costs with regards of the number of parameters that need to be stored and define

3.4 Fusion Quality and Hardware Requirements Analysis

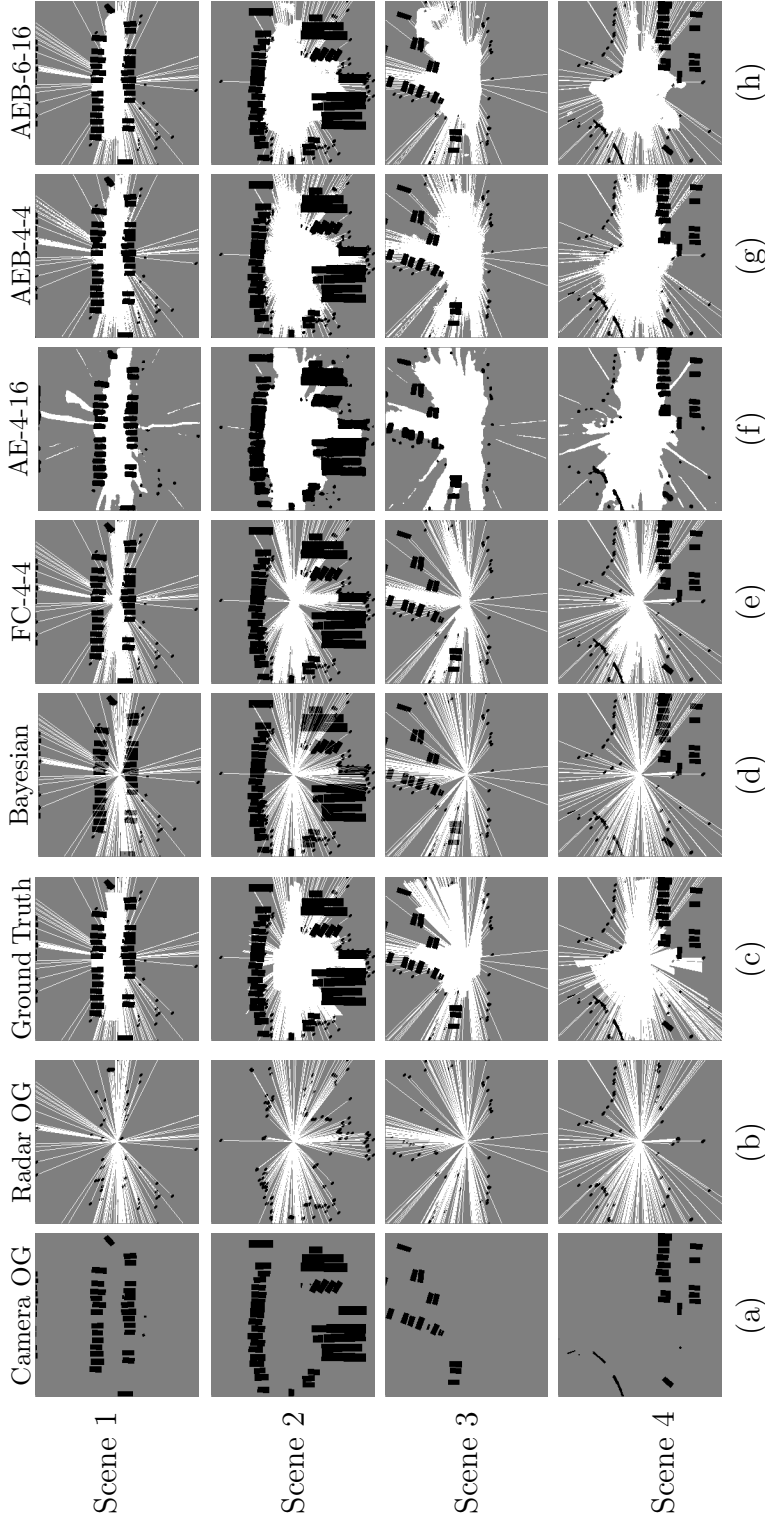


Figure 3.10: Comparison of grid fusion results. Column-wise: (a) Camera and (b) radar inputs. (c) Ground truth OG obtained from lidar measurements combined with object annotations. Results of (d) classical Bayesian fusion ($\delta = 4, n_{ch} = 4$), (e) FC-FN ($\delta = 4, n_{ch} = 4$), (f) AE-FN ($\delta = 4, n_{ch} = 16$), (g) AEB-FN ($\delta = 4, n_{ch} = 4$), (h) AEB-FN ($\delta = 6, n_{ch} = 16$). Row-wise: Different scenarios for fusion.

3 Neural Networks for Sensor Fusion

the weights of the networks, and the number of computations for one inference pass in form of multiply-accumulate (MAC) operations. Other aspects like energy consumption per forward pass or training costs are not part of this analysis, as these numbers are heavily dependent of the target platform and the training is expected to be done offline. The number of parameters are extracted with the Tensorflow API in the Python code using `float32` variables, and the MAC count is estimated by converting the Tensorflow model architectures into Caffe’s `prototxt` format [46].

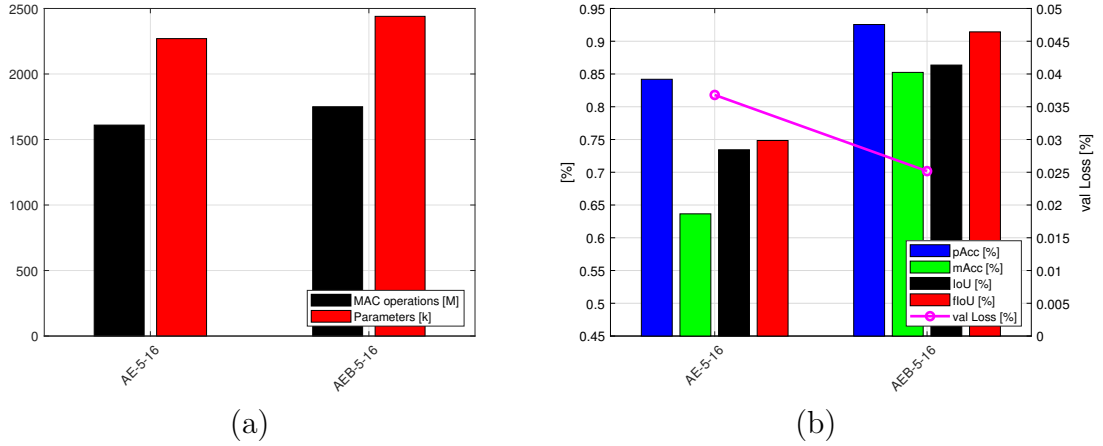


Figure 3.11: Comparison of the grid fusion network families, each with $\delta = 5$ and $\chi = 16$. (a) The computational costs and (b) the fusion quality shown by the evaluation metrics from Section 3.4.1.

As a result, one can observe that the AE-FN and AEB-FN have only minor differences in computational costs. On the one hand, the AE-FN needs 2,270k parameters and 1,610M MAC operations (Fig. 3.11a). On the other hand, bypassed feature maps in the AEB-FN architecture enlarge the tensor size in the decoder, thus leading to an increased computational cost. The AEB-FN needs 2,440k parameters and 1,750M MAC operations, which is an increase of 7.5% and 8.7%, respectively (Fig. 3.11a). Even though the costs are similar, the fusion performance of the AEB-FN shows significant improvement over the AE-FN. As the skipped connections help small details to pass through the deep parts of the network, AEB-FN improve all metrics considerably (Fig. 3.11b). For the same network configuration ($\delta = 5$, $\chi = 16$), the relative improvements of the evaluation metrics are as follows:

- $m_{\text{pAcc}} + 9.8\%$: (AE-5-16 = 84.2%, AEB-5-16 = 92.5%)
- $m_{\text{mAcc}} + 33.8\%$: (AE-5-16 = 63.7%, AEB-5-16 = 85.2%)
- $m_{\text{mIoU}} + 17.7\%$: (AE-5-16 = 73.4%, AEB-5-16 = 86.4%)
- $m_{\text{fIoU}} + 22.0\%$: (AE-5-16 = 74.9%, AEB-5-16 = 91.4%)

3.4 Fusion Quality and Hardware Requirements Analysis

Conclusively, the bypassed connections come at a minimal cost, when traded off to the benefit in the fusion performance of the network models.

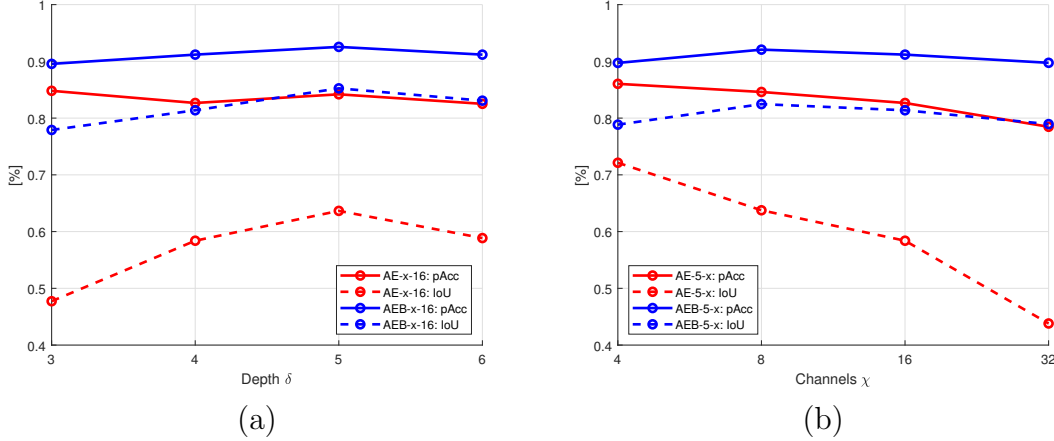


Figure 3.12: The fusion performances of the fusion model families over the architectural parameters. Correlation to the (a) depth and (b) the number of initial feature channels.

Having a closer look at the impacts of the network architecture parameters δ and χ , respectively, different effects can be observed. For a fixed number of initial feature channels ($\chi = 16$), the networks show increasing performances for larger encoding depths δ (Fig. 3.12a). AEB networks perform in a range of $m_{\text{pAcc}} = 91\% \pm 1\%$ and $m_{\text{mIoU}} = 77\% - 85\%$, with the performance peak at $\delta = 5$. The encoding depth of 5 shows the best scores for AE-FN and AEB-FN for input grids $\text{OG} \in \mathbb{R}^{256 \times 256 \times 3}$. In contrast, manipulating χ with a fixed depth $\delta = 5$ shows a negative impact (Fig. 3.12b). The AE-FN show the best performance at the lowest measured $\chi = 4$, which continually degrades with higher channel numbers. Finally, AEB-FN achieve the best results at $\chi = 8$.

The impact of the architectural parameters on the MAC count and the number of parameters can be analyzed separately. For a fixed χ , the computational costs are logged with a changing δ —and vice versa. The encoding depth is scaling the MAC count linearly (Fig. 3.13a), whereas the required number of parameters scales exponentially (Fig. 3.13b) for all networks. Altering χ with a fixed depth δ shows the impact of the number of initial feature channels. It is scaling exponentially for all networks, both in MAC count (Fig. 3.13c) and the number of parameters (Fig. 3.13d). Thus, the computations scale linearly with the depth δ , but exponentially with χ . The amount of parameters scale exponentially in both cases.

3 Neural Networks for Sensor Fusion

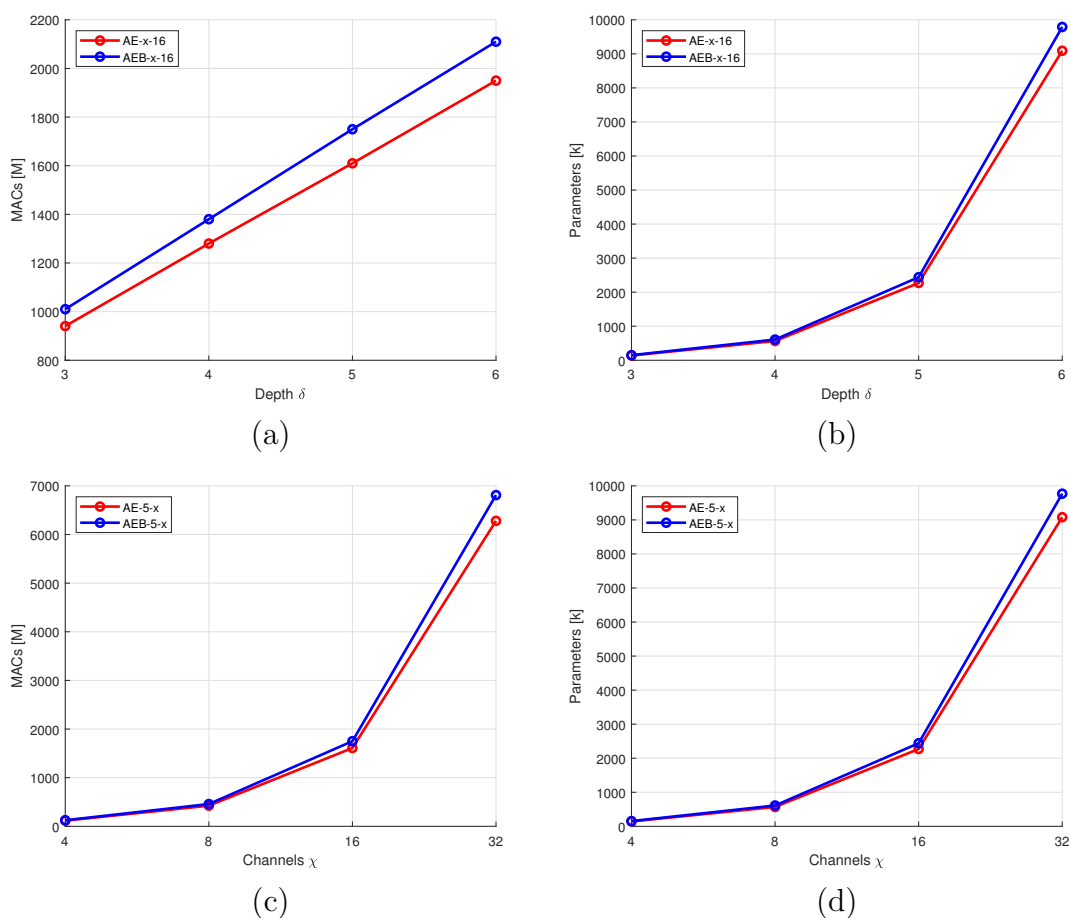


Figure 3.13: Computational costs for different architecture parameters. (a) MAC count vs. varying depths δ , (b) amount of parameters vs. varying depths δ , (c) MAC count vs. varying number of initial feature channels χ , (d) amount of parameters vs. varying number of initial feature channels χ .

Quality-Cost Trade-Off

On the one hand, as seen in Fig. 3.12a, increasing the depth increases the fusion performance, especially of networks with skipped connections. On the other hand, increasing χ impacts the fusion quality rather negatively (Fig. 3.12b), as the amount of parameters is sufficient at lower χ and additional weights lead to overfitting, thus degradation of fusion quality.

The sweet spot for AE-FN is $\delta = 5$, as the metric m_{mIoU} is maximal there, and $\chi = 4$, as fusion performance degrades for more parameters at given depth of five (AE-5-4). AEB-FN are mostly unaffected of the channel number. Consequently, a depth of five and eight initial feature channels are the best choice for fusion quality

(AEB-5-8).

Estimated Sufficiency for Deployment

The computational requirements of the fusion networks appear to be low when compared to large image processing networks like *AlexNet* [75], ResNet [50] or U-Net [113]. But in order to perform real-time computations on a low-power embedded accelerator, the networks need to be further optimized for deployment. The EV6x Embedded Vision Processors of Synopsys’ DesignWare IP offer ASIL B, C or D ready SoC blocks with up to four dedicated CNN accelerator cores, each performing with up to 880 MACs per cycle [124].

Table 3.9: Estimated inference times on Synopsys EV61 processor with different CNN-engine configurations at 500 MHz operating frequency for the auto-encoder fusion network families

Network	Grid	EV61 [ms]		
		CNN 1	CNN 2	CNN 4
AE-5-16	256	3.66	1.83	0.91
AEB-5-16	256	3.98	1.99	0.99
AE-5-16	512	14.6	7.33	3.66
AEB-5-16	512	15.9	7.93	3.97
AE-5-16	1024	58.6	29.3	14.7
AEB-5-16	1024	63.5	31.7	15.9

In Table 3.9, the different architecture families with the best performing variant from the grid search in Section 3.3.4 along with their estimated runtimes on the Synopsys EV61 processor with different CNN-engine configurations are shown. One forward pass of any of the fusion networks take up to few milliseconds at an input grid size of $\mathbf{T}_{in} \in \mathbb{R}^{256 \times 256 \times 3}$ and two input streams. The difference in execution time between AE-5-16 and AEB-5-16 is 8%, but still, both of the networks can perform real time operation on the given accelerator. However, once the input occupancy grid size is scaled to a higher resolution or larger coverage area of the sensors, the baseline networks perform too slowly. Especially the configuration with one or two CNN cores (CNN1 and CNN2) and a high resolution input grid $\mathbf{T}_{in} \in \mathbb{R}^{1024 \times 1024 \times 3}$, the real-time operation is not possible at a sampling rate of 50 Hz, which is equivalent to an upper limit of 20 ms per forward pass. Further, this effect is strengthened by the fact that the execution time estimates are based on 100% utilization of the 880 MAC/cycle of the CNN engines, which is not viable under real circumstances. Under real conditions, where feature maps not always fill all of the processing elements of

the accelerator’s systolic array, additional loading times slow down the absolute throughput.

3.4.3 Optimization and Compression Techniques

In order to reduce the computational costs and meet the real-time requirements based on the Synopsys EV61 processors, the network models have to be optimized. The majority of computational costs comes from the convolutional weight layers, especially in convolutional auto-encoder networks without large fully connected classification layers [118]. Based on this fact, the compression of the standard convolutions \mathbf{o}_{conv} is the focus of this section.

Model compression can be approached with various approaches [23]. Parameter pruning and knowledge distillation, where the redundant parameters are discarded from the network and the model is iteratively retrained in a compact graph, both are model compression techniques that retrospectively ease the computational costs. Besides the subsequent approaches, redesigning the convolution operation \mathbf{o}_{conv} into a compact and efficient format is a viable, widely used, a priori approach. Also, pruning and distillation focus heavily on fully connected layers of mostly classification networks, which does not apply for the grid fusion networks of this work [23].

Finally, weight quantization is an effective method of compressing the memory footprint of the model by reducing the precision of weights from `float32`, down to `float16`, or even integer precision `int16` or `int8` [60]. Research goes into the direction of further compressing the weights down to the absolute minimum of one bit, i.e. binary weights [104].

In this section, the approach of compressing the model size is by introducing parameter-efficient operations and replacing the computationally heavy, standard convolutions \mathbf{o}_{conv} in the blocks of the network. Recall that each encoding and decoding block of the auto-encoder networks incorporates a convolution \mathbf{o}_{conv} or deconvolution \mathbf{o}_{deconv} , respectively, described in the adjacency matrix of its directed acyclic graph. In order to reduce the amount of feature channels d_C of tensor $\mathbf{T} \in \mathbb{R}^{d_x \times d_y \times d_C}$ entering the operation, first a parameter-efficient, point-wise convolution $\mathbf{o}_{conv}^{1 \times 1}$ squeezes the channels with $d_C/4$ filters to one quarter of the input. Thus, the subsequent operations have greatly reduced filter maps and computations. Borrowing from *SqueezeNet* [61], the squeezed tensor is processed in two streams, once with a $\mathbf{o}_{conv}^{1 \times 1}$ and once with a $\mathbf{o}_{conv}^{3 \times 3}$, and each is expanding the channels to $d_C/2$. The concatenation of both streams forms the output of the *FireModule* (Fig. 3.14b) that replaces the standard convolution operation in the encoding block (Fig. 3.14a) and reduces the computational costs significantly. The squeezed fusion network’s

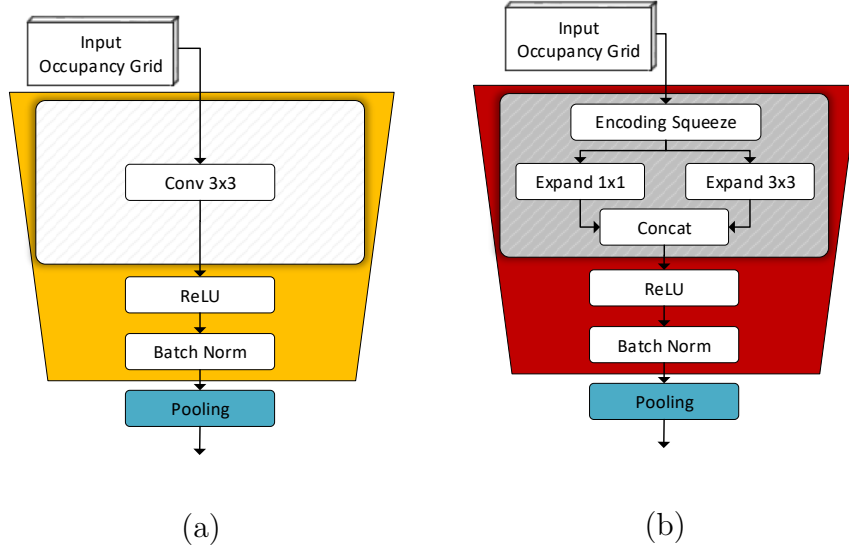


Figure 3.14: Block diagram of the encoding blocks in the AE-FN and AEB-FN architectures. (a) Initial and (b) optimized version of the encoding block.

(AESQ-FN) encoding block is based on the adjacency matrix

$$\mathbf{A}_{enc}^{SQ} = \begin{bmatrix} \mathbf{o}_{conv}^{1 \times 1} & & & & & & & & \\ & \mathbf{o}_{conv}^{1 \times 1} & & & & & & & \\ & & \mathbf{o}_{conv}^{3 \times 3} & & & & & & \\ & & & \mathbf{o}_{relu} & & & & & \\ & & & & \mathbf{o}_{max}^{2 \times 2} & & & & \\ & & & & & \mathbf{o}_{bn} & & & \\ & & & & & & & & \end{bmatrix}. \quad (3.26)$$

Analogically, the compressed encoding block is formed by replacing the plain deconvolution \mathbf{o}_{deconv} (Fig. 3.15a) by a *FireDeconv* module from *SqueezeSeg* [141] (Fig. 3.15b). Following the paradigm of the *FireModule*, first a 1×1 convolution reduces the feature channels, before the actual neural operation is performed. The spatial downsampling in form of the deconvolution is placed before the two expanding operations $\mathbf{o}_{conv}^{1 \times 1}$, $\mathbf{o}_{conv}^{3 \times 3}$ —now with a squeezed input tensor. The squeezed fusion network’s decoding block is based on the adjacency matrix

$$\mathbf{A}_{dec}^{SQ} = \begin{bmatrix} \mathbf{o}_{conv}^{1 \times 1} & & & & & & & & \\ & \mathbf{o}_{deconv}^{3 \times 3} & & & & & & & \\ & & \mathbf{o}_{conv}^{1 \times 1} & & & & & & \\ & & & \mathbf{o}_{conv}^{3 \times 3} & & & & & \\ & & & & \mathbf{o}_{relu} & & & & \\ & & & & & \mathbf{o}_{bn} & & & \\ & & & & & & & & \end{bmatrix}. \quad (3.27)$$

3 Neural Networks for Sensor Fusion

With the compressed encoding and decoding blocks (\mathcal{B}_{enc}^{SQ} , \mathcal{B}_{dec}^{SQ}), the hardware-aware grid fusion networks¹ are formed around the same macro-architecture as AEB-FN. Consequently, also the AESQ-FN incorporate bypassed connections, which lead to a similar fusion performance as the AEB-FN.

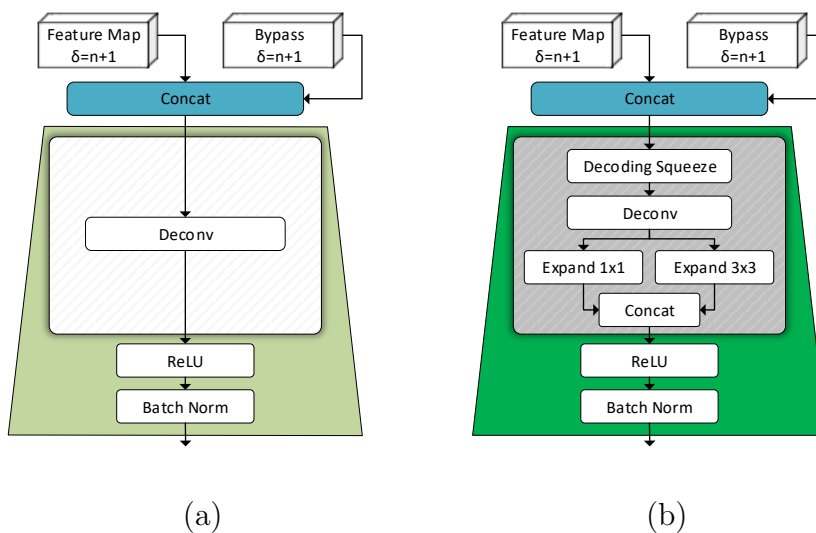


Figure 3.15: (a) Initial and (b) optimized version of the decoding block.

Table 3.10: Estimated inference times on Synopsys EV61 processor with different CNN-engine configurations at 500 MHz operating frequency for the optimized fusion architectures AESQ-FN

Network	Grid	EV61 [ms]		
		CNN 1	CNN 2	CNN 4
AESQ-5-16	256	0.82	0.41	0.20
AESQ-5-16	512	3.27	1.64	0.82
AESQ-5-16	1024	13.1	6.55	3.27

Results of Compressed Network Architectures

A reason to optimize the initial grid fusion networks (Section 3.3) is the missing scalability of input grid sizes and resolutions for real time application. The optimized fusion architectures AESQ-FN show significant improvements of the computational

¹Results published in [9].

costs at a cost of minimal degradation of the fusion quality. Thus, with the introduction of new, compact encoding and decoding blocks, the AESQ-FN architectures allow the fusion of either higher resolution ($\delta_x < 0.25$ m), or a larger coverage area of the occupancy grids (> 64 m \times 64 m). For input sizes up to 1024×1024 , one forward pass takes less than the required 20 ms on the Synopsys EV61 processors with any CNN-engine configuration (Table 3.10).

Comparing the estimated inference times of AEB-5-16 (Table 3.9) and AESQ-5-16, there is a relative speedup of 80.6%, or 50.4 ms in absolute difference, respectively. Even though the computations are eased significantly to a level of applicability, the quality metrics are only slightly decreasing, when compared to AEB-5-16 (Fig. 3.16). For the same network configuration ($\delta = 5$, $\chi = 16$), the relative change of the evaluation metrics from AEB to AESQ are as follows:

- m_{pAcc} -2.9%: (AEB-5-16 = 92.5%, AESQ-5-16 = 89.9%)
- m_{mAcc} -7.7%: (AEB-5-16 = 85.2%, AESQ-5-16 = 79.1%)
- m_{IoU} -5.1%: (AEB-5-16 = 86.4%, AESQ-5-16 = 82.2%)
- m_{fIoU} -2.9%: (AEB-5-16 = 91.4%, AESQ-5-16 = 88.8%)

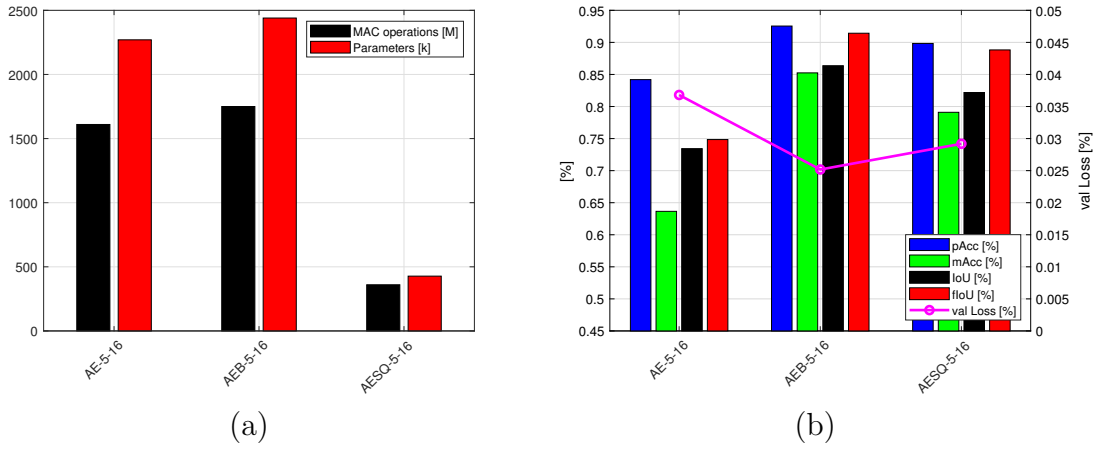


Figure 3.16: Comparison of the grid fusion network families, each with $\delta = 5$ and $\chi = 16$, including the optimized AESQ-FN. (a) The computational costs and (b) the fusion quality shown by the evaluation metrics from Section 3.4.1.

3.5 Conclusion

In this chapter, a novel method for fusing occupancy grids based on a convolutional auto-encoder architecture is introduced.

3 Neural Networks for Sensor Fusion

Current grid fusion algorithms rely on cell-wise independent Bayesian computations that require either information-rich inputs, such as raw sensor signals, or temporal integration to a map, in order to produce an environmental model suitable for ADAS functions of Level 2/2+.

As occupancy grids are two-dimensional matrices with features in the third dimension, they can be processed well with CNNs from computer vision tasks. Consequently, the convolutional auto-encoder offers the basic architecture for the grid fusion networks, which extend the standard one-to-one auto-encoders to a multiple-to-one paradigm.

Condensed from various architectural explorations, three families of grid fusion networks are proposed: The plain auto-encoder fusion network AE-FN, which is able to fuse occupancy grids and produce a free-space estimate, but has the issue of blurring small details. With the introduction of skipped connections, deep parts of the AE are bypassed, thus, small details propagate easier through the AEB-FN. However, these two architecture families rely on heavy computations and may not be deployed on resource-constrained embedded accelerators for real-time applications. The proposed squeezed version AESQ-FN employs *FireModules* and *DeconvFire* modules from the *SqueezeNets* and achieves a reduction in required computations by roughly 80% to the price of slightly lower fusion quality.

The training of the grid fusion network is fundamental for their working principle: During training, the networks learn to map from feature-level inputs to the ground truth, which is calculated from Lidar data. Thus, the training teaches the networks to leverage *low-quality* inputs to a *high-quality* environmental model.

Conclusively, the following main aspects of the grid fusion networks are achieved in this chapter:

- Fusion of multiple two-dimensional occupancy grids into one combined environmental model instantaneously, without the need to map over time to retrieve a free-space estimate.
- Based on the training with Lidar-based ground truth, the networks learn to map from cheap sensors to a Lidar-like output, which can be used for ADAS functions of Level 2/2+ cars.
- Compression of the model size and computations to a level, where real-time application is possible, even on resource-constrained, automotive microcontrollers.

4 Multi-objective Model Optimization with Neural Architecture Search

Taking a look at the architecture of the majority of classification and segmentation networks, including the grid fusion networks of this work, the macro-architecture is similar among networks with comparable tasks. Classification networks are formed usually by concatenating normal and reduction blocks linearly in order to reduce the input image to an embedding, which is then passed to a fully connected classification layer. Segmentation networks, on the other hand, usually have a symmetrical decoding of the embeddings, and may include skipped connections. Hence, the network models are differentiable mainly based on the chosen micro-architecture inside of the blocks—excluding the impact of training configurations by assuming similar setups.

After *AlexNet*'s initial success over traditional computer vision algorithms in 2012, *ResNet* made the next groundbreaking step in 2015 by introducing residual functions inside of a block¹ that allowed the successful training of substantially deeper networks [50, 75]. The initial *ResNet*-block's micro-architecture is basically two weight layers combined with a skipped connection from the block's input (Fig. 3.4, 4.1a). A refined variant of the initial blocks is used in *ResNet-1001*, where the gradient flow is ensured through enhanced shortcut connections (Fig. 4.1b) [51]. Further improvements of these blocks in *ResNeXt* are introducing multiple, parallel computational flows inside of the block with complicated interconnections (Fig. 4.1c) [143].

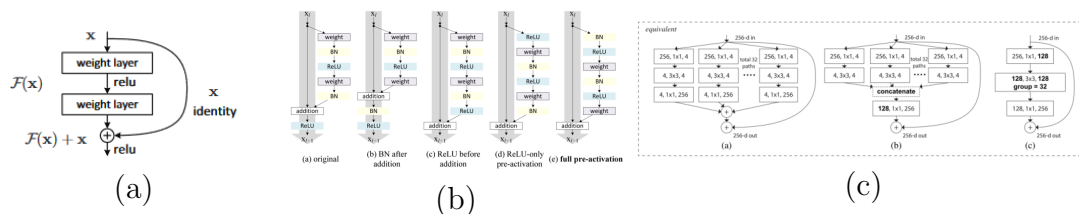


Figure 4.1: Different variants of *ResNet*-blocks with increasing complexity. (a) Initial *ResNet* [50], (b) *ResNet-1001* [51] and (c) *ResNeXt* blocks [143]. Images taken from the respective publications.

¹In *ResNet*, the blocks are denoted as cells.

4 Multi-objective Model Optimization with Neural Architecture Search

In general, the trend among network architectures was to create ever more sophisticated micro-architectures in order to push the results on the well-known classification benchmarks. Roughly until 2018, most state-of-the-art neural network models for image classification were designed manually. Figure 4.2 shows the history of the highest scoring networks on the *CIFAR-100* and *ImageNet* challenges. The first network to surpass handcrafted models on *ImageNet* was *NASNet* in 2017 [149], and the first to outperform on *CIFAR-100* was *EfficientNet* in 2019 [127].

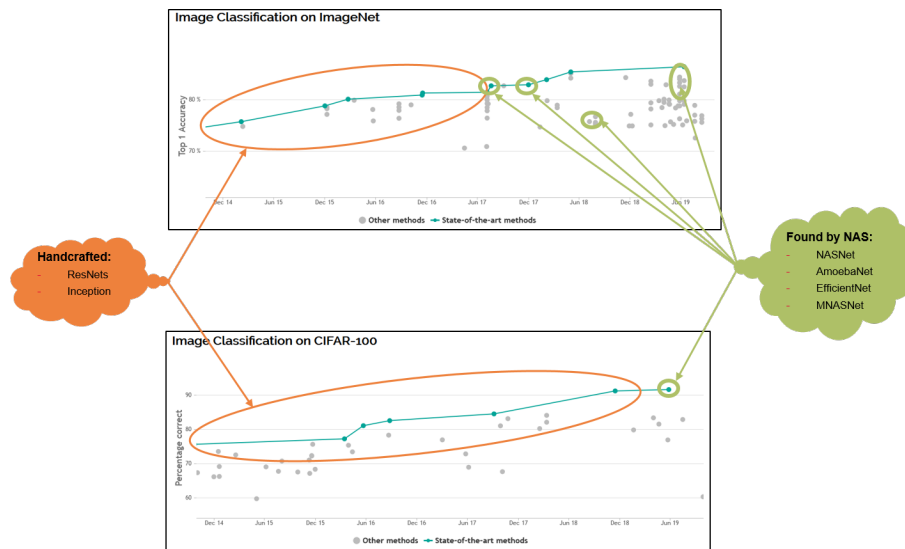


Figure 4.2: The evolution of state-of-the-art network models for image classification. NAS-based classification models surpass handcrafted architectures on *ImageNet* [30] since 2017 and on *CIFAR-100* [74] since 2019. Data and visualization from <https://paperswithcode.com/>.

As the neural architecture search strategies are proven to work well, the grid fusion task from Chapter 3 is investigated with an automated approach in this chapter. First, an overview about current NAS algorithms is laid out, starting with a literature review of general NAS projects for image classification and segmentation, followed by an analysis of how well the different NAS approaches perform. The best-fitting NAS algorithm is then applied to the grid fusion task by introducing the task-specific macro-architectural requirements, the micro-architectural search space and further details about the search process. Finally, an evaluation shows the progress and the results of the architecture search for an optimal network. The optimization is conducted with regards to specific accelerator hardware, on which the models are to be inferred.

4.1 State-of-the-Art Neural Architecture Search Algorithms and Embedded CNN Accelerators

This section gives an overview of existing methods and trends in the area of neural architecture search. Soon after initial successes in classical image classification tasks, NAS algorithms covered segmentation tasks. The most recent trend is multi-objective optimization of neural networks for embedded deployment on constrained edge devices. Hence, a suite of state-of-the-art accelerator hardware is outlined and the chosen devices are compared to each other regarding their performance and limitations towards the network models. While the majority of published works is about network topologies, NAS is also used for enhancing other tasks, such as searching for superior optimizers [15].

4.1.1 NAS Algorithms in Literature

NAS for Image Classification

The first leap towards automated model generation was done with *Neural Architecture Search v1-v3* by designing a controller that is based on reinforcement learning (RL) [148]. Generating model descriptions of classification CNNs for the CIFAR-10 dataset, it finally achieved a state-of-the-art error rate of 3.65%. *Block-QNN* laid the foundation for the block-wise search with the first definition of the macro-architecture for image classification in 2017 [147]. Two skeletons that are populated with the searched block and intermediate pooling layers, were presented—one for the CIFAR-10 and another for the ImageNet dataset. This approach is followed and adopted widely by the subsequent works about neural architecture search. While *Block-QNN* already outperformed many handcrafted state-of-the-art classification networks at that time and also *Neural Architecture Search v1-v3* with a CIFAR-10 error rate of 3.60%, *NASNet* further pushed the limits down to 2.4% [149]. The key contribution of *NASNet* was the introduction of two different types of blocks (normal and reduction cells), which allowed transferability to other problems. *NASNet* exploited this by searching for best-performing cell architectures on CIFAR-10 and then plugging the found micro-architectures into the ImageNet skeleton.

NAS for Semantic Segmentation

Whereas image classification has significantly improved through automated machine learning algorithms, such as neural architecture search, image segmentation, or in general semantic segmentation tasks got less attention. Using NAS to find optimal architectures for image segmentation is mostly applied for medical image analysis. *SCNAS* segments 3D medical images with an architecture similar to U-Net, but with searchable *encoding*, *encoding-normal*, *decoding* and *decoding-normal* blocks [71].

4 Multi-objective Model Optimization with Neural Architecture Search

NAS-Unet concentrates the search on only two lightweight cell types (*DownSC*, *UpSC*) [137]. It replaces standard skip connections with `cweight` operations known from *Squeeze-and-Excitation* networks [57]. In *Auto-DeepLab*, cell-level and network-level architectures for general image segmentation are jointly searched [79]. Disparity estimation is improved by *AutoDispNet* with an auto-encoder structure of *encoding*, *decoding* and *normal* cells [116].

Recent works on NAS focus on awareness to platform-specific memory footprint and accelerator topology. Aiming for mobile deployment, *MnasNet* conducts a multi-objective optimization in order to find multiple Pareto-optimal network architectures regarding accuracy and inference time [126]. Similar to *MnasNet*, *MobileNetV3* focuses on constrained devices and is being discovered by a hardware-aware neural architecture search, but it also proposes a segmentation head for pixel level predictions [56].

Comparison of State-of-the-Art Neural Architecture Search Algorithms

Each neural architecture search method is prominently defined by its search space and search strategy. While the search space is framed as a discrete space of possible architectures and confined set of operations in the majority of published works, there are multiple promising approaches for the search strategy [34]. Among the manifold search algorithms, such as Bayesian optimization, gradient-based methods or even random search, the two most prominent strategies are approaches based on the evolutionary algorithm (EA) [88, 108, 109] and reinforcement learning (RL) [15, 147, 149].

In a comparative study of Real et al., these two search strategies are analyzed and compared to random search [108]. Both approaches consistently outperform the random search equally in terms of their final test accuracy on the CIFAR-10 dataset. Still, in this case study, EA converges faster to its final architecture, compared to RL. Also, EA generates smaller candidate models than RL. Thus, building on the results of [108], the search strategy of this work is chosen to be based on the evolutionary algorithm.

4.1.2 State-of-the-Art Hardware Accelerators

In Section 3.4.2, it has been derived, which of the grid fusion models are lightweight and small enough in order to be inferred on a dedicated CNN accelerator hardware. In that analysis, the three different grid fusion networks with varying input grid sizes have been investigated, and the inference speed on the target hardware—the Embedded Vision Processor family of Synopsys’ DesignWare IP—has been estimated. For the further analysis and deployments, a set of commercially available accelerators has been chosen from the embedded accelerator market (Fig. 4.3).

With the rising demand for mobile applications, this market is emerging quickly [111].

4.1 State-of-the-Art Neural Architecture Search Algorithms and Embedded CNN Accelerators

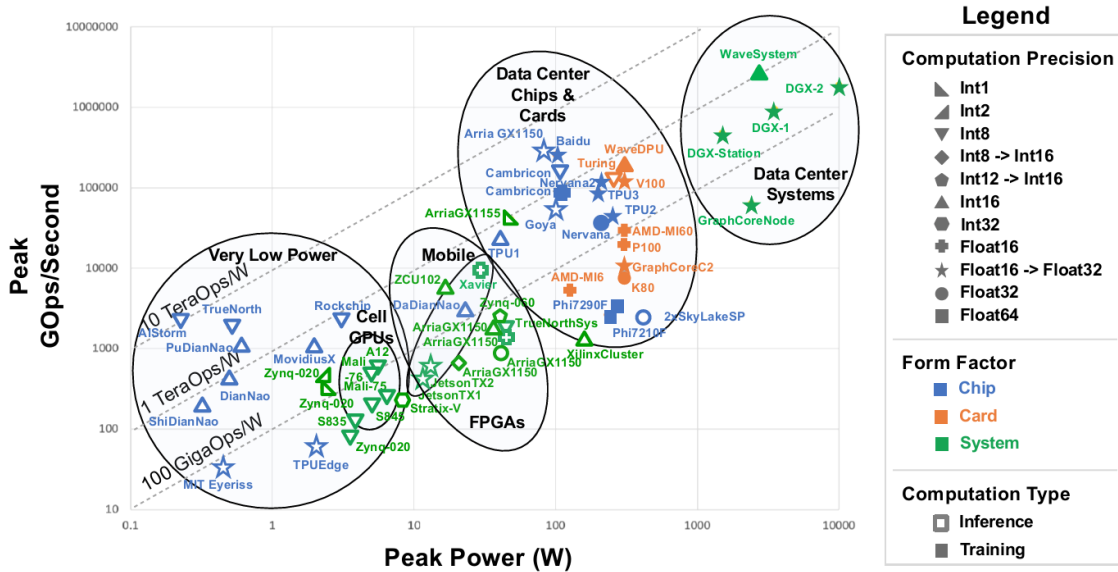


Figure 4.3: The AI accelerator market overview with the announced accelerators and processors shown in a performance vs. power consumption trade-off. Image taken from [111].

Mainly research teams and universities offer chips in the regime of low-power devices, such as the MIT Eyeriss chip [21], the Intel MovidiusX processor [62], Google’s EdgeTPU [44], the DianNao accelerator family [22] or the Rockchip RK3399Pro [112]. Out of this portfolio, only the Google EdgeTPU, the NVIDIA Jetson Nano and the Intel MovidiusX, used in the Neural Compute Stick 2 (NCS2), are commercially available for a competitive price in 2020. These three embedded CNN inference accelerators are used for further evaluations.

Whereas they are the most developed off-the-shelf evaluation kits, their firmware is still in an experimental phase and under constant development. In order to use the accelerators, the network model constraints of each hardware are to be met. In Table 4.1 the supported operations of each AI accelerator are shown, and it becomes obvious that for deployment on the *EdgeTPU*, strict constraints in the network architecture have to be met. Many state-of-the-art network topologies that claim to aim for edge deployment, lack the applicability to these low-power edge devices.

Table 4.1: Model requirements for low-power neural accelerators. 3D: Conv3D, RNN: Recurrent operations (RNN, LSTM, GRU), BN: Batch normalization, DO: Dropout, Basic OPs: Conv2D, ReLU and softmax activation, pooling and concat.

Edge-HW	3D	RNN	BN	DO	Basic
Google <i>EdgeTPU</i> [44]	-	-	-	-	✓
Intel <i>NCS2</i> [62]	-	-	✓	✓	✓
NVIDIA <i>Jetson nano</i> [97]	✓	✓	✓	✓	✓

4.2 Application of NAS Method to Sensor Fusion Networks

As introduced in Chapter 3.3.1, grid fusion networks are constructed out of repetitive, interconnected building blocks. Each building block \mathcal{B} inhibits inputs, multiple operations and one output, all together described by a directed acyclic graph \mathcal{G} and a set of operations \mathcal{O} (Definition 3.3.3).

A NAS system is usually built upon three dimensions (search space, search strategy, performance estimation strategy) [34]. In this Chapter, \mathcal{O} and \mathcal{G} are interpreted as the search space, which is outlined first. Then, the search strategy is chosen according the comparative analysis shown in Section 4.1.1, and its details, such as the genetic diversity in the EA, are discussed. Finally, the performance estimation strategy in form of a multi-objective reward function, is presented.

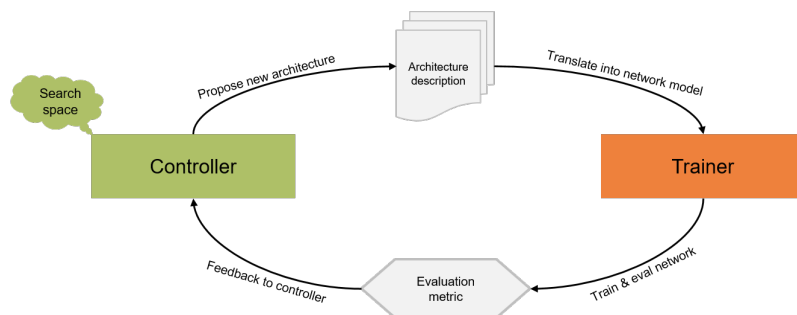


Figure 4.4: NAS principle of iteratively optimizing networks to a given task.

The system is designed to perform an architecture search for a grid fusion problem with multiple occupancy grids as inputs and one OG as the output. The number of input branches is set to two throughout this analysis, but the NAS can be configured for processing more inputs.

4.2 Application of NAS Method to Sensor Fusion Networks

The key contributions of this NAS approach are published in [11] as a conference paper¹ and can be summarized as follows:

- *A NAS framework that is configurable for arbitrary input branches and image dimensions:* This flexible design allows to search for network architectures different from standard classification or segmentation networks, such as the proposed grid fusion networks.
- *An extended search space for operations within a block with an adaptive strategy for reshaping tensors in-between the blocks of the macro-architecture:* The *NASNet* search space is extended to 13 operations and a correct inter-block data-flow is ensured.
- *A family of grid fusion networks optimized for embedded implementation:* The model sizes of the **GridFuN** networks range from 90 KiB to 4 MiB. The fusion performance increases proportional to the memory footprint.

4.2.1 Macro-architectural Presets

Given the analysis of Chapter 3.4, the auto-encoder with bypass is the base for the macro-architecture of this neural architecture search. In order to frame the problem, two types of search cells (encoding block \mathcal{B}_{enc} , decoding block \mathcal{B}_{dec}) are needed for an auto-encoder architecture. Each encoding block takes two inputs, the decoding blocks take up to three inputs, and each block outputs one tensor (Figure 4.5), following the configuration of [71]. The predefined connections and bypasses describe the actual macro-architecture of the grid fusion architectures that are to be found by this search. The cell-wise approach with a fixed macro-architecture known from [149] was adapted and extended to a framework that can be configured to arbitrary network macro-architectures. The connections of the blocks are hard-wired and do not change throughout the search. Only the micro-architecture is variable, thus, each macro-architecture incorporates a vast amount of variants, depending on the block-architectures. In the following, the family of these networks is denoted as **GridFuN (Grid Fusion Networks)**.

As in the handcrafted fusion networks in Chapter 3, **GridFuN** has various architectural parameters that influence the network performance significantly. Beside the already known χ and δ , a NAS-specific parameter is introduced in the following: the number of nodes per block K . The three parameters are set fix before the search algorithm starts.

¹Paper [11]: IEEE ICMLA 2020 (Best Paper Award).

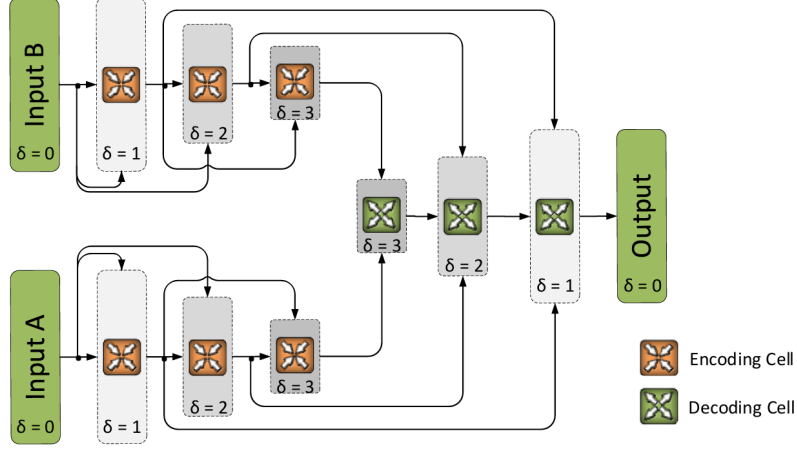


Figure 4.5: Macro-architectural concept of the neural architecture search for grid fusion networks. Only depth $\delta = 3$ is shown for visualization.

4.2.2 Micro-architectural Search Space

The actual search is performed on the micro-architecture level within the cells. In order to formalize the search, the directed acyclic graph \mathcal{G} is constrained in some aspects. K denotes the number of intermediate nodes within a block, thus, together with the two input nodes, it determines the size of the adjacency matrix \mathbf{A} . Each row of \mathbf{A} denotes an intermediate tensor, and each column describes which input tensors those intermediate tensors are calculated from. More precisely, each entry in a column of \mathbf{A} is combined with a combination operation, forming a node \mathcal{N} (Definition 3.3.4).

In order to formalize the search process, the nodes are set to have two inputs ($I_1: \mathbf{T}_A, I_2: \mathbf{T}_B$), each with an own operation $\mathbf{o}_1, \mathbf{o}_2$, and a combination operation \mathbf{o}_+ : $\mathcal{N}_+(\mathbf{o}_1(\mathbf{T}_A), \mathbf{o}_2(\mathbf{T}_B))$. Each node is indicated in \mathcal{G} , but during the search it is described by a 5-tupel $(I_1, I_2, \mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_+)$. In encoding blocks, the block input tensors I_1, I_2 are downsampled in image dimensions by 2 with strides $S = 2$. In decoding blocks, the image dimensions are kept constant ($S = 1$), and only after the last combination operation of the block, the image is upsampled with an upsampling operation \mathbf{o}_{up} .

There are three sets of operations: normal operations \mathcal{O} , combination operations \mathcal{O}_+ and upsampling operations \mathcal{O}_{up} . Reduction operations are part of normal operations, but with strides $S = 2$, where applicable. The set of possible normal operations \mathcal{O} out of which the 5-tupel can be populated is inspired by the work of [149], but extended to $|\mathcal{O}| = 13$ operations. Together with the combinations \mathcal{O}_+ with $|\mathcal{O}_+| = 2$ and upsampling OPs \mathcal{O}_{up} with $|\mathcal{O}_{up}| = 4$ operations, the total amount of searchable OPs is 19 (Table 4.2).

Hence, in a search for blocks with $K = 3$ nodes per block, $|\mathcal{O}|^{2K} + |\mathcal{O}_+|^K = 4,826,817$

4.2 Application of NAS Method to Sensor Fusion Networks

Table 4.2: The three sets of operations in the search space of the **GridFuN** framework. The last column indicates, whether the given operation changes the spatial dimensions: Equal (=), downsampling by two (\downarrow), upsampling by two (\uparrow).

OP Set	Name	Notation	Dim.
Normal \mathcal{O}			
	Standard convolution	$\mathbf{o}_{conv}^{1 \times 1}, \mathbf{o}_{conv}^{3 \times 3}, \mathbf{o}_{conv}^{5 \times 5}$	= or \downarrow
	Depth-wise separable conv.	$\mathbf{o}_{sep}^{1 \times 1}, \mathbf{o}_{sep}^{3 \times 3}, \mathbf{o}_{sep}^{5 \times 5}$	= or \downarrow
	Dilated convolution	$\mathbf{o}_{dil}^{1 \times 1}, \mathbf{o}_{dil}^{3 \times 3}, \mathbf{o}_{dil}^{5 \times 5}$	= or \downarrow
	Skip / bypass / residual	\mathbf{o}_{skip}	=
	Max pooling	$\mathbf{o}_{max}^{3 \times 3}$	= or \downarrow
	Average pooling	$\mathbf{o}_{avg}^{3 \times 3}$	= or \downarrow
	<i>FireModule</i>	\mathbf{o}_{fire}	=
Combination \mathcal{O}_+			
	Element-wise addition	\mathbf{o}_{add}	=
	Feature concatenation	\mathbf{o}_{cat}	=
Upsampling \mathcal{O}_{up}			
	Transposed convolution	$\mathbf{o}_{t_conv}^{1 \times 1}, \mathbf{o}_{t_conv}^{3 \times 3}, \mathbf{o}_{t_conv}^{5 \times 5}$	\uparrow
	Bilinear interpolation	\mathbf{o}_{interp}	\uparrow

different encoding blocks can exist. Additionally, $(|\mathcal{O}| + |\mathcal{O}_{up}|)^{2K} + |\mathcal{O}_+|^K = 24,137,577$ individual variants for the decoding block are possible. Further, the manifold increases drastically, when the number of filters in the convolutional layers are also included. The initial value for the number of filters is set to $\chi = 4$, and it will evolve over time. A special reward for the multi-objective optimization (Section 4.2.3) counteracts the unlimited growth of filters (Figure 4.11).

Figure 4.6 shows the random initialization of one node within a block. On the left side, a unpopulated raw node structure is depicted with unknown node inputs and unknown node operations. In the center of the figure, the controller samples random inputs to the node from the possible nodes connections, which are the two input nodes and the nodes before. Exemplary, the possible inputs to \mathcal{N}_5 are: $\mathcal{N}_0^{\text{In}}, \mathcal{N}_1^{\text{In}}, \mathcal{N}_2, \mathcal{N}_3$ and \mathcal{N}_4 . Next, the controller randomly assigns operations from the search space to the two node inputs, thus fully determining the node. During the search process, beside the random initialization of the complete node, also mutations to singular parts of the node are applied. There, for example, the controller changes only the assignment of one node input to another, or one operation to an other.

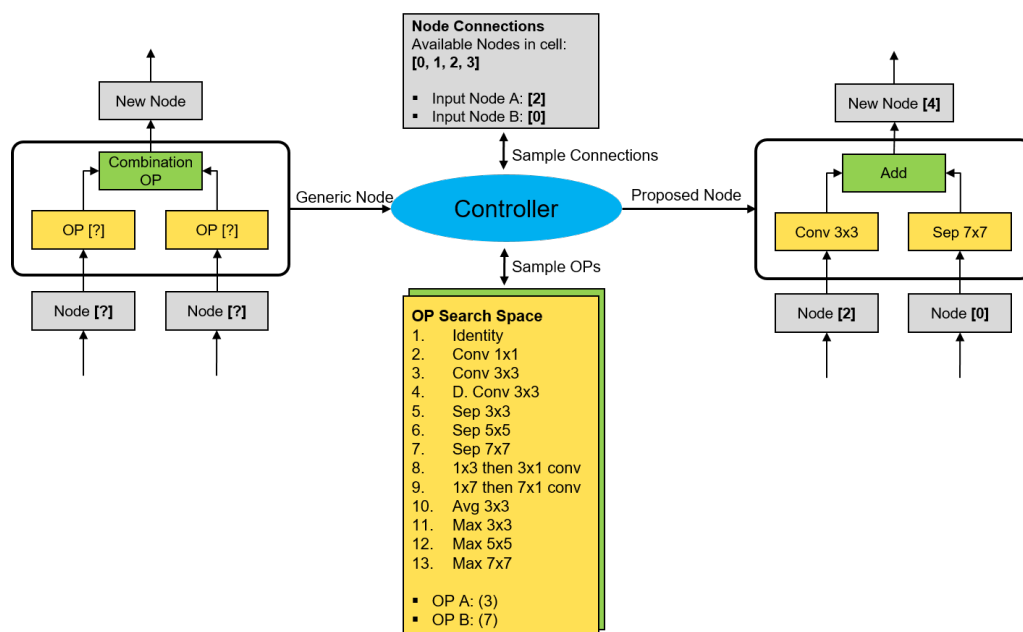


Figure 4.6: The initialization of nodes by the controller part of the NAS. A new, default node is populated randomly by sampling connections and operations of the search space.

4.2.3 NAS Method Details

In this section, important aspects of the NAS system at hand are explained. First, the search algorithm and the generation of diverse micro-architectures out of the model’s genotype is shown. Then, the training environment for a single model evaluation is outlined, laying the foundation to one iteration within the evolutionary algorithm. Next, considerations about the reshaping of tensors in-between blocks of different depth are mentioned, along with the mutations of the network genotypes, which are crucial for genetic diversity throughout the search. Finally, the working principle of the multi-objective optimization is described.

Search Algorithm

The proposed NAS framework is structured into a controller that coordinates network creation and a trainer that evaluates network performances (Figure 4.4). The network database, from which the controller coordinates the evolution process, stores only the network genotypes and their evaluation metrics. Thus, during the search process, no trained network is stored to the hard disk and a large volume of networks can be covered.

In the genetic algorithm, each network architecture is described by its genotype that defines the computational graph with a directed acyclic graph and additionally

4.2 Application of NAS Method to Sensor Fusion Networks

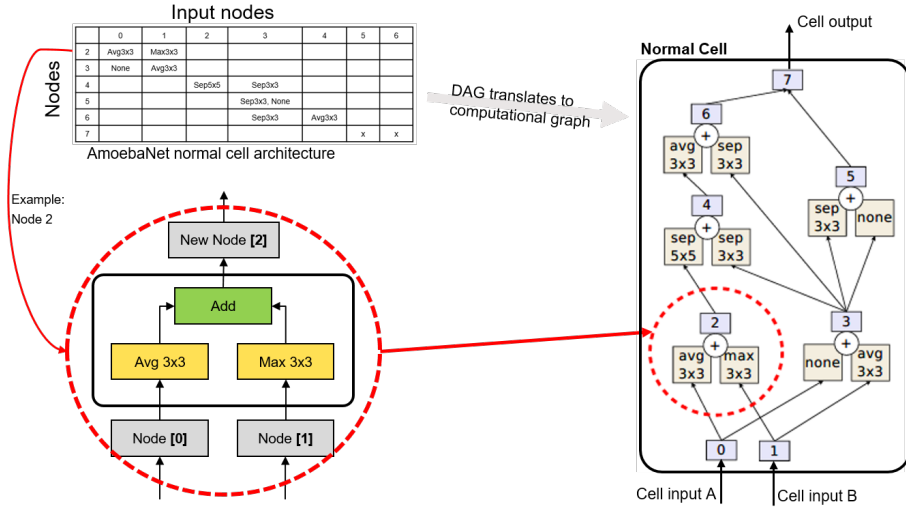


Figure 4.7: Translation of adjacency matrix to the computational graph of *AmoebaNet* (Picture of computational graph taken from [108]).

stores all the necessary hyperparameters. Basically, the DAG is handled as a matrix similar to the one shown in Figure 4.7, where each row of the adjacency matrix is describing one node. In the example of the figure, the *normal cell* of *AmoebaNet* [108] has 7 nodes, two of which are cell input nodes. Thus, adjacency matrix has 5 rows and 7 columns. As the final step of the translation from DAG to computational graph, the yet unused¹ nodes are combined with a last combination to form the cell output.

Before a NAS experiment can be started, first, the NAS parameters need to be defined. Namely, they are:

- The number of generations g , which indicates, how many rounds of subsequent evaluations/mutations are performed.
- The population size p , which indicates, how many models from the database are investigated in parallel.
- The number of epochs e of training for each model evaluation.
- The number of nodes per block K .
- The number of initial filters χ .

Additionally, for each search the dataset \mathcal{D} has to be specified, and if the result is planned to be platform-specific, the target hardware, too. Once the system is set up, the iterative search process works as follows. In Algorithm 2, the procedure of one NAS experiment is described.

¹Unused nodes are those that are not consumed by other nodes as an input within one block.

Algorithm 2 The framework of GridFuN.

Input: g generations, e epochs, p population size, K nodes per cell, χ initial filters, dataset \mathcal{D} , embedded hardware H .

Output: GridFuN architecture family

$P = \text{initialize}(p, K, f)$

for $i = 1$ **to** g **do**

$P' = \text{sample}(s, P)$

$P' = \text{mutate}(P')$

$P' = \text{train}(P', \mathcal{D})$

$P' = \text{evaluate}(P', H)$

$P = P \cup P'$

end for

The total population P starts off with p randomly initiated network models. The genotype of each model consists of two DAG, one for the encoding block and one for the decoding block. For each column of the upper right triangular matrices (\mathbf{A}_{enc} , \mathbf{A}_{dec}) up to two entries are chosen randomly from the set of possible operations. This defines the operations in the nodes within the blocks. The initiation phase is completed, once all networks of the initial population are evaluated and scored with the fitness value.

Next, out of the total population P , a set of p competitor networks are selected according to their fitness value. This selection incorporates the best performing networks, and they are now modified with mutations. These mutations introduce slight changes to the genotype of the network model, thus explore the search space in vicinity of the parent network. In order to more aggressively traverse through the search space, up to three simultaneous mutations are made to a genotype within one generation (Algorithm 3). Multiple instances of checks verify the validity of the novel, mutated genotype, and also, whether this new configuration has already been evaluated in the process before. If the proposed configuration already exists, the mutation process is repeated, starting with the network model of the beginning of this generation.

Once the selection of p networks is mutated successfully, it is trained by the trainer. The trainer incorporates the complete training procedure and reports only the evaluation metrics to the controller. These include the fusion quality, but also the hardware-specific metrics of inference time and memory footprint. The process of receiving the hardware-specific metrics is first converting the trained network model to a format that is compatible with the accelerator and then inferring a series of forward passes. The average inference time of 100 runs together with the memory footprint of the accelerator-specific model format conclude the deployment, and are the result of the evaluation process.

Finally, the results of this generation P' united with the total population P form

4.2 Application of NAS Method to Sensor Fusion Networks

the total population of the next generation. After g generations, a range of networks has been evaluated and the results stored in the network database. The Pareto-optimal solutions out of this manifold form the family of GridFuN networks.

Network Training Environment

The macro-architecture during the experiments is set to an encoding depth of $\delta = 4$, so that the lowest dimensions of the tensors are $\mathbf{T}^\delta \in \mathbb{R}^{d_x/16 \times d_y/16 \times 16 \times 16 \chi}$. Two experiments are conducted with blocks having $K = 2$ and $K = 3$ intermediate nodes with initially $\chi = 8$ feature channels in convolutional layers.

Similar to the training of the FC-FN, AE-FN and AEB-FN in Chapter 3.3.4, the training is done on the data derived from the *nuScenes* dataset. Here, the augmented dataset is denoted as \mathfrak{D} , incorporating the full 44,760 frames, with each frame consisting of the three OGs (x_{OG}^{rad} , x_{OG}^{cam} , x_{OG}^{gt}). In order to reduce the evaluation latency, a subset of the full dataset is used during the search process. A subset $\mathfrak{D}_{light} \subset \mathfrak{D}$ consists of 5,632 frames based on the first 100 scenes of *nuScenes*. Further latency reduction is gained by only partially training the networks on the \mathfrak{D}_{light} dataset for 2 epochs. The best performing architectures of the search are then retrained on the complete \mathfrak{D} dataset to display their full performance. The full training cycle consists of 10 epochs in contrast to the reduced training setup during the search process. Empirically, the final performances of the networks are proportional to the performances of the partially trained networks. From these observations we can derive that $f(a') > f(b') \rightarrow f(a) > f(b)$ holds for fully trained networks a, b , partially trained networks a', b' , and their network performance $f(\cdot)$ ¹. This way, the return time of an average network evaluation is reduced down to about 10 minutes, instead of several hours on the given hardware setup.

All experiments are conducted relying on Tensorflow 1.13.1 [3] with compiler version 5.4.0, and the training is performed on an NVIDIA Titan V GPU under CUDA 10.0.130. The training was done on the \mathfrak{D}_{light} dataset at a split of 80% training samples and 20% test samples. The evaluation metrics are averaged over 10 batches with a batch size of 16. The learning rate α is decreasing from $\alpha_0 = 0.0005$ to zero according to a cosine decay (Eq. 4.1) with a warm-up of 2.5% and a hold until 10% of the total iterations i_T , according to

$$\alpha(i) = \begin{cases} \alpha_0 \frac{i}{0.025 i_T}, & i < 0.025 i_T, \\ \alpha_0, & 0.025 i_T \leq i \leq 0.1 i_T, \\ \alpha_0 \cos\left(\frac{2\pi}{i_T} i\right) & i > 0.1 i_T, \end{cases} \quad (4.1)$$

where i is the current iteration. This learning rate scheduling ensures a smooth convergence of the network performance in contrast to methods that have $\alpha > 0$ at the end of the optimization. Particularly for the partially trained networks this is

¹The fitness value f is further explained in the paragraph *Multi-Objective Optimization* below.

4 Multi-objective Model Optimization with Neural Architecture Search

important, because the local minimum of the loss function is not reached perfectly, thus, any learning rate larger than zero would cause an unstable optimization result.

The training during the search process of the NAS is done for 2 epochs. The full training for the comparisons is done for 10 epochs, and it is sufficient for reaching the convergence. The ADAM optimizer is used with default parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$ to minimize the pixel-wise softmax classification loss.

Reshaping Strategies

At some configurations of cells, the dimensions of the inputs to a combination operation \mathbf{o}_+ do not fit, but subsequent combination operations require equal dimensions to work correctly. Thus, adaptive reshaping is necessary to construct a valid network model. Reshaping involves image dimension adjustment and feature channel adjustment.

Image dimensions have to be reshaped, whenever an encoding block \mathcal{B}_{enc}^δ is bypassed and the bypassing tensor $\mathbf{T}^{\delta-1}$ has different dimensions compared to the output of this cell. The correction follows through additional adjustment operations $\mathbf{o}_{adj} \in \mathcal{O}_{adj}$. The set of \mathcal{O}_{adj} consists of convolution and pooling layers with strides $S = 2$ for downsampling.

A change in the number of feature channels F is needed, whenever operations are used that do not change the number of feature channels themselves, such as normal pooling operations. The F -correction can be done with simple point-wise convolutions $\mathbf{o}_{conv}^{1 \times 1}$ or by changing the combination operation from \mathbf{o}_{add} to \mathbf{o}_{cat} , where the tensors are concatenated along the channels, thus, forming a valid model.

Following the work of *FishNet* [123], the two strategies for image size adjustment are explored during the search. Using convolutions with strides for image dimension reduction early in the network significantly reduces the amount of computations needed in subsequent layers and has limited loss of information when compared to loss-prone pooling. In contrast, using pooling layers is beneficial in reshaping skipped connections, where no additional weights have to be learned, thus, gradient propagation is eased, when compared to additional weight layers.

Multi-Objective Optimization

In order to fit networks onto strictly constrained embedded hardware, optimizing the networks only for accuracy metrics is not enough. The model size has to be shrunk and the network architecture chosen according to the capabilities of the hardware accelerators. The NAS boils down to a multi-objective optimization with constraints dependent of the specific embedded hardware at hand.

The grid fusion quality is evaluated with performance metrics known from Chapter 3.4.1. The four quality metrics (pixel accuracy m_{pAcc} , mean accuracy m_{mAcc} , mean intersection over union m_{mIoU} , frequency weighted intersection over union m_{fIoU}) are measured after the training of each network.

4.2 Application of NAS Method to Sensor Fusion Networks

During the search process, the networks are ranked according to their fitness value. The networks with the highest fitness value form the current population and generate offspring networks. As a consequence of the aforementioned objectives, the fitness value is dependent on multiple factors that have to be balanced. The fitness function determines the fitness of a model, and it is a sum of three separate reward functions, each described in the following.

- *Quality Rewards:* For a given network x , the rewards R_μ for the four quality metrics μ are calculated separately and then summed up to a combined quality reward R_q . To avoid bias towards any of the quality metrics, the reward for each has an upper bound R_{max} that is defined by the upper bound for all metrics λ . The rewards for each metric μ are calculated as

$$R_\mu = \begin{cases} \frac{1}{1-\mu_x}, & \mu_x < \lambda, \\ \frac{1}{1-\lambda}, & \mu_x \geq \lambda. \end{cases} \quad (4.2)$$

The four metrics from PASCAL VOC are translated into the combined quality reward

$$R_q(x) = \sum_{\mu_x} R_\mu(x), \quad (4.3)$$

with $\mu_x \in \{m_{acc}(x), m_{pAcc}(x), m_{mAcc}(x), m_{mIoU}(x), m_{fIoU}(x)\}$. During the experiments, the upper bound of the metrics was set to $\lambda = 93.33\%$, which is capping each of the quality metric rewards to $R_{\mu, \max} = 15$ and the total fusion quality reward to $R_{q, \max} = 75$.

- *Inference Time:* The averaged execution time on the embedded hardware t_x is used to calculate the inference time reward

$$R_t(x) = \begin{cases} \frac{t_{\min}}{t_x} R_{t, \max}, & t_x > t_{\min}, \\ R_{t, \max}, & t_x \leq t_{\min}, \end{cases} \quad (4.4)$$

with a lower boundary of inference time t_{\min} and the maximum reward R_{max} for meeting the latency requirements of the task. This timing requirement may be set to any value depending on the task—in this work it is chosen to be $t_{\min} = 20$ ms in order to meet the 50 Hz real-time requirements for the fusion task. During the experiments, the lower boundary of the inference time is set to $t_{\min} = 10$ ms. With this choice, the accelerator is occupied 50% of the time with computations of the grid fusion networks and has still compute capacities for other tasks. According to Equation 4.4, the inference time reward is capped to $R_{t, \max} = 10$.

4 Multi-objective Model Optimization with Neural Architecture Search

- *Model Size:* The total memory footprint m_x —on-chip- and off-chip-memory combined—is taken into account in the model size reward

$$R_m(x) = \begin{cases} \frac{1}{m_x} R_{m,\max}, & m_x > m_{\min}, \\ R_{m,\max}, & m_x \leq m_{\min}, \end{cases} \quad (4.5)$$

with m_{\min} as the minimal required memory footprint and $R_{m,\max}$ as the maximum reward for meeting the memory requirements of the hardware. During the experiments, the optimal memory size is set to $m_{\min} = 150$ KiB. This memory size is chosen in order to fit on to on-chip-memory of the EdgeTPU [44]. Other values of m_{\min} do not influence the $R_{m,\max}$.

- *Fitness Function:* Given a proposed network model x and the respective rewards $R_t(x)$, $R_m(x)$ and $R_q(x)$, let $f(x)$ denote the network’s fitness value

$$f(x) = R_t(x) + R_m(x) + R_q(x). \quad (4.6)$$

In total, the fitness function can reach a maximum value of $f_{\max} = R_{q,\max} + R_{t,\max} + R_{m,\max} = 75 + 10 + 10 = 95$.

With a given, hardware-dependent set of (t_{\min}, m_{\min}) , the goal of the neural architecture search is to find multiple Pareto-optimal solutions. Thus, the multi-objective optimization is maximizing the fitness function

$$\max_x f(x). \quad (4.7)$$

The set of Pareto-optimal solutions to this problem is the hardware-specific family of **GridFuN**-networks. With the outlined maximum rewards, 20 out of 95, so roughly 20% of the incentive is for the hardware-specific metrics and 80% for the fusion quality.

Genetic Diversity by Mutations

The population of network models is initialized by creating random directed acyclic graphs within the constraints of the hyperparameters as cell descriptions. After training the first generation of the population P , the best architectures are determined according to the fitness function (Eq. 4.6). Offsprings with slightly mutated properties are created from these chosen architectures, and the next generation is prepared (Algorithm 2). To maintain the genetic diversity of the current population and to avoid getting stuck in local minima during the search, the following four different mutations are implemented:

4.2 Application of NAS Method to Sensor Fusion Networks

- *Operation Mutation* M_{op} : Within a random node in a random cell type, one of the operations \mathbf{o}_i is selected. After the mutation M_{op} , the operation becomes $\mathbf{o}_j \in \mathcal{O} \setminus \{\mathbf{o}_i\}$.
- *Connection Mutation* M_{con} : If this mutation is called, one of the inputs (I_1, I_2) to a random node i will be changed, so that (I_1, I_2) is becoming either (I'_1, I_2) or (I_1, I'_2) with $I_1 \neq I'_1$ and $I'_1 \neq I_2$, respectively.
- *Number of Filter Maps Mutation* M_{fmap} : Also, the number of initial feature channels χ is searchable as a hyperparameter of the grid fusion networks. This mutation allows the NAS to find an optimal number of filters χ per convolutional layer. With every call of this mutation, χ increases or decreases by 4. The minimal number of feature channels is set to $\chi_{\text{min}} = 4$.
- *Cell Re-initialization* M_{init} : With a call of M_{init} , a cell type is re-initialized completely. This serves the purpose of avoiding local minima in the design space of network architectures. A block type (encoding, decoding) is chosen randomly and reset to a random configuration—ignoring its configuration before.

Algorithm 3 Choice of Mutation.

Input: GridFuN architecture x
Output: Mutated GridFuN architecture x
 $r = \text{randomInteger}(1, 3)$
for $i = 1$ **to** r **do**
 $m = \text{randomInteger}(1, 100)$
 if $m \leq 30$ **then**
 $x = M_{\text{op}}(x)$
 else if $30 < m \leq 60$ **then**
 $x = M_{\text{con}}(x)$
 else if $60 < m \leq 90$ **then**
 $x = M_{\text{fmap}}(x)$
 else
 $x = M_{\text{init}}(x)$
 end if
end for

Each offspring of a parent architecture is mutated according to Algorithm 3. Before the model is trained, it is checked whether this architecture configuration has already been evaluated and stored in the network database. This check saves time as duplicate networks are not trained multiple times and thus, the search space is traversed more efficiently.

4.3 Analysis of Automatically Designed Grid Fusion Networks

Two main experiments are conducted, one with two nodes per encoding and decoding blocks ($K = 2$) and one with a more complex configuration consisting of $K = 3$ nodes per cell. The goal of these two main experiments is to find encoding and decoding cell architectures that are optimal for embedded inference.

The auto-encoder depth is chosen to be $\delta = 4$, as the receptive fields of convolutional filters do not cover the full input occupancy grid at lower depths. The final performance results are received from networks that have been trained on the full dataset \mathcal{D} for 10 epochs. Each search was going on until 1,000 networks have been evaluated. In the following, the evolution of accuracies throughout the search is discussed and the resulting micro-architectures are described.

4.3.1 Search Progress and Evolution Overview

A good metric for a successful evolution is to display the performance of the networks over the duration of the evolution. In Figure 4.8 the test accuracy of each partially trained network throughout the search of the two experiments are displayed as a black dot. The gray lines show the mean accuracy at that time, and it indicates the tendency towards better performing network models. In the experiment with $K = 2$ (Fig 4.8a), the accuracies range from 80 % up to 88 %, and in the experiment with $K = 3$ (Fig 4.8b) from 80 % up to 89 %. Both evolutions show an increase in fusion accuracy over time, thus, they evolve successfully. The more complex architectures ($K = 3$) have an increased capability of learning patterns and features, which becomes apparent in the accuracy gap between the two experiments. Recall that during the search, the networks are partially trained only, and they will perform proportionally and significantly better after full training.

Note, that both experiments topped out after around 500 evaluated architectures. This does not apply for the fitness value over time, which is not only dependent of the accuracy and loss, but also on hardware constraints (Eq. 4.6). The architectures from the current generation with the highest fitness values form the next generation, thus the hardware related rewards are further optimized even after the accuracy has topped out after 500 evaluated networks. The result can be observed in the fitness values throughout the search (Figure 4.9), which steadily increase and top out at around three quarters of the search.

The hardware-related metrics (Fig. 4.10) are received from the deployment on an *EdgeTPU* accelerator. The mean inference times for both experiments are nearly constant for all models; around 8 ms for $K = 2$, and 10 ms for $K = 3$. The discrete patterns that can be observed in the model sizes throughout the search (Figures 4.10c, 4.10d) describe certain configurations of χ feature channels, which are present in steps of 4. Besides the horizontal patterns, there is a minor tendency

4.3 Analysis of Automatically Designed Grid Fusion Networks

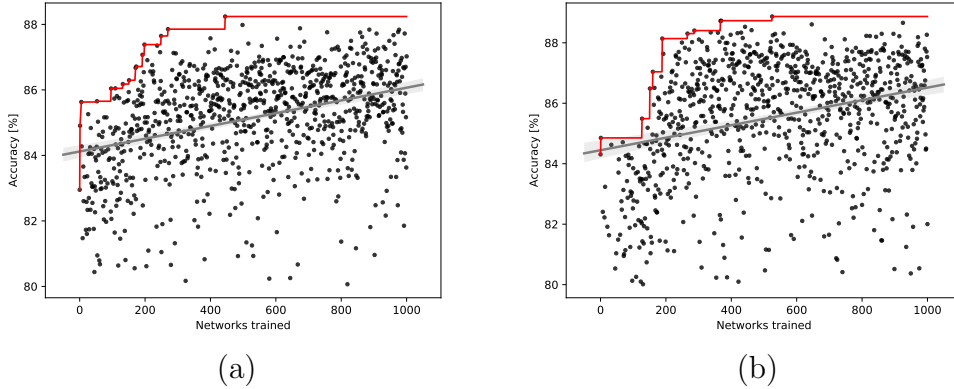


Figure 4.8: Evolution of accuracies over the evaluation of 1,000 network models. (a) $K = 2$, (b) $K = 3$. The red line indicates the best accuracy at this time of the search process. The gray line indicates the mean accuracy rising over time.

of the networks towards larger model sizes at the experiment $K = 2$. For the more complex block configuration $K = 3$, the mean model size remains constant. This shows that the smaller blocks have room for performance improvements in increasing the model size by introducing more weight layers, or in increasing the number of feature channels. The larger blocks ($K = 3$) incorporate an additional node, which intrinsically increases the capabilities of the network by two additional neural operations. Thus, the experiments are designed in a way that a further increase in model complexity is not needed, as the evolution indicates a saturation at $K = 3$.

Further, architectures that have extreme inference times, are discarded by the genetic algorithm, as the time reward R_t is punished. Around network 500 in Figure 4.10b, a model is evaluated on the embedded accelerator with an inference time of 100 ms. The genotype of this model is obviously discarded as no architecture has followed with such a high inference time. High inference times are caused mostly by a bad combination of memory loads, where some weights are stored in the off-chip memory and first need to be time-costly transferred to the processing units of the accelerator.

Ablation Study: Search Without Hardware Constraints

The main experiment was designed with multiple objectives that partially contradicting each other. The hardware constraints are limiting the model size, and thus, the quality. The other way round, a high level of fusion quality requires a high number of computations, and thus, leads to an increased inference time. If the hardware limits are discarded during the search, the main objective is to maximize the fusion

4 Multi-objective Model Optimization with Neural Architecture Search

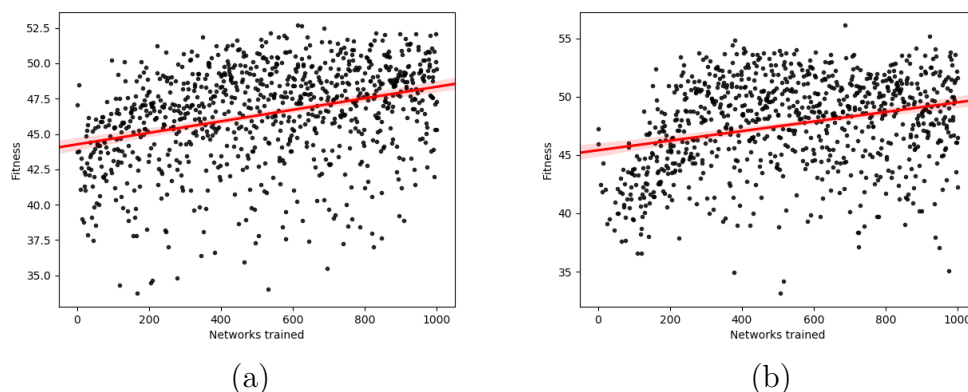


Figure 4.9: Evolution of model fitness over the evaluation of 1,000 network models. (a) $K=2$, (b) $K=3$. The red line indicates the mean fitness value rising over time.

quality only. One can expect the evolution of models to go into the direction of ever larger networks.

This was investigated in a separate NAS experiment, where the fitness function (Eq. 4.6) is limited to the combined quality reward R_q (Eq. 4.3). The experiment with $K=3$ shows that:

- The model size steadily grows throughout the search until the model sizes grow too big for the GPU memory.
- The M_{fmap} mutation is the dominant function of the search, as it lays foundation to increased quality rewards without the need to find an efficient micro-architecture.
- The maximum accuracy is reached faster than during the main experiment, where hardware constraints have to be accounted for simultaneously.
- The search tends to optimize for local minima, as block re-initializations M_{init} are more likely to decrease the fusion quality itself, but may have found a more efficient cell architecture for the embedded hardware.

The multiple evaluations of this experiment are depicted in Figure 4.11. The mean and variance of the best model performances at every state of the search are depicted in Fig. 4.8. It can be observed that the given search method results in a consistent performance of just above 90 % accuracy of the discovered block micro-architectures. Also, the initially high variance of performances decreases with the ongoing evolution until convergence. The convergence is achieved after around 300 evaluated networks and simultaneously by reaching the maximum model size with the ever larger filter map amounts. For the deployment on real-world embedded hardware, searching

4.3 Analysis of Automatically Designed Grid Fusion Networks

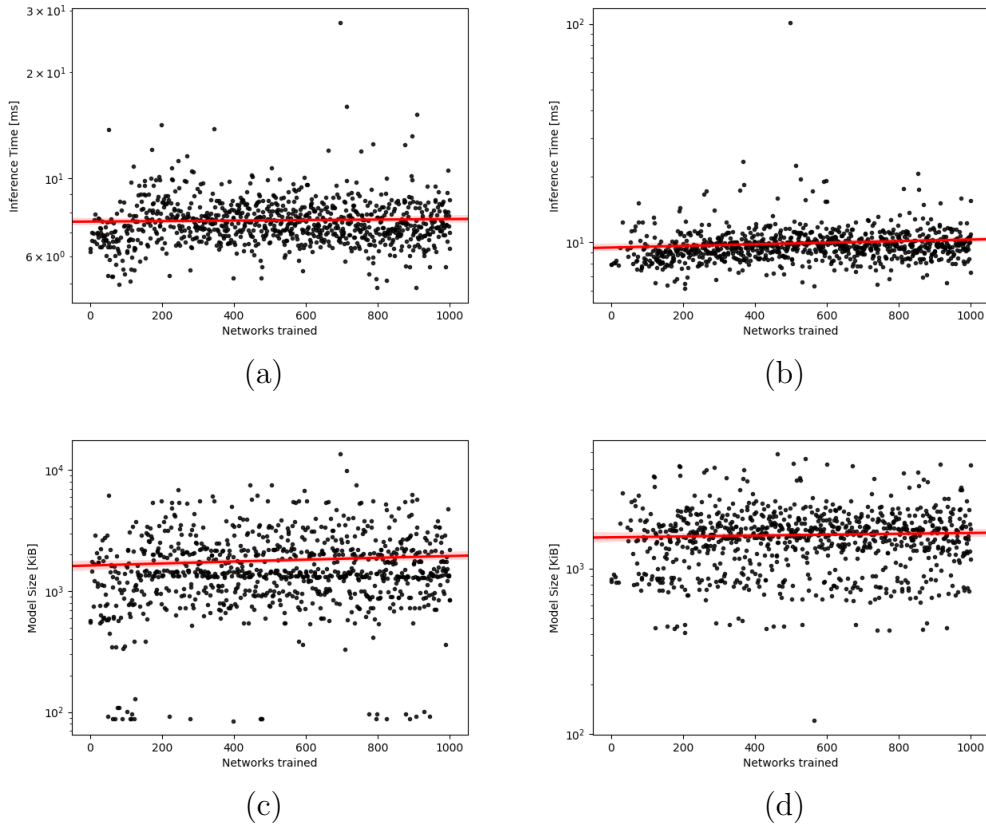


Figure 4.10: Evolution of inference times and model sizes over the evaluation of 1,000 network models. Inference times for (a) $K = 2$, (b) $K = 3$. Model sizes for (c) $K = 2$, (d) $K = 3$. The red line indicates the mean values over time. Note that plots (b), (c) and (d) are displayed on a semi-logarithmic scale.

for network architectures with multiple, contradicting objectives results in a better overall solution. This can be anticipated, because too large number of filters leads to potential overfitting to the dataset. Thus, a smart architecture, not based on a large number of filter channels, generalize better.

4.3.2 Results of the Search

The results of the neural architecture search are the encoding and decoding block micro-architectures that are used to populate the macro-architectural skeleton of the grid fusion networks¹. One way to assess the quality of a model is to look only for one dimension, eg. one metric or the fitness value. A better way of displaying

¹Experimental results published in [11]

4 Multi-objective Model Optimization with Neural Architecture Search

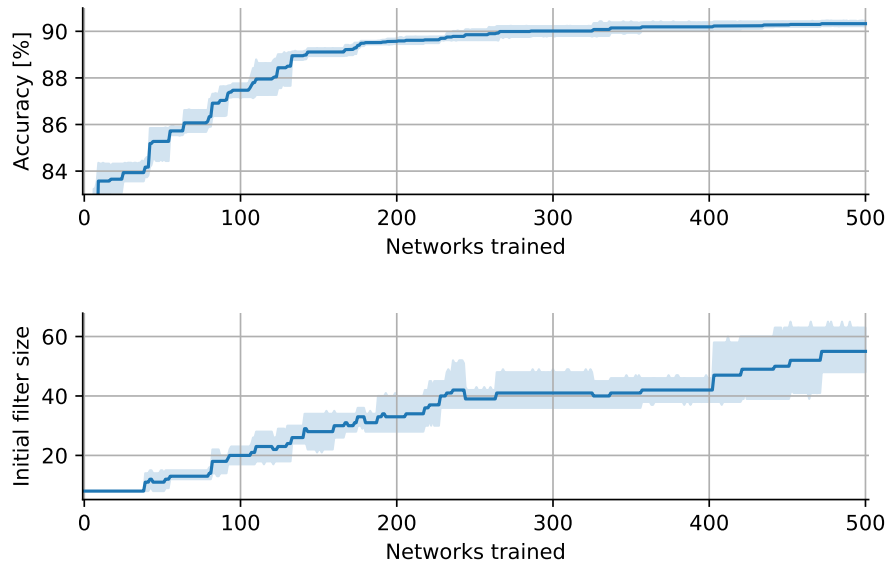


Figure 4.11: Top: Mean and standard deviation of four NAS experiments for optimizing towards accuracy and fusion quality metrics only. Bottom: The number of initial filters χ of the best performing models at this time.

the performance of an architecture is to incorporate also the second dimension of the analysis—the costs for the hardware.

Pareto-Optimal Solutions

Plotting the accuracy over the model size shows the trade-off and the resulting range of network models along a Pareto-optimum (Fig. 4.12).

Note that the networks are optimized to reduce the pixel-wise classification loss during the training—but during the search process, the networks are selected according to the fitness function $f(x)$ described in Chapter 4.2.3. This leads to the stepped appearance of the Pareto-optimum between m_{acc} and the model size.

Both experiments ($K=2$, $K=3$) generated architectures with model sizes ranging from around 100 KiB up to a few MiB (7 MiB and 4 MiB for $K=2$ and $K=3$, respectively). Throughout this range, the accuracies of the fully retrained networks ranged, proportionally to the model size, from 86.5% to 89.8% for $K=2$ and from 88.9% to 90.5% for $K=3$.

Discovered Micro-Architectures

The discovered encoding and decoding blocks of the two experiments are described in this paragraph. The resulting adjacency matrices \mathbf{A} in form of the according directed acyclic graphs are plotted in Figure 4.14. Whereas the search yielded a variety of

4.3 Analysis of Automatically Designed Grid Fusion Networks

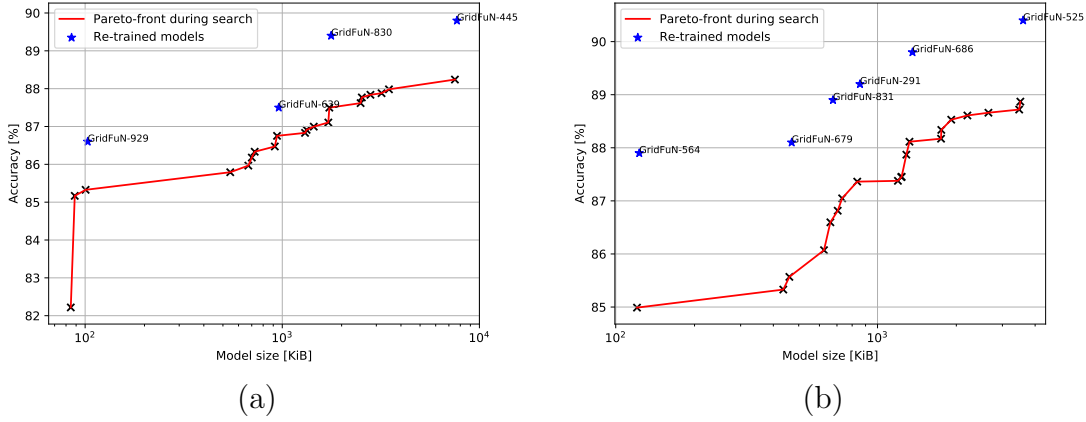


Figure 4.12: The trade-off between model size and accuracy for the two experiments (a) $K=2$ and (b) $K=3$. The black crosses on the red Pareto-optimum are the results of the NAS. Fully re-trained networks are shown as blue stars.

different models, only two exemplary networks are chosen for visualizations. The smallest network (**GridFuN-887**, $K=2$, 90 KiB, $f = 53.14$) is denoted as **GridFuN-S**, and the network with the best fitness value (**GridFuN-686**, $K=3$, 1.36 MiB, $f = 62.48$) is denoted as **GridFuN-L**.

In the micro-architecture of **GridFuN-S**, the edges are populated mostly with computationally cheap operations, such as pooling layers ($\mathbf{o}_{avg}^{3 \times 3}$, $\mathbf{o}_{max}^{3 \times 3}$) and depthwise separable convolutions \mathbf{o}_{sep} (Fig. 4.13a). Note that the decoding block $\mathcal{B}_{dec}^{K=2}$ incorporates pointwise convolutions only (Fig. 4.13b). Thus, the upsampling of the code solely relies on bilinear interpolations $\mathbf{o}_{ups}^{bilinear}$ at the end of the decoding block. The choice of simple operations leaves room for a fairly high number of initial feature channels $\chi = 16$.

The more complex micro-architecture of **GridFuN-L** allows more variety in the choice of operations. All kinds of convolutional layers are present in the encoding block $\mathcal{B}_{enc}^{K=3}$ (Fig. 4.13c)—even standard convolutional layers, whereas \mathbf{o}_{conv} consume more memory than \mathbf{o}_{sep} . In the decoding block $\mathcal{B}_{dec}^{K=3}$, sole separable convolutions are employed (Fig. 4.13d). Note that the input node I_1 is the code from a deeper block of the network, and only point-wise convolutions consume this tensor, but the bypassed information in input node I_2 is analyzed with spatial convolutions, too.

On the node-level of this experiment, it is notable that no intermediate nodes are used, where a node N_i uses another node N_{i-1} as one of its inputs. This means that a flat architecture is preferred, and additional complexity is not beneficial anymore.

4 Multi-objective Model Optimization with Neural Architecture Search

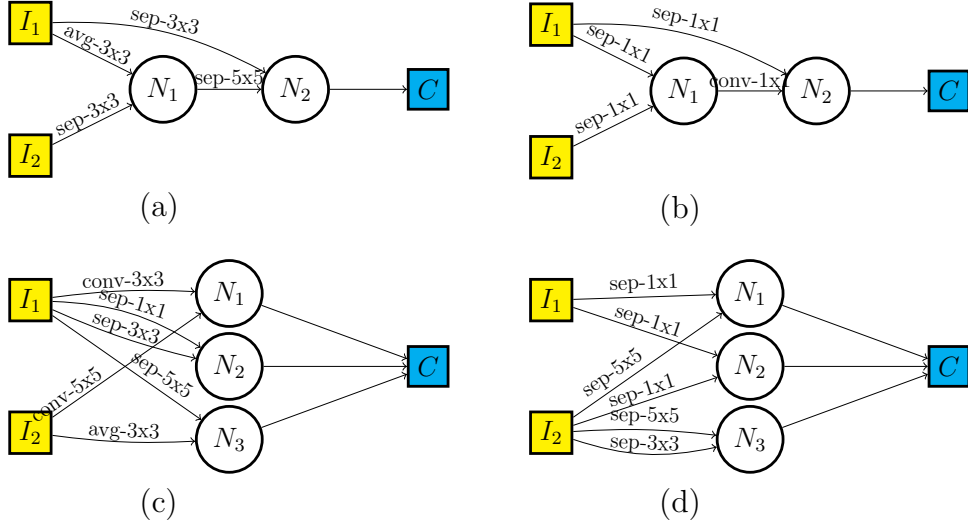


Figure 4.13: Connection graphs of the best cell architectures with different number of intermediate nodes K . GridFuN-S ($K = 2, \chi = 16$): (a) Encoding block $\mathcal{B}_{enc}^{K=2}$, (b) decoding block $\mathcal{B}_{dec}^{K=2}$. GridFuN-L ($K = 3, \chi = 12$): (c) Encoding block $\mathcal{B}_{enc}^{K=3}$, (d) decoding block $\mathcal{B}_{dec}^{K=3}$.

Comparison to Cell Architectures From Literature

To assess the performance of the discovered micro-architectures, a comparison with literature comes at hand. Various neural architecture search projects have searched for similarly defined problems, where only the micro-architecture is searched at a given macro-architecture. Most NAS projects focus on image classification or image segmentation tasks (Chapter 4.1), but none on the same macro-architecture that is introduced in this work. A comparison is still viable, as other NAS projects search for encoding and decoding blocks, too. Thus, for the comparison in this chapter, cells from the literature are plugged into the macro-architecture of the grid fusion task (Fig. 4.5).

The candidates for comparison are chosen according to the similarity and compatibility to our task and listed as follows:

- *AmoebaNet* [108]: The micro-architecture of the *AmoebaNet* encoding cell¹ has $K(\mathcal{B}_{enc}^{Amoe}) = 5$ intermediate nodes. As the original *AmoebaNet* is a classification model, no decoding cell is provided. Thus, the decoding cell of the best performing GridFuN network is used, which has $K(\mathcal{B}_{dec}^{\text{GridFuN-L}}) = 3$ intermediate nodes. The number of initial feature channels of the *AmoebaNet* encoding cell is $\chi = 12$.

¹In the original paper it is called the reduction cell.

4.3 Analysis of Automatically Designed Grid Fusion Networks

- *Auto-DeepLab* [79]: In this work only one block type with $K(\mathcal{B}^{DeepL}) = 5$ was searched. Its architecture is used as encoding and decoding block in the grid fusion macro-architecture. The number of initial filters is $\chi = 8$.
- *Auto-DispNet* [116]: As an auto-encoder, *Auto-DispNet* incorporates both, encoding ($K(\mathcal{B}_{enc}^{Disp}) = 3$) and decoding blocks ($K(\mathcal{B}_{dec}^{Disp}) = 3$). The number of initial filters is $\chi = 18$.

Each of the state-of-the-art blocks is evaluated for the same network depth ($\delta = 4$) that is employed for the NAS experiments of this work, too, and for a reduced depth $\delta = 3$. The reduced depth is investigated due to the large cell sizes (up to 10 operations per block), compared to the **GridFuN** blocks, which have up to 6 operations from cell input to cell output. With an equal depth of $\delta = 4$, the total number of operations from network input to network output sums up to $2 \cdot K \cdot \delta = 2 \cdot 5 \cdot 4 = 40$ operations for the state-of-the-art models and only $3 \cdot 2 \cdot 4 = 24$ for the **GridFuN** networks. Reducing the depth for the models from literature to $\delta = 3$, the operations are reduced to $2 \cdot 5 \cdot 3 = 30$ to the cost of a reduced receptive field of the convolutional filters. On the other hand, training with an increased depth of $\delta = 5$ leads to unstable training results, based on the large number of operations and the hindered gradient flow. Thus, the training configuration with increased depth is not part of the comparison.

These networks are evaluated with the same training environment as the fully trained, proposed networks (Chapter 4.2.3). The results are listed in Table 4.3 along with the performances of the initial, handcrafted models (Chapter 3.3) and the discovered architectures of this chapter. Out of the evaluation of 1,000 networks during the search, about 20 models describe the Pareto-front (Fig. 4.12) for each experiment. Three exemplary architectures are selected from the Pareto-optimal models; besides **GridFuN-S** and **GridFuN-L**, also **GridFuN-M** (**GridFuN-831**, $K = 3$, 680 KiB, $f = 58.99$). **GridFuN-M** offers a well-balanced trade-off between model complexity, model size and the overall fitness.

Figure 4.14 displays the contents of Table 4.3 and, additionally, the performances of different depths for the state-of-the-art networks. The reduced depth $\delta = 3$ implies a reduced receptive field for the filters, which can be observed in the degraded performance of those networks.

4.3.3 Discussion of Results

Whereas the network constructed out of the *AutoDispNet* blocks has the largest model size with 6.36 MiB, the performance metrics are rather poor, compared to the other models with lighter models. The $\chi = 18$ feature channels obviously enlarge the computational load, seen in the latency of 19.3ms for a forward pass, but in return the performance metrics are not enhanced in the same extent. Thus, the fitness of *AutoDispNet* is the lowest of this comparison ($f = 31.20$).

Table 4.3: Performances of different cells in the grid fusion macro-architecture. (*) *Auto-DeepLab* [79] cell placed as \mathcal{B}_{enc} and \mathcal{B}_{dec} . (**) *AmoebaNet-A* [108] reduction cell as C_E and best *GridFuW* decoding cell. (***) *Auto-DispNet* [116] encoding and decoding cells. ^{1,2} Best performing networks from [10] after full training. DGF uses unsupported operations that could not be mapped to the *EdgeTPU*.

Methods	$K(\mathcal{B}_{enc}/\mathcal{B}_{dec})$	χ	δ	NAS	Acc. [%]	mIoU [%]	Size [MiB]	Latency [ms]	Fitness
DGF ¹	n/a	16	5		92.6	85.2	8.87	-	-
DGF-SQ ²	n/a	16	5		89.8	79.1	1.71	-	-
GridFuW-S	2	16	4	✓	87.1	80.2	0.09	5.8	53.14
GridFuW-M	3	8	4	✓	88.9	82.6	0.68	8.3	58.99
GridFuW-L	3	12	4	✓	89.8	84.3	1.36	9.1	62.48
Auto-DeepLab*	5	8	4	✓	83.5	70.0	1.49	12.2	42.47
AmoebaNet**	5/3	12	4	✓	88.1	81.2	1.49	11.2	55.04
AutoDispNet***	3	18	4	✓	83.2	79.9	6.36	19.3	31.20

4.3 Analysis of Automatically Designed Grid Fusion Networks

Auto-DeepLab on the other hand has the blocks with most intermediate nodes ($K=5$) and the least number of filters ($\chi=8$) of this analysis. With these parameters, it achieves better results than *AutoDispNet* with a quarter of memory requirements (1.49 MiB). Still, the quality lacks behind the proposed **GridFuN** models, so that the fitness value is only at $f=42.47$.

The encoding cells of *AmoebaNet* together with the decoding cells of **GridFuN-L** form a grid fusion model, which is the best of the three external models. Its size is equal to *Auto-DeepLab*, but composed of operations that are executed faster on the target hardware (12.2 ms vs. 11.2 ms per forward pass). The accuracy m_{acc} and the m_{mIoU} perform very well in the range of the final **GridFuN** networks. This is based on the specially searched, thus superior decoding cell of **GridFuN-L**, which leverages especially the quality metrics. When compared to the proposed models, *AmoebaNet*'s encoding cell is computationally expensive, thus, resulting in a lower fitness score ($f=55.04$) while showing good fusion quality.

The cell-based, NAS-found **GridFuN** architectures are also compared to the handcrafted networks from Chapter 3, namely the DGF and its squeezed variant DGF-SQ. The fusion performance of the DGF is superior to the automatically designed ones, but it comes to the cost of significantly larger model sizes. To cope with the large model size, DGF-SQ employs *FireModules* and *DeconvFire* modules from the *SqueezeNet* family, but the fusion quality suffers as a consequence. The plain DGF model consumes 8.87 MiB of memory achieving the top accuracy of 92.6% and the top $mIoU$ of 85.2%, whereas the squeezed variant DGF-SQ uses only 1.71 MiB, but with an accuracy of 89.8% and an $mIoU$ of 79.1%.

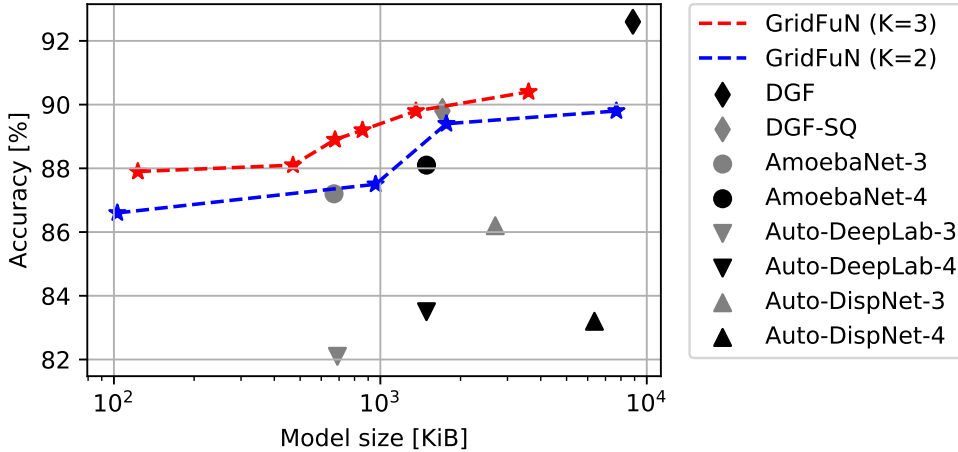


Figure 4.14: Comparison of blocks discovered by this NAS [11] to cell architectures from literature (*AmoebaNet* [108], *Auto-DeepLab* [79], *Auto-DispNet* [116]) and handcrafted networks (*DGF* [10], *DGF-SQ* [9]).

The results from Table 4.3 are depicted in Figure 4.14, where the accuracy is plot-

ted over the logarithmic model sizes. The resulting networks of the two experiments ($K=2$, $K=3$) are shown in a Pareto-front representation as stars on the blue and red lines. Each experiment has models ranging from a tiny sizes of about 100 KiB, up to a quality-oriented variant with multiple MiB. The networks constructed of blocks from the literature are evaluated in two variants each, one with depth $\delta = 3$ and another with $\delta = 4$, in order to span over a range of possible performances. Also, the handcrafted DGF and DGF-SQ are placed in the comparative figure, but only in the state that is described in Chapter 3.

The higher the fitness value of a network is, the more it is placed towards the top-left corner. Now it can be easily seen that out of the models from literature, *AmoebaNet* performs best, as it reaches comparable performance as the **GridFuN** ($K=2$) experiment. The other candidates from literature are outperformed by the proposed approaches in this given task.

Limitations of the Experiments

The experiments have been conducted in order to enhance and outperform the handcrafted models with the embedded deployment in mind. Due to the limited search time and computational power, the macro-architecture has been fixed for this given task to have only two input branches and a static interconnection in between the blocks. If this constrained is loosened, even better results can be expected.

Also, the the **GridFuN**-cells are specifically searched for the grid fusion task, whereas the blocks from literature were discovered for different datasets. *AmoebaNet*'s reduction cells are found for the *CIFAR* dataset, *Auto-DeepLab*'s blocks for the *Cityscapes* and *PASCAL VOC* datasets, and the cells of *Auto-DispNet* for the *FlyingThings3D* and *Sintel* datasets. Still, on a higher abstraction level, the blocks are performing the same task, meaning encoding and decoding tensors. The differentiation in macro- and micro-architecture is based on the scalability and exchangeability of those computational blocks, so that this comparison stands to reason.

4.4 Conclusion

In this chapter, an automated multi-objective optimization is performed on the grid fusion problem from Chapter 3. In order to be applicable for the neural architecture search (NAS) optimization, the grid fusion networks are described in a macro- and a micro-architecture, where the macro-architecture is a preset, fixed skeleton of blocks and the micro-architecture is the description of the blocks.

For a given auto-encoder macro-architecture, the NAS finds optimal blocks that maximize a custom reward function. The goal of the reward function is twofold and contradicting: Maximize the fusion quality and minimize the hardware requirements. As a result of the search, a series of networks is found that describes the Pareto-front between model size and fusion quality.

4.4 Conclusion

The networks found in two experiments with 1,000 evaluated networks each, are denoted as the **GridFuN** family. They are built from encoding blocks (\mathcal{B}_{enc}) and decoding blocks (\mathcal{B}_{dec}), which are comparable to other semantic segmentation tasks in literature. Thus, in a comparison between the proposed blocks and blocks from literature, the **GridFuN** micro-architecture outperforms blocks from *AmoebaNet*, *Auto-DeepLab* and *Auto-DispNet* in the grid fusion task.

5 Additional Application for Sensor Fusion Neural Networks

The generic architecture of the neural networks discussed in Chapter 3 and Chapter 4 is also applicable to other sensor fusion use-cases. Once the input data is transformed into the correct form to feed into the CNN structure and enough training data is available, the fusion can be learned. In this chapter, an additional use-case is evaluated applying a sensor fusion architecture to combine a radar and an infrared time-of-flight (ToF) camera. First, the use-case is outlined, followed by the description of the special dataset for this application. Then, a variety of model architectures is discussed and, finally, the performance of the architectures is compared to literature by experimental evaluation.

5.1 Use-case: Gesture Recognition for Multimedia Controls

In the cockpit of upcoming cars, the interface to the machine will be different than how human-machine-interfaces (HMI) work today. A trend towards a system without touch sensors is seen, for example devices with voice-control such as smartphones or home assistants. Non-haptic controls are also beneficial for safe driving, as the driver does not need to split his concentration between the road and multimedia controls [144]. The driver will be able to swiftly adjust any multimedia settings with his voice or the help of hand gestures without driving blindly.

In accordance with the steadily increasing awareness and need of privacy and personal data protection, there is an interest in a system for driver monitoring without camera sensors. Radar and depth sensors produce data that denies easy identification of individuals, thus, are a good choice for this task.

The different nature of data coming from time-of-flight and radar sensors leads to different pre-processing schemata and difficulties in combining the information with traditional methods. The superior classification accuracy and generalization capabilities of neural networks come to the cost of immense computational effort for the processor [6].

To meet the requirements of mass-produced vehicles, all computations have to be performed on automotive microcontrollers and embedded CNN accelerators. This applies not only to safety-critical tasks as ADAS, discussed in Chapter 4, but also

for multimedia applications. Thus, also the hand gesture recognition system has to be optimized for embedded inference with lightweight models, instead of relying on heavy, deep neural networks.

5.1.1 Spatio-temporal Classification in Literature

Similar to the related works in Chapters 3 and 4, the traditional methods of processing are outperformed with deep learning models. After the initial success of neural networks for static image processing [43, 75], architectures were introduced to also deal with spatio-temporal data, such as video sequences. A way of interpreting it is to use the different video frames as feature channels of one input tensor to 2D CNNs [39, 117].

The authors of [31] use a combination of convolutional layers and long-short term memory (LSTM) cells in order to classify spatio-temporal data. A CNN is applied frame-wise to extract features, which then are passed to the LSTM for classification over time. Similar approaches use 3D CNNs to acquire local spatio-temporal features, which are fed into an LSTM to calculate global, long-term features of the video [91, 129].

These advances opened the door to activity classification and gesture classification frameworks. Most hand gesture sensing systems rely on optical sensors, especially on vision data from camera sensors. A very powerful approach is using the combination of RGB images with the optical flow to enhance accuracy of spatio-temporal classification [4, 73]. Authors of [131] make use of optical flow, too, but feed it along with frame-wise features into a three-stream 2D CNN for action recognition.

While vision based approaches show good results, research also focuses on robust multi-modal systems and

5.2 Dataset Description

In this section, the dataset, its gathering and mandatory pre-processing steps are described. Also, the data is analyzed statistically in order to gain insights on the subsequent neural network design phase.

5.2.1 Data Gathering

The data was gathered indoors and inside the car with the desired mounting position in vicinity of the gear selection lever. With regards to train an online-classification system, a constant stream of data is recorded as a raw base for further processing. Two sensors are fixed to a custom, 3D-printed mount. They monitor the hand gestures in the proximity of the gear selector lever, pointing up towards the rear mirror.

Table 5.1: Amount of gesture sequences per gesture class in the dataset recorded with radar and ToF sensors.

Gesture	<i>down</i>	<i>up</i>	<i>left</i>	<i>right</i>	<i>forward</i>	<i>rotate</i>	<i>open</i>	<i>piano</i>	<i>rub</i>	Total
Amount	186	275	256	235	260	192	286	224	311	2,225

Signal Recordings

The gestures are expected to be performed above the sensors in a close range of $r = [0 \text{ m}, 0.3 \text{ m}]$. The ToF sensor¹ is configured to scan this distance and it delivers a three-dimensional point cloud. This point-cloud is projected into a two-dimensional image plane, where the depth is denoted by the grayscale intensity. This image spans a field of view of 62° in azimuth and 45° in elevation with a resolution of 224×171 pixels.

The radar sensor² uses a FMCW radar—the same modulation scheme as in Chapter 3.3, but with the center frequency at 60 GHz, instead of 77 GHz. The regulation allows to sweep a larger frequency bandwidth (58 GHz – 63 GHz) at this frequency band, resulting in a finer range resolution. One radar frame is composed of 32 chirps ($N_c = 32$, $T_c = 0,8 \text{ ms}$) and each chirp consists of 64 samples. The radar is configured to measure 10 frames per second ($T_f = 0.1 \text{ s}$). The resulting maximum velocity

$$v_{max} = \frac{\lambda}{4T_c} = \frac{5 \text{ mm}}{3.2 \text{ ms}} = 1.56 \text{ m/s} \quad (5.1)$$

allows the system to detect gestures, even when carried out quickly. The range resolution based on the 5 GHz frequency sweep is $\Delta r = \frac{c}{2B} = 0.03 \text{ m}$. The velocity resolution is

$$\Delta v = \frac{V_{max}}{\frac{N_c}{2}} = \frac{\lambda}{2N_c T_c} = 9.75 \text{ cm/s}. \quad (5.2)$$

Both sensors of the system are synchronized to a rate of 10 Hz and during the dataset recording, they are sampling constantly. The individual gesture sequences are extracted offline after the recordings. In order to have a dataset with high variance, multiple subjects were recorded performing the hand movements above the sensors. Gestures performed by left and right hands were recorded in order to learn control inputs of both, the driver and the passenger. The classes form a set of intuitive gestures for multimedia control.

From the stream of information, gestures are extracted for building the training dataset. A thresholding method is employed to the ToF and radar streams to detect active gestures and mark the start and end frames of a gesture. The gesture is saved to a hard disk in raw format along with the corresponding label. In total, the dataset

¹pmd CamBoardpico flexx

²Infineon BGT60TR13C

5 Additional Application for Sensor Fusion Neural Networks

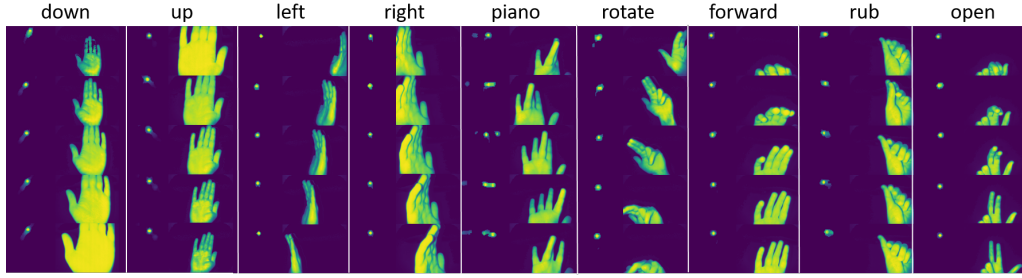


Figure 5.1: Exemplary visualization of the gesture classes with the two sensor modalities each.

consists of 2,225 gestures with gesture lengths up to 29 frames, describing 9 gesture classes (Tab. 5.1, Fig. 5.1).

Data Pre-Processing

The ToF images are processed with a Wiener filter to smooth the noisy depth values. Furthermore, the depth values beyond the desired ranges are filtered out. This is done to mitigate extensive noise in the background and focus the signal on the important gesture itself. The noise is based on irrelevant objects of the scene and, if there is no object within the range of operation, thermal noise.

The radar data is processed in multiple steps similar to the radar pre-processing introduced in Chapter 2.1.1: First, the chirps are brought to zero mean by subtracting the mean value of a chirp from each of the samples. Then, the range is computed with a first-stage fast Fourier transform (FFT) over the range samples with an FFT size of 128, from which the positive half is used. The range-Doppler images (RDI) are computed with the second stage FFT with an FFT size of 64, resulting in an RDI dimension of 64×64 . Before each of the FFTs, the signal is multiplied with a Hann window function in order to identify clean peaks and optimize the peak to sidelobe ratio (PSLR). Other window functions, such as the Chebyshev window, result in higher PSLR, but to the cost of broader mainlobes. Subsequently, the absolute values of the RDI are thresholded with an ordered statistic CFAR (OS-CFAR) in order to maximize the signal-to-noise ratio. OS-CFAR was chosen because of better multi-target capabilities in comparison to cell averaging CFAR (CA-CFAR). This is expected to have a positive impact on the classification of the gestures with individual fingers moving (compare RDIs of *piano* gesture in Fig. 5.1).

Data Format

After data pre-processing, the contents of one training sample are the two input sequences and the corresponding label. A raw radar sequence is a volume $\mathbf{T}_{\text{rad}} \in \mathbb{R}^{t \times x \times y \times f}$, where $t \geq 1$ denotes the timesteps in this sequence. Each timestep

stores a RDI with $x \times y$ as the range and Doppler dimensions, and f as the number of feature channels. Here, $f = 1$, as only the intensity of the RDI is used. A raw ToF sequence $\mathbf{T}_{\text{tof}} \in \mathbb{R}^{t \times x \times y \times f}$ is a similar volume to \mathbf{T}_{rad} , but $x \times y$ denote the pixel dimensions of the ToF sensor output and there might be a different value of t . There is only one feature $f = 1$ for \mathbf{T}_{tof} , too, as it describes the distance of a target to the sensor.

A transformation τ needs to be performed in order to feed these tensors into the network model. τ depends on the network type and the embedded hardware to be deployed on. Further information about τ is described along the neural network models in Section 5.3. In the following a gesture \mathcal{G} denotes spatio-temporal data in form of a tuple $\mathcal{G} = (\mathbf{T}_{\text{rad}}, \mathbf{T}_{\text{tof}})$. The entries of \mathcal{G} are a radar and a ToF sequence of the same class.

Temporal Adjustments

The gestures vary in length so that it is not possible to directly feed them into a network of constant input size (Fig. 5.2a, 5.2b). In order to adjust \mathbf{T}_{rad} and \mathbf{T}_{tof} to the same length t_0 , the sequences are zero-padded, before and after the original sequence. The fix length t_0 is chosen to be the maximum gesture length of the dataset. Within these t_0 frames, the gesture is put into a random position.

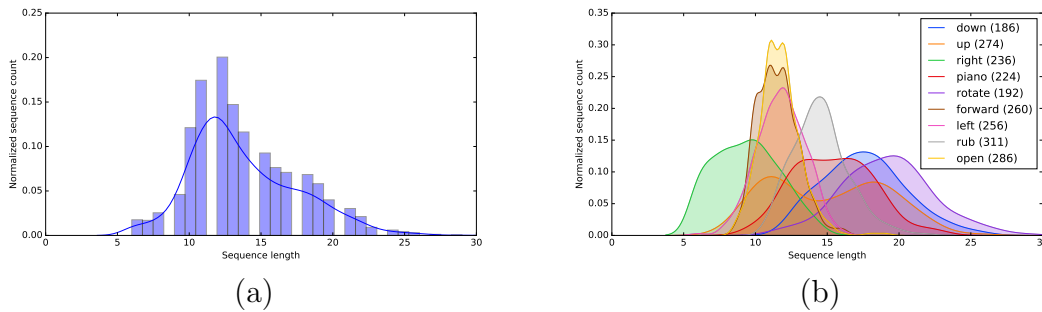


Figure 5.2: (a) The dataset consists of gestures of various lengths. During training, the gesture lengths are padded to a uniform length in order to have constant length input tensors. (b) Per-class distribution of gesture lengths. The number in brackets denotes the amount of gestures in this class.

Dataset Statistics

5.2.2 Data Augmentation

In order to increase robustness of the system, following data augmentation techniques are applied. They are utilized randomly to individual input streams.

5 Additional Application for Sensor Fusion Neural Networks

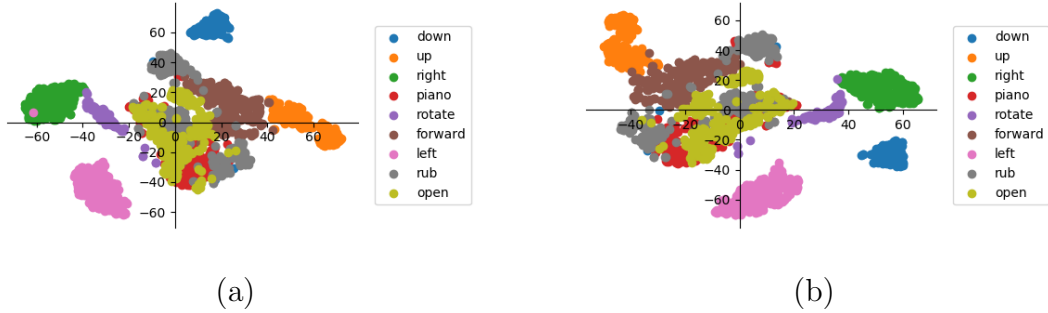


Figure 5.3: The plots show the t-SNE representation of the down-scaled (a) ToF and (b) radar sequences of the dataset. The image dimensions are 32×32 and the sequences are padded to a uniform length. Clearly, the gestures for navigation (*left, right, up, down*) form distinct clusters, whereas the other gesture classes are more difficult to distinguish. Both sensor modalities provide cluster information about the classes so that the networks can benefit from both sensor types.

- *Random shifting of complete sequences*

The ToF sequences \mathbf{T}_{tof} are shifted in both pixel dimensions (x, y) by a random value which can be up to 10% of the respective pixel dimensions ($\pm 0.1x, \pm 0.1y$). The radar sequences \mathbf{T}_{rad} are shifted randomly by up to 5% in Doppler, and 10% in range dimension ($\pm 0.05x, \pm 0.1y$). The empty space is filled with zeros.

- *Zeroing out regions*

A random areas in the image are selected, which are then filled with zeros. The selection can either be the borders of the image for ToF, or patches within the image for both sensor modalities. The border padding simulates different scenarios of gestures that are not completely included within the field of view of the sensor. Random patch zeroing reduces overfitting of the network to certain regions. The patch sizes are from one pixel up to a square of 5×5 pixels.

- *Adding constants to the sequence*

To reduce the impact of numerical values, a random integer with a value up to 5% of the maximum pixel value is added to the sequences.

5.3 Neural Network Architectures

The sensor fusion neural networks for classifying the gesture sequences—short gesture classification networks (GCN)—have two parts. First a data pre-processor

converts the gestures \mathcal{G} into a compatible format with the transformation τ . Then, the actual network architecture \mathcal{A} predicts a class upon the input. Each network consists of repeated individual encoding blocks, similar to the encoding structures of Chapter 3 and 4. As these networks are classifying data, no decoding is needed to upsample, only a fully connected classification layer after the encoder. Each cell consists of operations that are compatible with the Google EdgeTPU (normal 2D convolutions \mathbf{o}_{conv} , max pooling \mathbf{o}_{max} , batch normalization \mathbf{o}_{BN} , dropout \mathbf{o}_{DO} and ReLU activations \mathbf{o}_{relu}). Multiple encoding blocks are stacked after each other until the processed tensors reach a desired final embedding shape. Each sensor modality incorporates its own encoding branch, where finally the embeddings are used for late fusion and classification. The fusion is done with a fully connected layer $\mathbf{o}_{fc}^{n_c}$ with n_c neurons that uses the concatenated information of both encoding branches. n_c denotes the number of classes in the dataset. As a summary, each network $\mathcal{F} = (\tau, \mathcal{A})$ maps the input gesture \mathcal{G} to an n_c -dimensional vector ($\mathcal{F} : \mathcal{G} \rightarrow \mathbb{R}^{n_c}$).

In the following, four network architectures \mathcal{A} with their individual transformations τ are described. Each approach has a common that first, the spatial information is retrieved from individual frames, followed by a temporal integration over the time steps of the gesture.

5.3.1 Transformations for Dimensionality Reduction and Network Architectures

Time Distributed, $\mathcal{F}^{TD} = (\tau^{TD}, \mathcal{A}^{TD})$

This network variant processes each time step with a shared CNN backbone with `TimeDistributed` layers¹ to retrieve spatial information, followed by temporal integration using LSTM cells. Each input stream has its own n_{LSTM} LSTM cells. For this approach, no transformation is needed, because the LSTM handles four-dimensional data intrinsically. Thus, the transformation is the identity $\tau^{TD} : \mathbb{R}^{t \times x \times y \times f} \rightarrow \mathbb{R}^{t \times x \times y \times f}$.

The encoding cells use convolutions with 3×3 filter size, and are wrapped into `TimeDistributed` layers. Once a spatial dimension (x, y) is below 8, the embedding of the last cell is passed to an LSTM cell. Each input modality is assigned to its individual LSTM unit. Hence, the temporal integration takes part after spatial feature extraction and before the fusion.

3D Conv, $\mathcal{F}^{3D} = (\tau^{3D}, \mathcal{A}^{3D})$

In this approach, 3D convolutions are used directly on the spatio-temporal data of each sensor type. No transformation is needed for this approach, consequently $\tau^{3D} : \mathbb{R}^{t \times x \times y \times f} \rightarrow \mathbb{R}^{t \times x \times y \times f}$.

¹`TimeDistributed` layers from Tensorflow Keras.

5 Additional Application for Sensor Fusion Neural Networks

The encoding cells of the \mathcal{A}^{3D} architecture use 3D convolutions and 3D max-pooling. Their filter dimensions are chosen in order to filter separately for spatial and temporal features. For a given spatio-temporal tensor $\mathbf{T} \in \mathbb{R}^{t \times x \times y \times f}$, three-dimensional convolutional filter $f_s \in \mathbb{R}^{1 \times 3 \times 3}$ is applied for spatial, and $f_t \in \mathbb{R}^{3 \times 1 \times 1}$ for temporal filtering. Accordingly, the max pooling sizes are chosen to reduce the respective dimension. Once the spatial dimension (x, y) is below 8, both tensors are flattened to vectors and concatenated for fusion and classification.

Video as Image, $\mathcal{F}^{VI} = (\tau^{VI}, \mathcal{A}^{VI})$

Each RDI and ToF image can be reshaped to vectors in order to solve the problem of an additional dimension for the time. Those vectors are stacked to one single image per gesture, which can be processed with a standard network with two-dimensional convolutions. The transformation needed is

$$\tau^{VI} : \mathbb{R}^{t \times x \times y \times f} \longrightarrow \mathbb{R}^{t \times xy \times f}. \quad (5.3)$$

After the transformation τ^{VI} , the input tensors do not have the 4th dimension. Hence, they are processed with a 2D CNN. As the individual frames in this tensor are vectors, spatial feature extraction is done with 2D convolutions with filter size 1×3 along the vectors dimension. Generally, 2×2 max pooling layers reduce the image area by $1/4$. Similarly, the max pooling in this approach reduces by the same amount, but in the single direction of the image vectors by using 1×4 max pooling. These encoding cells are stacked after each other until the image vector dimension z is below 32, resulting in an embedding of shape $\mathbb{R}^{t \times z \times 1}$, $z \in (0, 32)$.

Time as Feature, $\mathcal{F}^{TF} = (\tau^{TF}, \mathcal{A}^{TF})$

Both inputs have only one feature ($f = 1$), namely the reflectivity in RDI, and distance in ToF images. This allows rearranging the input tensor in a way, so that the time dimension t is placed as the feature channel f of the input \mathbf{T}_{in} . The transformation

$$\tau^{TF} : \mathbb{R}^{t \times x \times y \times 1} \longrightarrow \mathbb{R}^{x \times y \times t} \quad (5.4)$$

produces a three-dimensional input tensor $\mathbf{T}_{in}^{TF} \in \mathbb{R}^{x \times y \times t}$ to the network.

In contrast to τ^{VI} , where the spatial context of the frames is lost, the transformation τ^{TF} preserves the images, but still reduces the dimensionality to three. The input tensors are processed with encoding cells using 3×3 convolutions until the spatial dimension (x, y) is below 8 for both inputs branches. The embeddings of the tensors are flattened, concatenated and then fed into the fully connected layer for classification.

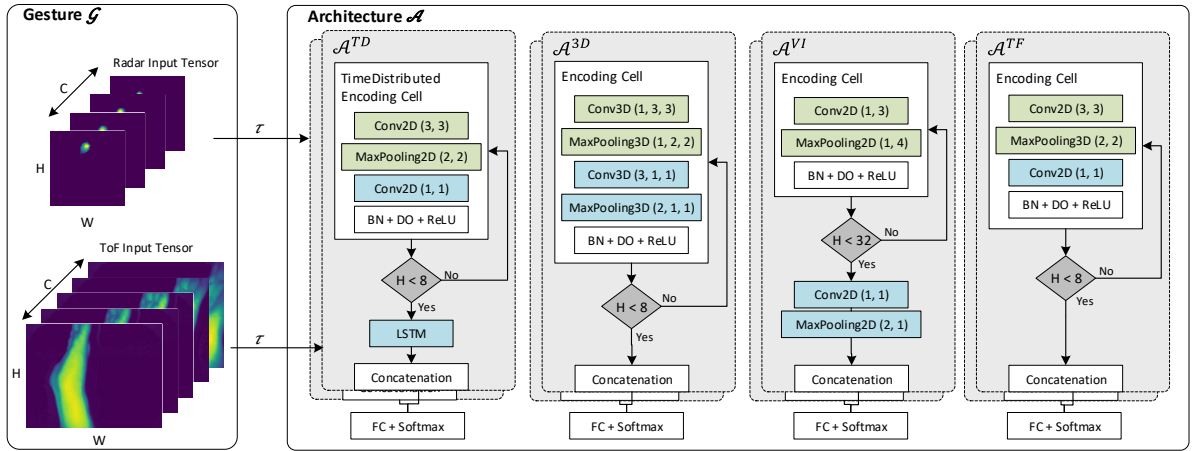


Figure 5.4: The general architecture of the proposed gesture classification networks GCN. The gestures \mathcal{G} from the dataset are fed through the transformation τ to the architectures \mathcal{A} . Together they form the network $\mathcal{F} = (\tau, \mathcal{A})$. At the end of each encoding cell, batch normalization (BN) and drop out (DO) are applied before ReLU-activation. Green colored nodes denote operations for spatial feature extraction. Blue nodes show the operations that reduce in the time dimension. Both sensor modalities are processed with the same architectural structure. Here, the processing of only one modality is shown in the foreground for a better visualization. The concatenation fuses the information of both input streams.

5.3.2 Network Scaling Factor

Each above mentioned approaches \mathcal{F} can process arbitrary input tensor sizes, such as the raw input format (RDI: 64×64 , ToF: 224×171) or a reduced input size S (RDI: 32×32 , ToF: 32×32), and each approach can be tuned with the number of convolutional filters b . The scaling factor directly influences the size and capabilities of the model, and it is determined by

$$b = (b_0 + 2)^\delta, \quad (5.5)$$

where parameter b_0 is the offset and δ the depth of the encoding cells.

Several configurations of b are evaluated in the experiments in order to find a suite of networks with varying capabilities. As b is dependent of b_0 and δ , and the network depth is known from the spatial dimensions of the input and the architecture type, the number of filters per convolutional layer is set by $b_0 \in \{0, 1, 2, 3\}$.

5.3.3 Embedded Deployment

The GCN models aim for embedded deployment, thus, have to meet the hardware requirements of the targeted accelerator devices. Similarly to Chapter 4.1.2, the selection of hardware accelerators is the suite of Google EdgeTPU, Intel Neural Compute Stick 2 and NVIDIA Jetson nano. Recall the supported operations for each HW accelerator (Table 4.1). \mathcal{A}^{3D} uses 3D convolutions, \mathcal{A}^{TD} relies on LSTM and TimeDistributed Keras layers. Consequently, only the \mathcal{A}^{TF} and \mathcal{A}^{VI} architectures are used for the embedded deployment, as they comply with the model requirements of the embedded accelerators.

5.4 Experimental Evaluation

The GCN networks are evaluated in this section, where first, the training on a GPU is described, followed by the embedded implementation, and finally, the results are compared to literature. The analysis and results are published in [8] as a conference paper¹.

5.4.1 Training Results and Embedded Implementation

Training Configuration

The networks are trained on an NVIDIA TITAN V GPU. Convergence is achieved after 100 epochs of training with a train-test split of 80%. Each network architecture is evaluated three times and the mean performance values are reported. Weight

¹Paper [8]: ICAART 2021.

training is done with an ADAM optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The learning rate is regulated with a cosine decay with an initial value of $1e-4$, a warm-up of 10 % of the total iterations and holding the maximum learning rate for 10 % of training iterations. ADAM optimizer minimizes the softmax cross-entropy loss of the predicted n_c classes. The weights of the convolutional filters decay with an L2-regularizer with a regularization factor of $1e-4$ and a dropout of 40 % is applied to fully connected layers.

After each epoch, the class-wise accuracy α_s is evaluated on the test set and used for scaling the weights of each class w_s for the next training epoch. The optimizer weights the classes according to

$$w_s = 0.5 + \frac{1 - \alpha_s}{2}. \quad (5.6)$$

Weights range from 0.75 to 1.25, where higher accuracy leads to lower weights and vice versa.

Classification Performance

All of the approaches are trained on the GPU¹ and then saved to frozen network model files. The frozen representations are then used to be deployed on the three accelerator devices (Section 5.3.3) by converting the models to the corresponding intermediate representations of the hardware.

Besides the pure classification accuracy, the networks' performances are measured also in the following metrics (TP: *true positive*, FP: *false positive*, TN: *true negative*, FN: *false negative*) [100]:

- Accuracy $m_{\text{acc}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$
- Precision $m_{\text{prec}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$
- Recall $m_{\text{rec}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$
- F1-score $m_{\text{f1}} = 2 \frac{m_{\text{prec}} \cdot m_{\text{rec}}}{m_{\text{prec}} + m_{\text{rec}}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$

The four approaches ($\mathcal{F}_b^{\text{TD}}$, $\mathcal{F}_b^{3\text{D}}$, $\mathcal{F}_b^{\text{VI}}$, $\mathcal{F}_b^{\text{TF}}$) show a good classification performance that is strongly dependent on the architectural parameter b . With the largest number of convolutional filters in the experiments ($b = 32$), the best accuracies are achieved with all approaches (Fig. 5.2). The best performing model is the time-as-feature approach with an accuracy $m_{\text{acc}} = 97.9\%$ at full precision, resulting in a model size of $M_{32}^{\text{TF}} = 967$ KB.

Decreasing the number of filters b in the models leads to lower classification performance. As seen in Fig. 5.5, the accuracy is directly correlated to b . The

¹NVIDIA TITAN V GPU

5 Additional Application for Sensor Fusion Neural Networks

Table 5.2: The best performances of the different gesture classification approaches given the best scaling factor b and the reduced input S (32×32 , 32×32). M denotes the model sizes.

Model	Input S	
	m_{acc}	M
$\mathcal{A}_{32}^{\text{TF}}$	95.1 %	598 KB
$\mathcal{A}_{32}^{\text{3D}}$	96.6 %	529 KB
$\mathcal{A}_{32}^{\text{VI}}$	96.9 %	667 KB
$\mathcal{A}_{32}^{\text{TF}}$	97.9 %	967 KB

Table 5.3: Comparative results of proposed GCN architectures. M denotes the model sizes. The measurements are based on the reduced input resolution S (32×32 , 32×32).

Model	M	Acc	Prec	Rec	F1
$\mathcal{A}_{32}^{\text{VI}}$	0.67 MB	96.9 %	97.1 %	96.9 %	97.1 %
$\mathcal{A}_{32}^{\text{TF}}$	0.97 MB	97.9 %	98.0 %	97.9 %	97.9 %
3DCNN [90]	10.86 MB	98.6 %	98.4 %	98.3 %	98.3 %

largest scaling of accuracy (77.9%–95.1%) is seen in Fig. 5.2 with the model $\mathcal{F}_b^{\text{TD}}$, $b \in \{4, 8, 16, 32\}$. On the low-end of model scaling, the network $\mathcal{F}_4^{\text{TF}}$ achieves 92.3% accuracy with only $M_4^{\text{TF}} = 149$ KB of full precision parameters.

Ablation Study - Importance of Multiple Sensor Modalities

The importance of a multi-modal system is underlined by the results of this ablation study. For that, the best performing network $\mathcal{F}_{32}^{\text{TF}}$ is modified predict the class of a gesture based on only one sensor modality. This is done for both, radar and ToF inputs, and then compared to the classification performance of the proposed multi-modal approach (Tab. 5.4). The results show that the performance is significantly improved, when using the fused information of both sensors. Relying only on the ToF signals is better than on only the radar data, where neither of the performance metrics surpasses 70%.

The benefit of the sensor fusion of ToF and radar can be seen especially for the gesture classes *piano* and *rub*. For small input sizes, the ToF does not deliver much information, as the reduction to 32×32 pixels blurs the depth image quality. In contrast to that, the networks can rely on the micro-Doppler signature of the FMCW radar, where still in the reduced input dimension, individual fingers can be identified moving up and down.

Table 5.4: Comparison of results based on the classification of multi-modal input (Radar+ToF) and the single modalities (Radar, ToF), all evaluated for network model $\mathcal{F}_{32}^{\text{TF}}$.

Metric	Radar	ToF	Radar+ToF
m_{acc}	68.8 %	87.7 %	97.9 %
m_{prec}	66.9 %	96.3 %	98.0 %
m_{rec}	64.9 %	90.4 %	97.9 %
m_{f1}	54.4 %	53.4 %	97.9 %

Embedded Deployment Results

After the evaluation on the GPU of all approaches, only the \mathcal{F}^{VI} and \mathcal{F}^{TF} network graphs are frozen and used for embedded inference. Only these architectures comply with the model requirements of all hardware accelerators.

The main advantage of these approaches is the application-oriented design that allows the networks to be deployed on multiple embedded accelerators—they are not bound to a single hardware. This is due to the simple model architecture that only relies on supported operations and omitting operations such as 3D convolutions or LSTM cells (Tab. 4.1.2). As a consequence of the slim models, fewer weights have to be learned during training, and stored and convolved during inference. The full precision model size of the largest proposed approach is only $M_{32}^{\text{TF}} = 967$ KB. The quantization further improves applicability, because the model size is now compressed to only 250 KB.

Counter-intuitively, the inference of the larger model \mathcal{F}^{TF} on the *Intel NCS* and the *NVIDIA Jetson nano* is faster than the execution of the small model \mathcal{F}^{VI} (Tab. 5.5). The reason is the faster execution of standard 3×3 convolutions, instead of the atypical 1×3 filters in \mathcal{F}^{VI} . The hardware accelerators are optimized for inferring with standard convolutions [55].

Table 5.5: Performances of the GCN architectures on various embedded CNN accelerators. Inference times t are averaged over 50 forward passes and M denotes the model sizes. Values are measured for the reduced input resolution S (32×32 , 32×32). *due to unsupported operations. \ddagger *uint8*-quantized values. \dagger *float16*-quantized values.

Model	EdgeTPU		NCS		Nano	
	t	M	t	M	t	M
$\mathcal{A}_{32}^{\text{VI}}$	21.1 ms	169 KB \ddagger	11.4 ms	316 KB \dagger	20.3 ms	0.67 MB
$\mathcal{A}_{32}^{\text{TF}}$	38.7 ms	250 KB \ddagger	5.4 ms	459 KB \dagger	16.5 ms	0.97 MB
3DCNN [90]	n/a*	n/a*	n/a*	n/a*	27.4 ms	8.0 KB

5.4.2 Comparison to Literature

Authors of [90] propose a network architecture for fusing multi-modal input data for gesture classification. The re-implemented version of their architecture is denoted as 3DCNN in the comparisons. It is trained on the GCN dataset and evaluated on the NVIDIA Jetson nano accelerator, which is the only device supporting all neural operations used in 3DCNN. Whereas the 3DCNN model is implemented without quantization of weights in the original paper, here, a quantized version of the graph is used for the comparison on the CNN accelerator. The model requires 10.86 MB of memory in full precision and 8.0 MB in the optimized version for the NVIDIA Jetson nano.

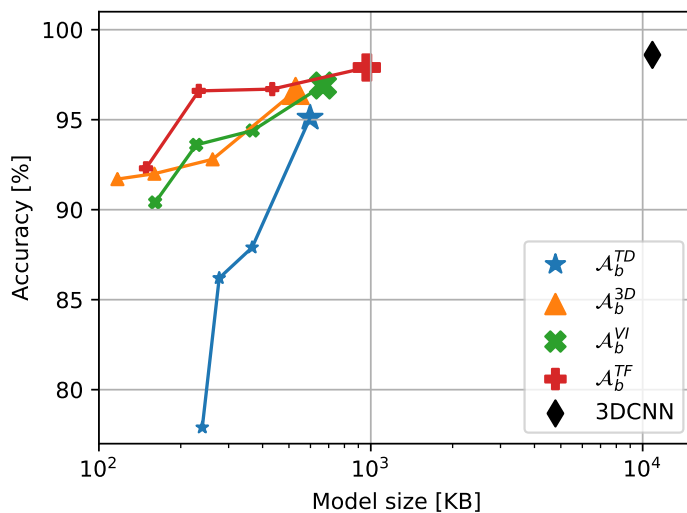


Figure 5.5: Classification accuracies over the model sizes of the network architectures from Section 5.3, compared to 3DCNN [90]. Each of the investigated architectures is evaluated with the scaling parameter $b \in \{4, 8, 16, 32\}$, describing an ever increasing amount of convolutional filters (Eq. 5.5). The colored Pareto-fronts are described by the best values out of three evaluation runs for each network variant.

Compared to 3DCNN [90], the proposed network \mathcal{F}_{32}^{TF} shows similar classification performance: The accuracy is -0.7% , the precision -0.4% , the recall -0.4% and the F1-score -0.4% (Tab. 5.3).

Compared to 3DCNN, \mathcal{F}_4^{TF} achieves an improvement of 69 times in the relation between accuracy and model size m_{acc}/M .

The plot in Fig. 5.5 highlights the trade-off between accuracy and model size, where the model size is on a logarithmic scale. Each colored line represents one

model family’s performance on the GPU evaluations in full precision. The black diamond shows the performance of 3DCNN [90].

5.5 Conclusion

In this chapter, a intrinsically privacy-aware gesture recognition system based on a radar and a ToF sensor is proposed.

Current gesture sensing solutions typically rely on camera sensors that may violate privacy standards [20, 134]. Moreover, many solutions tend to large model sizes that are infeasible to deploy on resource-constrained embedded accelerators. A further restriction of the embedded hardware is the high number of unsupported operations that limit the applicability of many state-of-the-art networks (Tab. 4.1).

The proposed solution employs a lightweight, two-stage algorithm that first transforms the spatio-temporal 4D data of each sensor modality to a 3D tensor with a fixed shape. The gesture, now in the form of 3D data, is then classified by a GCN with one input branch for each sensor modality. The gesture recognition networks are designed for the deployment on the *Google EdgeTPU*, the *Intel NCS2* and *NVIDIA Jetson nano*, avoiding unsupported operations, such as recurrent layers or 3D convolutions.

The largest proposed network $\mathcal{F}_{32}^{\text{TF}}$ achieves equal classification performance as 3DCNN [90] with only 8.9% of the model size. On the low-end of the model sizes, $\mathcal{F}_4^{\text{TF}}$ only uses 149 KB of memory while still performing robustly (92.3% accuracy). Thus, the proposed network models can even be deployed on resource-constrained embedded accelerators in the performance range of the *Google EdgeTPU*.

The main aspects of this chapter can be summarized as follows:

- Introduction of a lightweight, yet robust, multi-modal system for hand gesture recognition. The system relies on combined radar and ToF sensory data only. It offers variants that meet the different model requirements of state-of-the-art embedded accelerators.
- Assurance of privacy by not relying on camera sensors, thus, addressing the growing concerns about internet-of-things devices spying on the private life of customers.
- A system design for edge devices with limited compute capabilities. Accordingly, the proposed CNN models use standard neural network operations with few parameters only.

6 Conclusion

6.1 Summary

The thesis introduces a new approach for the automotive sensor fusion of Level 2/2+ cars. The topic of sensor fusion is positioned after the raw data processing, which is described in the preliminary signal processing chapter (Chapter 2), and before the mapping task, which integrates sensory information over time in order to create an environmental map. With the focus on current market trends towards a massive growth in partially automated vehicles, especially the sensor fusion of camera and radar systems is treated. Those sensor systems are, contrary to most of the research regarding sensor fusion, increasingly distributed and inhibit large portions of the pre-processing chain. Thus, a central fusion ECU has to cope with pre-processed feature-level data.

Traditional sensor fusion in cars with low automation levels is the combination of object- or track-level data. This environmental model allows the system to build up knowledge about distinctive objects, but not the area in between those objects. A typical environmental model, where the system can deduct, where it is drivable and where it is not, is the occupancy grid. There, the surrounding area is subdivided into small, uniform, checkerboard-like cells, with each cell described by the probability of it being occupied by an obstacle or being free. Usually, the data source for generating occupancy grids is the raw data of a precise ranging sensor, such as a lidar.

However, in mass-produced, low-cost vehicles of automation Level 2/2+, only camera and radar sensors are available and there is no access to the raw sensor data because of the distributed architecture. Thus, this work aims to fill this gap by leveraging the feature-level sensor data to create a meaningful, dense environmental model in form of an occupancy grid.

In Chapter 3, a framework for fusing multiple occupancy grids with neural networks is introduced, namely the fully convolutional fusion network (FC-FN), the auto-encoder fusion network (AE-FN) and the auto-encoder with bypass fusion network (AEB-FN). These architectures process sparsely populated occupancy grids of the individual sensors of different modalities to a fused representation. In the fused occupancy grid, all the information of the incoming OGs are combined and, additionally, an estimation about the drivable free-space is performed.

The basic working principle is a special form of neural network training, where the ground truth occupancy grid is based on data from a lidar sensor. Thus, during

6 Conclusion

the training phase, the networks learn to translate the inputted occupancy grids to a lidar-like representation. Simultaneously, the networks are trained to recognize the underlying structures of the scene, based on the radar and camera inputs.

The benefits of this approach are:

- Lidar-like environmental model from low-cost sensors available in mass-produced cars.
- Free-space estimation based on the underlying structural components and surroundings.
- Instantaneous occupancy grid with an estimate for the drivable space, without the need for temporal integration of traditional mapping.

The grid fusion networks require heavy computations for real-time performance, especially when considering large, high resolution occupancy grids. In order to make the neural networks applicable on automotive grade microcontrollers, the network architecture is designed with respect to the hardware. The neural architecture search (NAS) approach in Chapter 4 optimizes the network models according to a fitness function. Maximizing this fitness function incorporates a multi-objective optimization that favors slim models that simultaneously produce a good fusion quality. With the application of the NAS to the grid fusion task, a family of grid fusion networks is proposed in this work, namely the **GridFuN**. They are designed to run on a low-power CNN accelerator, which can be expected to be included in a similar form in future automotive microcontrollers. The **GridFuN** models scale with the model size, from well-performing slim models to big high-performance models. They describe a Pareto-front of model size vs. fusion quality and the discovered models show superior performance to comparable architectures.

The paradigm of sensor fusion networks with multiple input streams for multiple sensor modalities is adopted to another automotive sensor fusion use-case: a robust gesture recognition system that relies on ToF and radar data only. The crux of this use-case is the compression of typically large vision-based networks for gesture classification, to lightweight models that are applicable on constrained embedded devices. This is achieved with the gesture recognition networks (GCN) by two major adaptations: on the one hand, the choice of sensor modalities reduces the required processing power, as image processing is computationally more costly, than radar or ToF signal processing. On the other hand, with an optimized input data pipeline, the 4D spatio-temporal data is reduced to three dimensions, allowing the application of standard neural operations. The GCN compares to similar approaches from literature, but utilizing only 8.9% of the model size.

6.2 Publications

In total, four conference papers have been published throughout this thesis, out of which two were presented with awards:

1. Paper [10], discussed in Chapter 3.3: *Deep Grid Fusion of Feature-Level Sensor Data with Convolutional Neural Networks*. Awarded with the **Best Student Paper Award** in Graz during the IEEE ICCVE 2019.
2. Paper [9], discussed in Chapter 3.4.3: *Hardware-Aware Grid Fusion Networks for Automotive*. Presented in Nuremberg during the ewC 2020.
3. Paper [11], discussed in Chapter 4: *Neural Architecture Search for Automotive Grid Fusion Neural Networks*. Awarded with the **Best Paper Award** online¹ during the IEEE ICMLA 2020.
4. Paper [8], discussed in Chapter 5: *Sensor Fusion Neural Networks for Gesture Recognition on Low-Power Edge Devices*. Presented online during the ICAART 2021.

6.3 Outlook

The automotive market is ahead of two major changes that might disrupt not just traditional car manufacturers, but whole economic ecosystems, as the developed hierarchy of OEMs and suppliers will be scrambled up.

On the one hand, the shift from cars with combustion engines towards electric vehicles is already taking place and visible in the beginning of the 2020s. Growing regulatory requirements for low emissions thrive the development and production of electric vehicles, which rely on significantly less components than combustion engines. Thus, suppliers of those parts will be rendered redundant. Overall, the change introduced by the shift towards emission-free vehicles is large, but compared to the advances in automated driving, minor.

On the other hand, automated driving technology has the potential to disrupt complete economies and societies. The latter, because once autonomous driving is established, numerous jobs will be rendered obsolete, starting with the large number of truck and cab drivers. The introduction of increasingly automated vehicles is slow and incremental, but can be anticipated in the following three stages. The scope of this work is targeting the short- and mid-term perspective, where a still distributed system architecture makes it hard for a fusion ECU to access raw sensor data.

¹The conference in Miami was canceled due to the pandemic situation.

Short-Term Perspective (until 2025)

Coming from the strictly distributed system architecture, the development is limited to the paradigm of *one function for one ECU*. In some cases, this will evolve to merging some ECUs, but this is still limited to a few functions. In total, there will be more than 100 ECUs distributed in the vehicles [18].

This hardware forms the base for driver assistance systems of Level 1 and 2 for mass-market penetration. The assistance systems evolve incrementally, for example the *Lane Departure Warning* to *Lane Keep Assist* to *Lane Keep Steer* systems. Typically, these algorithms rely on feature-level camera and radar data.

Mid-Term Perspective (2025 - 2030/2035)

In these years, the vehicle architecture will evolve to a domain centralization. Here, five main domains will be created (powertrain, safety, sensing, comfort and infotainment), each of which will be driven by a domain-specific controller. Even though this reduces the number of ECUs in the vehicle, still, around 30–60 ECUs are needed in this case [18]. However, within the *sensing* domain, the distributed system architecture further remains.

With this hardware setup, more advanced driver assistance systems are enabled, starting with more sophisticated systems of Level 2 (Level 2+), or Level 3. In general, the carmakers favor incrementally enhance Level 2 systems, than introducing a Level 3 system, in order to avoid the trap of promoting the driver's ability to become disengaged from driving [18]. Rather they will advise that the drivers may take their hands and eyes off, but remain engaged.

Long-Term Perspective (2030/2035 onward)

In a final step towards a system architecture for fully automated vehicles, all the domain controllers are centralized into one high-performance computer. It is expected to reduce the number of required ECUs down to 20–45 units [18]. The consequence of central computing is that the sensors are connected directly to the central controller, instead of the intermediate ECUs within smart sensors. Thus, to transmit raw data streams of multiple sensors, the vehicle network requires a new level of bandwidth for this architecture.

The central processor requires dramatically increasing processing power, as the number of computationally expensive neural networks will increase for the assistance systems of L3, L4 and L5. Initially, neural networks are used for computer vision tasks in a single camera module, but with more sensors integrated in the systems and requirements to a robust environmental perception, leverage manifold networks. On the one hand, the perception of each modality and the fusion of multi-modal data can be enhanced with ML. On the other hand, it is needed to have multiple perception pipelines in parallel, in order to robustify the system. To achieve this, Mobileye and

Tesla separately proposed systems (L3, L4, L5) that use various perception engines simultaneously, each with own, separate ML algorithms [7, 70].

Future Works

In the context of the outlook described above, the following investigations are next to be done with respect to sensor fusion neural networks:

- *Expansion of the sensor modalities to ultrasonic sensors.* Especially for short-range applications, such as parking lot detection, parking assistance and automated parking, low-cost ultrasonic sensors could be leveraged to higher value for the vehicular perception. It could be investigated, how beneficial the inclusion of a third modality would be to the perception system.
- *Integration into mapping task.* Whereas mapping is not needed for low automation levels, it can be anticipated that integrating the output of the proposed sensor fusion networks into the mapping, especially for dense urban traffic situations, could be beneficial for a faster assessment of the situation.
- *Extension of the NAS to search for macro-architecture.* Generally, the performance of the NAS was constrained by the limited search space on the micro-architecture. Extending the search on a macro-architectural level, too, would result in even better performing network models.

Further interesting research topics, such as 3D imaging, can be anticipated with the use of the special training paradigm applied in this thesis. That is to train a network with expensive data from various sources in order to generate high-quality data from cheap sensors in the deployment phase, where not all data sources are available.

Bibliography

- [1] *Fundamentals of Radar*, chapter 4, pages 93–115. John Wiley & Sons, Ltd.
- [2] Free space computation using stochastic occupancy grids and dynamic programming. In *Workshop on Dynamical Vision, ICCV, Rio de Janeiro, Brazil*, volume 20. Citeseer, 2007.
- [3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [4] Mahdi Abavisani, Hamid Reza Vaezi Joze, and Vishal M. Patel. Improving the performance of unimodal dynamic hand-gesture recognition with multimodal training. *CoRR*, abs/1812.06145, 2018.
- [5] M. Aeberhard and N. Kaempchen. High-level sensor data fusion architecture for vehicle surround environment perception. 2011.
- [6] Md. Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C. Van Esesn, Abdul A. S. Awwal, and Vijayan K. Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *CoRR*, abs/1803.01164, 2018.
- [7] Mobileye Annon Shashua. Engines powering l2+ to l4, 2020.
- [8] G. Balazs, M. Chmurski, W. Stechele, and M. Zubert. Sensor fusion neural networks for gesture recognition on low-power edge devices. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pages 141–150. INSTICC, SciTePress, 2021.

- [9] G. Balazs and N. Leteinturier. Hardware-aware grid fusion networks for automotive. In *embedded world Conference 2020 Proceedings*, 2020.
- [10] G. Balazs and W. Stechele. Deep grid fusion of feature-level sensor data with convolutional neural networks. In *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, pages 1–6, 2019.
- [11] G. Balazs and W. Stechele. Neural architecture search for automotive grid fusion networks under embedded hardware constraint. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020.
- [12] Yaakov Bar-Shalom and Edison Tse. Tracking in a cluttered environment with probabilistic data association. *Automatica*, 11(5):451–460, 1975.
- [13] Dan Barnes, Matthew Gadd, Paul Murcutt, Paul Newman, and Ingmar Posner. The oxford radar robotcar dataset: A radar extension to the oxford robotcar dataset. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Paris, 2020.
- [14] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. Similarity Matching in Computer Vision and Multimedia.
- [15] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. 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 459–468, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.
- [16] J. Berger. *Statistical Decision Theory: Foundations, Concepts, and Methods*. Springer Series in Statistics. Springer New York, 2013.
- [17] S.S. Blackman. *Multiple-target Tracking with Radar Applications*. Artech House radar library. Artech House, 1986.
- [18] Pierrick Boulay. From advanced driver-assistance systems (adas) to automated driving, 06 2020.
- [19] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

- [20] Enea Ceolini, Gemma Taverni, Lyes Khacef, Melika Payvand, and Elisa Donati. Sensor fusion using emg and vision for hand gesture classification in mobile applications. In *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE, 2019.
- [21] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, pages 262–263, 2016.
- [22] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DianNao Family: Energy-Efficient Hardware Accelerators for Machine Learning. *Commun. ACM*, 59(11):105–112, October 2016.
- [23] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *CoRR*, abs/1710.09282, 2017.
- [24] Kyle Hyatt Chris Paukert. Tesla autonomy investor day: What we learned, what we can look forward to. *Roadshow by cnet*, 2019. <https://www.cnet.com/roadshow/news/teslas-autonomy-investor-day-recap/>, Accessed: 2021-02-15.
- [25] Bryan Clarke, Stewart Worrall, Graham Brooker, and Eduardo Nebot. Sensor modelling for radar-based occupancy mapping. pages 3047–3054, 10 2012.
- [26] 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 (CVPR)*, 2016.
- [27] I. Cumming and F. Wong. Digital processing of synthetic aperture radar data: Algorithms and implementation. 2005.
- [28] Fred Daum. Multitarget-multisensor tracking: Principles and techniques [book review]. *Aerospace and Electronic Systems Magazine, IEEE*, 11:41–, 03 1996.
- [29] A. P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *The Annals of Mathematical Statistics*, 38(2):325–339, 1967.
- [30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

- [31] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.
- [32] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, June 1989.
- [33] A. Elfes and L. Matthies. Sensor integration for robot navigation: Combining sonar and stereo range data in a grid-based representataion. In *26th IEEE Conference on Decision and Control*, volume 26, pages 1802–1807, Dec 1987.
- [34] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A Survey. *arXiv e-prints*, page arXiv:1808.05377, Aug 2018.
- [35] Patrick Emami, Panos M. Pardalos, Lily Elefteriadou, and Sanjay Ranka. Machine learning methods for solving assignment problems in multi-target tracking. *CoRR*, abs/1802.06897, 2018.
- [36] O. Erdinc, P. Willett, and Y. Bar-Shalom. Probability hypothesis density filter for multitarget multisensor tracking. In *2005 7th International Conference on Information Fusion*, volume 1, pages 8 pp.–, 2005.
- [37] Hannes Estl. Paving the way to self-driving cars with advanced driver assistance systems. *SAE International*, 2020. <https://www.sae.org/news/2020/12/rise-of-sae-level-2>, Accessed: 2021-02-15.
- [38] Mark Everingham, Luc van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.
- [39] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. *CoRR*, abs/1604.06573, 2016.
- [40] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.
- [41] Rubén García, Olivier Aycard, Trung-Dung Vu, and Malte Ahrholdt. High level sensor data fusion for automotive applications using occupancy grids. pages 530–535, 12 2008.
- [42] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

- [43] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.
- [44] Google. Google EdgeTPU Documentation, 2020.
- [45] Karl Granström and Marcus Baum. Extended object tracking: Introduction, overview and applications. *CoRR*, abs/1604.00970, 2016.
- [46] David Gschwend. Netscope cnn analyzer, 2015.
- [47] V. Gupta, G. Singh, A. Gupta, and A. Singh. Occupancy grid mapping using artificial neural networks. In *2010 International Conference on Industrial Electronics, Control and Robotics*, pages 247–250, Dec 2010.
- [48] Christian Hartmann. Automated driving at a new level: the audi ai traffic jam pilot. *Audi MediaCenter*, 2017.
- [49] Caner Hazirbas, Lingni Ma, Csaba Domokos, and D. Cremers. Fusetnet: Incorporating depth into semantic segmentation via fusion-based cnn architecture. In *ACCV*, 2016.
- [50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.
- [52] Shaun Helman and Oliver Carsten. What does my car do? *PACTS*, 2019.
- [53] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185–203, 1981.
- [54] John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset. *arXiv preprint arXiv:2006.14480*, 2020.
- [55] Andrew Howard and Suyog Gupta. Introducing the Next Generation of On-Device Vision Models: MobileNetV3 and MobileNetEdgeTPU. ai.googleblog.com/2019/11/introducing-next-generation-on-device.html. Accessed: 13-11-2020.
- [56] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

- [57] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.
- [58] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [59] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The apolloscape dataset for autonomous driving. *CoRR*, abs/1803.06184, 2018.
- [60] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016.
- [61] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.
- [62] Intel. Intel Neural Compute Stick 2 Documentation, 2020.
- [63] Mordor Intelligence. Automotive camera market - growth, trends, covid-19 impact, and forecasts (2021 - 2026). <https://www.mordorintelligence.com/industry-reports/automotive-camera-market>. Accessed: 2021-02-15.
- [64] Road vehicles — Functional safety — Part 3: Concept phase. Standard, International Organization for Standardization, Geneva, CH, December 2018.
- [65] Vijay John, Seiichi Mita, and Hossein Tehrani. Sensor fusion of intensity and depth cues using the chinet for semantic segmentation of road scenes. 06 2018.
- [66] Vikas Joshi, R.Anitha Nithya, K. Takayuki, Naveen Prathapaneni, and L.V. Subramaniam. Information fusion based learning for frugal traffic state sensing. pages 2826–2832, 08 2013.
- [67] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.
- [68] Nico Kaempchen and Klaus Dietmayer. Data synchronization strategies for multi-sensor fusion. 02 2021.
- [69] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.
- [70] Andrej Karpathy. Ai for full-self driving at tesla, 04 2020.

- [71] Sungwoong Kim, Ildoo Kim, Sungbin Lim, Woonhyuk Baek, Chiheon Kim, Hyungjoo Cho, Boogeon Yoon, and Taesup Kim. Scalable neural architecture search for 3d medical image segmentation. *CoRR*, abs/1906.05956, 2019.
- [72] Philip Koopman. A strategy for evolving self-driving car safety assurance, 11 2019. Keynote Speech by Philip Koopman at the 8th IEEE International Conference for Connected Vehicles and Expo (ICCVE), Graz, Austria.
- [73] Okan Köpüklü, Neslihan Köse, and Gerhard Rigoll. Motion fused frames: Data level fusion strategy for hand gesture recognition. *CoRR*, abs/1804.07187, 2018.
- [74] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).
- [75] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.
- [76] Mingkang Li, Zhaofei Feng, Martin Stolz, Martin Kunert, Roman Henze, and Ferit Küçükay. High resolution radar-based occupancy grid mapping and free space detection. In *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems - Volume 1: VEHITS*, pages 70–81. INSTICC, SciTePress, 2018.
- [77] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7337–7345, 2019.
- [78] Dharmendra Lingaihah. Kalman filtering: Theory and practice using matlab, 2nd ed [book review]. *Circuits and Devices Magazine, IEEE*, 19:37– 38, 08 2003.
- [79] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *CoRR*, abs/1901.02985, 2019.
- [80] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.
- [81] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

- [82] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.
- [83] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017.
- [84] Siegfried Martens, Gail A. Carpenter, and Paolo Gaudiano. Neural sensor fusion for spatial visualization on a mobile robot 1. 2008.
- [85] Stephan Matzka and Richard Altendorfer. *A Comparison of Track-to-Track Fusion Algorithms for Automotive Sensor Fusion*, volume 35, pages 69–81. 03 2009.
- [86] A. Meta. Signal processing of fmcw synthetic aperture radar data. 2006.
- [87] Michael Meyer. Automotive radar dataset for deep learning based 3d object detection. 01 2019.
- [88] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. In Robert Kozma, Cesare Alippi, Yoonsuck Choe, and Francesco Carlo Morabito, editors, *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam: Elsevier, 2018.
- [89] Masha Mikhal and Mykel J. Kochenderfer. Convolutional neural network information fusion based on dempster-shafer theory for urban scene understanding. 2017.
- [90] Pavlo Molchanov, Shalini Gupta, Kihwan Kim, and Kari Pulli. Multi-sensor system for driver’s hand-gesture recognition. 05 2015.
- [91] Pavlo Molchanov, Xiaodong Yang, Shalini Gupta, Kihwan Kim, Stephen Tyree, and Jan Kautz. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. pages 4207–4215, 06 2016.
- [92] H. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Mag.*, 9:61–74, 1988.
- [93] Hans Moravec and Alberto Elfes. High resolution maps from wide angle sonar. volume 2, pages 116 – 121, 04 1985.
- [94] Euro NCAP. Euro ncap 2025 roadmap: In pursuit of vision zero. 2017.

- [95] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kotschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *International Conference on Computer Vision (ICCV)*, 2017.
- [96] D. Nuss, M. Thom, A. Danzer, and K. Dietmayer. Fusion of laser and monocular camera data in object grid maps for vehicle environment perception. In *17th International Conference on Information Fusion (FUSION)*, pages 1–8, 2014.
- [97] NVIDIA. NVIDIA Jetson Nano Documentation, 2020.
- [98] Simon O’Callaghan and Fabio Ramos. Gaussian process occupancy maps. *I. J. Robotic Res.*, 31:42–62, 01 2012.
- [99] Simon O’Callaghan, Fabio Ramos, and Hugh Durrant-Whyte. Contextual occupancy maps using gaussian processes. pages 1054 – 1060, 06 2009.
- [100] David Olson and Dursun Delen. *Advanced Data Mining Techniques*. 01 2008.
- [101] Abhishek Patil, Srikanth Malla, Haiming Gang, and Yi-Ting Chen. The H3D dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes. *CoRR*, abs/1903.01568, 2019.
- [102] S. M. Patole, M. Torlak, D. Wang, and M. Ali. Automotive radars: A review of signal processing techniques. *IEEE Signal Processing Magazine*, 34(2):22–35, 2017.
- [103] S. O. Piper. Fmcw linearizer bandwidth requirements. In *Proceedings of the 1991 IEEE National Radar Conference*, pages 142–146, 1991.
- [104] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, Sep 2020.
- [105] Tiana Rakotovo Andriamahefa. *Integer Occupancy Grids : a probabilistic multi-sensor fusion framework for embedded perception*. Theses, Université Grenoble Alpes, February 2017.
- [106] Karthik Ramasubramanian. mmwave radar for automotive and industrial applications. *Texas Instruments*, 2017.
- [107] Fabio Ramos and Lionel Ott. Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent. *The International Journal of Robotics Research*, 35:1717–1730, 12 2016.
- [108] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018.

- [109] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc V. Le, and Alex Kurakin. Large-scale evolution of image classifiers. *CoRR*, abs/1703.01041, 2017.
- [110] Stephan Reuter. *Multi-object tracking using random finite sets*. PhD thesis, Universität Ulm, 2014.
- [111] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators. *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep 2019.
- [112] Rockchip. Rockchip Documentation, 2020.
- [113] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.
- [114] Taxonomy SAE. Definitions for terms related to driving automation systems for on-road motor vehicles. *SAE Standard J3016*, 2016.
- [115] Raghavender Sahdev. Free space estimation using occupancy grids and dynamic object detection.
- [116] Tonmoy Saikia, Yassine Marrakchi, Arber Zela, Frank Hutter, and Thomas Brox. Autodispnet: Improving disparity estimation with automl. *CoRR*, abs/1905.07443, 2019.
- [117] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014.
- [118] K. Siu, D. M. Stuart, M. Mahmoud, and A. Moshovos. Memory requirements for convolutional neural network hardware accelerators. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 111–121, 2018.
- [119] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015.
- [120] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.
- [121] Alan Steinberg and C.L. Bowman. Revision to the jdl data fusion model. *Handbook of Multisensor Data Fusion*, 01 2011.
- [122] P Sun, H Kretzschmar, X Dotiwalla, A Chouard, V Patnaik, P Tsui, J Guo, Y Zhou, Y Chai, B Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. arxiv 2019. *arXiv preprint arXiv:1912.04838*.

- [123] Shuyang Sun, Jiangmiao Pang, Jianping Shi, Shuai Yi, and Wanli Ouyang. Fishnet: A versatile backbone for image, region, and pixel level prediction. *CoRR*, abs/1901.03495, 2019.
- [124] Synopsys. *DesignWare EV61, EV62 and EV64 Processors Datasheet*, 5 2018.
- [125] Sean Szymkowski. Tesla will never achieve full self-driving capabilities, waymo ceo says. *Roadshow by cnet*, 2021. <https://www.cnet.com/roadshow/news/tesla-will-never-achieve-full-self-driving-capabilities-says-waymo-ceo/>, Accessed: 2021-02-15.
- [126] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018.
- [127] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.
- [128] S. Thrun. Learning occupancy grids with forward sensor models. 2002.
- [129] Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3D: generic features for video analysis. *CoRR*, abs/1412.0767, 2014.
- [130] Yohann Tschudi and Guillaume Girardin. Road to robots - sensors and computing for autonomous vehicle. In *Autonomous Vehicle Sensors Conference 2018*, 2018.
- [131] Darwin Ttito Concha, Helena Maia, Helio Pedrini, Hemerson Tacon, Andre Souza Brito, Hugo Chaves, and Marcelo Vieira. Multi-stream convolutional neural networks for action recognition in video sequences based on adaptive visual rhythms. pages 473–480, 12 2018.
- [132] Michelle Valente, Cyril Joly, and Arnaud de La Fortelle. Fusing laser scanner and stereo camera in evidential grid maps. *CoRR*, abs/1805.10046, 2018.
- [133] Vij Vikrant and Dr. Rajesh Mehra. Fpga based kalman filter for wireless sensor networks. *International Journal of Computer Technology and Applications*, 02, 01 2011.
- [134] Saiwen Wang, Jie Song, Jaime Lien, Ivan Poupyrev, and Otmar Hilliges. Interacting with soli: Exploring fine-grained dynamic gesture recognition in the radio-frequency spectrum. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 851–860. ACM, 2016.
- [135] Shenlong Wang, Min Bai, Gellert Mattyus, Hang Chu, Wenjie Luo, Bin Yang, Justin Liang, Joel Cheverie, Sanja Fidler, and Raquel Urtasun. Torontocity: Seeing the world with a million eyes. 12 2016.

- [136] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. *CoRR*, abs/1812.07179, 2018.
- [137] Y. Weng, T. Zhou, Y. Li, and X. Qiu. Nas-unet: Neural architecture search for medical image segmentation. *IEEE Access*, 7:44247–44257, 2019.
- [138] Rob Weston, Sarah H. Cen, Paul Newman, and Ingmar Posner. Probably unknown: Deep inverse sensor modelling in radar. *CoRR*, abs/1810.08151, 2018.
- [139] Hermann Winner, Stephan Hakuli, and Gabriele Wolf. *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort: mit 550 Abbildungen und 45 Tabellen*. Springer-Verlag, 2009.
- [140] Sascha Wirges, Tom Fischer, Jesus Balado Frias, and Christoph Stiller. Object detection and classification in occupancy grid maps using deep convolutional networks. *CoRR*, abs/1805.08689, 2018.
- [141] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. SqueezeSeg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3d lidar point cloud. *CoRR*, abs/1710.07368, 2017.
- [142] Huadong Wu, M. Siegel, R. Stiefelhagen, and Jie Yang. Sensor fusion using dempster-shafer theory. In *Frontier of instrumentation and measurement*, pages 7–12. IEEE, 2002.
- [143] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.
- [144] Kristie Young, Michael Regan, and Mike Hammer. Driver distraction: A review of the literature. *Distracted Driving*, 01 2003.
- [145] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving video database with scalable annotation tooling. *CoRR*, abs/1805.04687, 2018.
- [146] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *CoRR*, abs/1906.05113, 2019.
- [147] Zhao Zhong, Junjie Yan, and Cheng-Lin Liu. Practical network blocks design with q-learning. *CoRR*, abs/1708.05552, 2017.

- [148] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.
- [149] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.