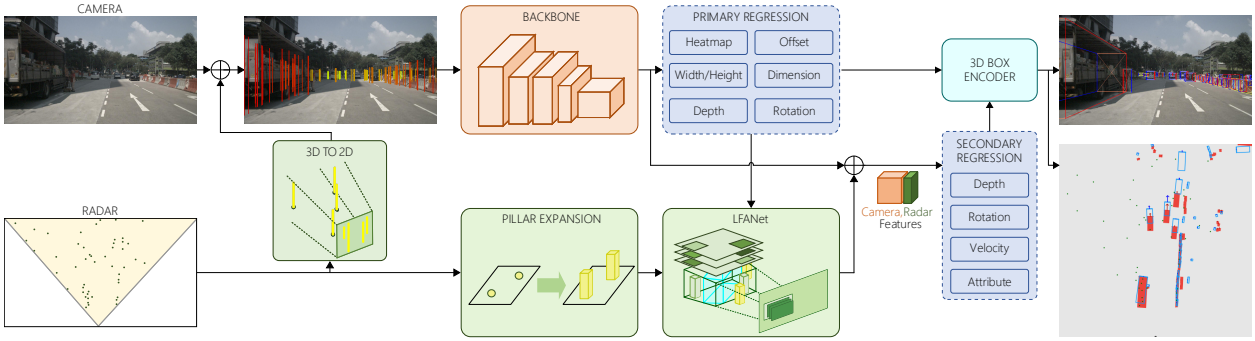
CHALMERS
UNIVERSITY OF TECHNOLOGY

University of Stuttgart
Germany



Camera-Radar Sensor Fusion using Deep Learning

A frustum proposal-based 3D object detection network for multi-stage fusion in autonomous driving

Master's thesis in Systems, Control and Mechatronics & Engineering Cybernetics

Johannes Kübel, Julian Brandes

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022

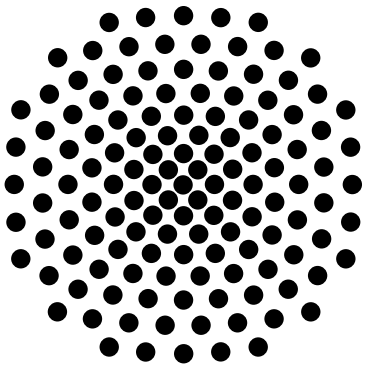
Institute of Systems Theory and
Automatic Control
UNIVERSITY OF STUTTGART

Master's thesis 2022

Camera-Radar Sensor Fusion using Deep Learning

A frustum proposal-based 3D object detection network for multi-stage fusion in autonomous driving

CHALMERS
UNIVERSITY OF TECHNOLOGY



University of Stuttgart
Germany

Department of Electrical Engineering
Chalmers University of Technology

Institute of Systems Theory and Automatic Control
University of Stuttgart

Gothenburg, Sweden 2022

Camera-Radar Sensor Fusion using Deep Learning
A frustum proposal-based 3D object detection network for multi-stage fusion in autonomous driving

Supervisors: Maryam Fatemi, Zenseact AB
Mahandokht Rafidashti, Zenseact AB
Advisors: Yibo Wu, Department of Electrical Engineering,
Chalmers University of Technology
Patricia Pauli, Institute of Systems Theory and Automatic Control,
University of Stuttgart
Examiners: Henk Wymeersch, Department of Electrical Engineering,
Chalmers University of Technology
Frank Allgöwer, Institute of Systems Theory and Automatic Control,
University of Stuttgart

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Submitted to the University of Stuttgart
Institute of Systems Theory and Automatic Control
University of Stuttgart
DE-70511 Stuttgart
Telephone +49 711 685 67736

Gothenburg, Sweden 2022

Abstract

To be able to guarantee safe navigation through an unknown environment, objects in the three dimensional surroundings of the ego vehicle, including their velocities, have to be detected accurately and robustly at different weather and lightning conditions. This work deals with the sensor fusion of two complementary sensor modalities radar and camera using deep machine learning within autonomous driving. The goal is to combine the advantages of both sensor modalities and to compensate for their disadvantages. Common sensor fusion algorithms often use well-researched algorithms such as Kalman filters. However, with the increasing popularity of machine learning algorithms, the interest in deep learning based fusion of different sensor modalities is deepened. After an extensive literature research, this work motivates, implements, and evaluates two major modifications to a state-of-the-art network within deep learning camera radar sensor fusion. The proposed adaptions are a sub-network that learns from a selected part of a point cloud as well as the introduction of early fusion to the network. The training runs of all proposed architectures are performed on the popular *nuScenes* dataset [1]. The code is available at https://github.com/brandesjj/centerfusionpp.

Keywords: deep machine learning, sensor fusion, camera, radar, 3D object detection, point cloud based network, proposal-based fusion, frustum proposal, multi-stage fusion

Acknowledgements

We want to thank our supervisors and all team members at Zenseact that helped us in the process of creating this work. Furthermore, I want to thank the Baden-Württemberg STIPENDIUM for supporting me, Johannes Kübel, with a scholarship over the duration of the thesis.

Johannes Kübel and Julian Brandes, Gothenburg, August 2022

Contents

List of acronyms

ADAS	advanced driver assistance system
AD	autonomous driving
AP	average precision
AV	autonomous vehicle
BB	bounding box
BEV	bird's-eye view
DCL	deformable convolutional layer
DLA	deep layer aggregation
FMCW	frequency modulated continuous wave
FoV	field of view
IMU	inertial measurement unit
IoU	intersection over union
LiDAR	light detection and ranging
mAAE	mean average attribute error
mAOE	mean average orientation error
mAP	mean average precision
mASE	mean average scale error
mATE	mean average translation error
mAVE	mean average velocity error
NDS	nuScenes detection score
NMS	non-maximum suppression
NN	neural network
RCS	radar cross section
ReLU	rectified linear unit
ROI	region of interest

1
Introduction

This chapter introduces the reader to the research performed in this work, considering its motivation and goals alongside ethical, social and sustainability aspects.

1.1 Motivation

The origins of self-driving vehicles being a vision in industry and science reach far back [2]. Zenseact AB is a company founded to bring software products tackling the challenges of autonomous driving (AD) and advanced driver assistance systems (ADAS) to the global market. In AD and ADAS one of the main tasks is the perception of the surroundings of the ego vehicle. Perception is based on the measurements of sensors scanning the environment. Objects in the ego vehicle's proximity need to be detected and localized reliably.

In recent years, multiple sensor modalities have been proposed and further developed to help perceiving objects. Two sensors are especially promising, namely camera and radar, since they complement each other well and are both relatively cheap. A camera is typically good at distinguishing features in the visual space because of their higher spatial resolution [3], while being of poor quality in harsh environmental conditions as rain, fog or darkness [4]. In addition, they lack the ability of estimating the depth and velocity of objects without the use of temporal information [3]. A radar provides data that is relatively sparse in spatial dimensions while not being affected by rain, fog or darkness and can measure both – distance and velocity of reflecting objects in the environment [5].

As a consequence, new methods to fuse the data of camera and radar arouse, increasing interest within the automotive industry [3, 4]. Sensor fusion is usually performed using classical methods such as Kalman filters, but there are multiple strong assumptions to be made on the data in those methods. Due to the complexity of the problem, deep learning is introduced to learn the underlying correspondences from data. By approximating the relationship between data and object detection, deep learning networks solve the problem of sensor fusion. In this work, we propose a neural network that handles both camera and radar data and outputs 3D objects detections.

1.2 Research question

The main research question of this work is how to fuse radar and camera data effectively. In short, the problem is to explore the fusion of 2D camera and 3D

radar data in a deep learning based architecture. Therefore, we focus on improving the state-of-the-art use of deep learning in sensor fusion of camera and radar. A promising approach described in Chapter 4 is *CenterFusion* which is a proposal-based camera and radar fusion network for 3D object detection. Our contribution should improve the performance in object detection as well as velocity estimation w.r.t. the baseline and other approaches using sensor fusion of camera and radar, and especially approaches only using either of the two.

1.3 Ethical and social aspects

The ethics of autonomous vehicles (AVs) is a topic widely discussed recently in our society. Whilst most papers describe the challenges companies, engineers and developed algorithms are facing [6, 7, 8, 9, 10] some also consider the role of society and individuals w.r.t. the usage of self-driving cars [11, 12]. Autonomous driving requires a system that is as safe as possible since it directly involves humans in a high dynamic environment with potentially great casualties. As a consequence, the perception of the environment has to be as reliable as possible.

Since this work tackles the problem of sensor fusion for camera and radar data, there is a need to handle ethical and social aspects and include the reliability of the system in the evaluation of the results. Some challenges on the way towards autonomous driving are summarized in [7]. In this work, the need for safety is met by evaluating the reliability of the developed object detection algorithms as a final step. One has to ensure to capture and perceive at least all objects and obstacles within the environment of the car that are relevant for the safety of the passengers and fellow humans and animals. Equally strict requirements apply for security of the systems used which also includes the accessibility of the data even in case of accidents.

Privacy concerns come up when dealing with data collection and recording. Whenever data is used that might contain sensitive information of pedestrians this information should be protected. In this work, only data provided by the *nuScenes* dataset [1] is processed, where privacy concerns are tackled by blurring faces and license plates.

1.4 Sustainability aspects

With global warming in mind, sustainability becomes increasingly important to the automotive industry and personal transportation as well. The authors of [13] analyze the impact of self-driving automobility on carbon emission and compare them to conventional modes of transportation by conducting various simulations. By creating eight different scenarios for the future of personal transport, [13] compares modes of transportation including as well as excluding full self-driving vehicles. Both of the scenarios providing the biggest positive impact on carbon emissions involve autonomous driving.

Since the potential effects of AVs on greenhouse gasses are not certain yet, [14] summarizes the current state in literature. The use of AVs allows easy and fast travel

to all people, especially the elderly or disabled, which results in an increasing vehicle mile travel overall. Nevertheless, a positive effect on emission reduction is to be expected whose peak might be at 60–80% of AV penetration into the transportation network [14]. However, in [15] the authors conclude that there is still a lack of research regarding the sustainability aspects of AVs, especially compared to the amount of research performed on technological level.

The proposed work does not affect sustainability in a direct manner since it deals with the perception part of the autonomous driving software. In spite of that, one could argue that an increased performance in environment perception, especially regarding more robust detections at long distance, can also lead to more energy-efficient driving in AVs and lower risk of accidents, e.g. through early breaking and early sensor failure detection. Fewer accidents also result in less material loss due to broken cars and infrastructure and can contribute to a positive effect on the carbon footprint. In conclusion, a robust and well performing algorithm can have a positive impact on the sustainability of autonomous cars.

2
Theory

Machine learning has become of increasing interest in literature and research over the last decade. This especially applies to image based object detection. Recently, machine learning has also been applied to sensor fusion of different sensor modalities. Since this is the main goal of this work, this chapter first gives a brief introduction into machine learning and the sensors modalities radar and camera used in the scope of this work. Additionally, Section 2.3 introduces the general principles of sensor fusion using machine learning. Section 2.3.2 summarizes current approaches for the fusion of camera and radar data using deep learning in literature.

2.1 Machine Learning

This work deals with supervised deep learning which is a subarea of machine learning, see [16, 17, 18, 19] for further reading. In supervised deep learning a multi-layer artificial neural network (NN) receives a multi-dimensional input and computes an output. The output is compared to a target set by the supervisor. The NN is optimized w.r.t. a loss function, taking the error from the output to the target into account.

An NN consists of multiple elements that contribute to the learning mechanism. Firstly, the input to an NN is forwarded through nodes. Computing the output of a node given an input is called a forward-pass. Nodes can also take the output of another node as input. The concatenated forward-pass through all nodes12 can approximate any function from a finite-dimensional space [19]. In these nodes, weights are stored that are optimized with the following machine learning elements. The loss function computes the error to the corresponding target for the given input. The gradient of the loss function w.r.t. the weights of the nodes is used in the optimizer where the weights of the nodes are updated in a specific pattern. With a small enough learning rate and enough data, the loss function converges to a local minimum while iterating over the dataset multiple times. A fully trained network can be used at inference to evaluate a given input and give a prediction.

2.1.1 Node

Nodes consist of a main computation function that in the most cases are either a linear function or a convolutional function. Networks that use the node type with linear functions are called fully connected networks. Convolutional networks [20] use convolutional functions that are designed to share the weights of the node over

iterative calls of the forward-pass for a single input.

The output of each main computation is fed through an activation function. The value of the activation function is the output of the entire node.

Fully Connected

The input for nodes with linear functions as the main computation is weighted and added to a bias. The weights and bias are the learnable parameters. Note, formally one would address the bias when needed but most of the time all parameters are summarized by the term "weights" for simplicity.

A fully connected node receives the input $x = \begin{bmatrix} x_1 & x_2 & ... & x_n \end{bmatrix}^{\mathrm{T}}$ which is weighted by the weights $w = \begin{bmatrix} w_1 & w_2 & ... & w_n \end{bmatrix}^{\mathrm{T}}$ added to a bias b. The output y is the value of the activation function f

$$y = f\left(\sum_i w_i x_i + b\right). \tag{2.1}$$

Convolutional

Convolutional nodes stride over a 3D pixel grid applying a local 2D filter for a subset of the input at each stride. The filter multiplies all pixel values in the first two dimensions from the input with the corresponding pixel from the kernel and sums the results over the third dimension. Finally, a bias is added to all results of the convolution. The number of image-like slices in the third dimension of the pixel grid is called "channels". Convolutional nodes include hyperparameters in contrary to nodes with linear functions that need to be set in prior to training. The parameters are the kernel size, stride in pixel and number of filters as well as padding and dilation factors. The padding parameter is the amount of additional zero valued pixel that are added at each border of the input. The dilation factor is the distance in pixel of two neighboring kernel pixels.

Deformable Convolutional

Deformable convolutional nodes [21] are a special type of convolutional nodes where the offset between each kernel pixel is defined as an additional degree of freedom and therefore a learnable parameter.

Activation Function

A node contains an activation function additional to the main computation. The computed result for the given input to a node is the input to the activation function. The activation function is used to introduce nonlinearity to the node such that nonlinear functions can be approximated. There are many proposals of activation functions and the research into them is still an active field. Although, in the most cases one of the following functions is used. The sigmoid function is primarily used at the final nodes of a network for predicting the class of an object due to its mapping between 0 and 1. Its output can be interpreted as a probability of the input object

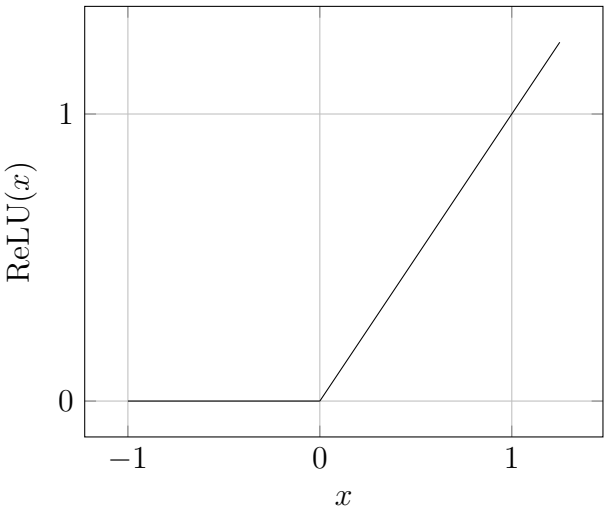


Figure 2.1: ReLU activation function.

to be in a given class. The rectified linear unit (ReLU) function is very common since it was shown to yield better performance [22, 23].

2.1.2 Layer

A layer in an NN combines multiple nodes that receive the same input. A convolutional layer typically only contains one node but fully connected layers are often build up from many nodes. The depth of the network is defined by the number of layers of the network. There are also other layers such as pooling layers.

Pooling layers are used after convolutional layers and they are typically used to reduce the shape of the tensor that is outputted from the convolutional node. There are different types of pooling, e.g. maximum, global or average. Maximum pooling is the most common one and it takes the maximal value in a local region of the tensor and yields a tensor with compressed information for each filter of the convolutional layer. Global pooling is the same as maximum pooling but over the whole tensor and it outputs only one value for each filter. The average pooling take the average of all local elements of the input tensor and returns a tensor with slightly less compressed information.

2.1.3 Loss function

The entire forward-pass through all layers in an NN results a prediction $\hat{x}$. The loss function $f(\hat{x}, x)$ computes the error from the prediction to the annotation x. The annotation x is the ground-truth which needs to be labeled before training the NN. The NN learns to minimize the loss function and therefore the error from the prediction to the ground-truth. There are many loss functions proposed in literature such as the mean absolute error – or L1 –, mean squared error – or L2 –, smooth-L1, focal loss [24] and cross-entropy. The loss functions are summarized in Table 2.1. For the focal loss α and β are hyperparameters. The cross-entropy loss can be either in binary or categorical version where the difference is that x is in $\{0, 1\}$ in the binary

case and a one-hot encoding vector in the categorical – or multi-label – case. All loss functions are usually averaged over a batch. Further research into loss functions is still ongoing.

Table 2.1: Popular loss functions.

Name	$f(\hat{x}, x)$
L1	$\|\hat{x} - x\|_1$
L2	$\|\hat{x} - x\|_2^2$
Smooth-L1	$\begin{cases} 0.5 \|\hat{x} - x\|_2^2, & \text{if } \|\hat{x} - x\|_1 < 1 \\ \|\hat{x} - x\|_1 - 0.5, & \text{otherwise} \end{cases}$
Focal [24]	$\begin{cases} (1-\hat{x})^\alpha \log(\hat{x}) & \text{if } x = 1 \\ (1-x)^\beta \hat{x}^\alpha \log(1-\hat{x}) & \text{otherwise} \end{cases}$
Cross-entropy	$-\sum_{c=1}^{C} x_c \log\left(e^{\hat{x}_c} / \sum_{j=1}^{C} e^{\hat{x}_j}\right)$

2.1.4 Optimizer

In general, loss functions are functions depending on the weights and therefore highly dimensional. During backpropagation the gradients of the loss function w.r.t. the weights in the NN are calculated. The essence of machine learning is to minimize the loss function by iterating over optimization steps of a numerical optimizer. The optimizer updates all trainable weights by following an update scheme. There are many optimizers in literature. [25] gives an overview of the most used optimizers. In this work, the Adam optimizer – first presented in [26] – is used and we are not looking into the effect of other optimizers. To speed up training, the data is usually grouped into batches after which an optimization step is applied. Note, numerical optimizers only guarantee to approximate local minima but not necessarily global ones.

When an NN is trained all weights that are set to be trainable are updated by the optimizer. Weights – or entire layers – that are not updated are called *frozen*.

2.1.5 Network architecture

On a high level there exists, especially in object detection NNs, three main components. Firstly, the *backbone* is usually designed to extract a rich representation of the input called *feature map* FM. For an NN that receives an image as an input the backbone usually downscales the size of the image but keeps the ratio for the feature map FM the same. Secondly, the *head* receives the feature map FM and extracts in multiple parallel shallow subnetworks a set of parameters for regression

or classification. Lastly, the *neck* is the link between the backbone and the head and usually upsamples the feature map FM in a learnable fashion. The output of the neck is the input of the head.

While training a network the backbone and neck are usually trained for a large amount of epochs since they are often very deep. The head is then often modified iteratively to achieve the best performance while the trained backbone and neck are used for its input. For this step, the backbone and neck are often frozen and only the head is trained. When the subnetworks in the head are converged, the entire network is usually fine-tuned over a small amount of epochs. The weights of any part of the network that are already trained can be saved at different points and loaded in for other experiments. A model used for this concept is called *pretrained* model.

For measuring how well the network is performing not only the loss function but also evaluation metrics are needed. For this work the mean average precision (mAP) and the nuScenes detection score (NDS) are used, see Section 3.4 for more details.

2.2 Sensor Technology

With increasing popularity and interest in autonomous driving in both, industry and research, the question on the choice of a reliable and precise sensor setup for environmental perception arises. The three most popular sensors are camera, radar and light detection and ranging (LiDAR). Table 2.2 compares the three modalities using different criteria. Choosing both, the camera and radar modality, one achieves the highest score for all the criteria with at least one of the sensors. In consequence, this sensor setup is a sensor setup with promising properties that will be investigated further in the following.

Table 2.2: Comparison of the sensor modalities camera, radar, and LiDAR for the use within the automotive industry according to [27, 28]. The valuation ranges from ○ being the worst to ● as the best score for the respective criteria.

	Camera	Radar	LiDAR
Spatial Resolution	●	◔	◕
Distance estimation	◑	●	●
Velocity measurement	◔	●	◔
Range	◑	●	◑
Weather robustness	◔	●	◑
Lighting robustness	◑	●	●
Cost	●	●	◔
Amount of data	◔	●	◔
Color estimation	●	○	◔
Lane Detection	●	○	◔
Obstacle Edge Detection	●	○	●

2.2.1 Radar

Frequency modulated continuous wave (FMCW) radars are the most popular type of radar sensor in the automotive industry. A radar sensor emits electromagnetic waves using at least one transmitter antenna [29]. When the waves hit an object, they are reflected and received by (at least one) receiver antenna. Comparing both signals, the transmitted and received waveforms allows one to calculate distance, radial velocity, azimuth angle and elevation of the reflector. For a more detailed explanation, please refer to [29]. The output of the radar sensor used in the scope of this thesis consists of a point cloud. Each of the points in the point cloud contains a measurement for the values in Table 2.3.

Table 2.3: Measurements of a single radar point in the radar point cloud for the data used in this work.

Value	Description
x	X-component of the position
z	Z-component of the position
v_x	X-component of the radial velocity
v_z	Z-component of the radial velocity
RCS	Radar Cross Section value

Radar cross section

Alongside with the distance and radial velocity, each radar point contains a measurements called the radar cross section (RCS) value. It is a measure for the detectability of the reflecting object displays how well the object reflects the electromagnetic waves. In consequence, this allows to draw conclusions on the material and size of an object in the environment.

2.2.2 Camera

One major advantage of the camera sensor is its high spatial resolution which enables it to capture small objects and objects at a great distance. However, a mono-camera setup does not allow for any direct measurement in the 3D space but is restricted to the 2D image plane. To link the 2D camera measurement and the 3D space, one can use a model of the camera. The most common model used is the so-called pinhole camera model which will be briefly described in the following. For more information on this and other camera models please refer to [30, 31].

Pinhole camera model

The pinhole camera model links the coordinates of a point in the 3D space to the corresponding point in the 2D image plane. The camera aperture is modeled as a single point, the so called pinhole which represents the origin of the camera

coordinate frame C. A scene point ${\mathbf{x}'}_{\mathrm{C}}$ in the 3D space, expressed in the camera frame C, is defined as

$$ {\mathbf{x}'}_{\mathrm{C}} = \begin{bmatrix} x'_C & y'_C & z'_C \end{bmatrix}^{\mathrm{T}} \tag{2.2} $$

where x'_{C}, y'_{C}, and z'_{C} describe the X, Y, and, Z components of the 3D scene point respectively.

This 3D scene point can then be projected into the image plane resulting in an image projection

$$ \bar{\mathbf{x}} = \begin{bmatrix} \bar{x} & \bar{y} & 1 \end{bmatrix}^{\mathrm{T}}. \tag{2.3} $$

This projection is the intersection of the line from the scene point to the camera center to the plane with $z = 1$, i.e. the image plane. To account for rotation and translation of the cameras regarding a fixed frame, a rotation matrix $\mathbf{R}$, see Section 2.3.2, and translation vector $\mathbf{t}$ are introduced. They transform the scene point $\mathbf{x}'_{\mathrm{C}}$ from the camera frame into a scene point $\mathbf{x}'_{\mathrm{F}}$ in the fixed frame F as

$$ \mathbf{x}'_{\mathrm{F}} = [\mathbf{R} \;\; \mathbf{t}] \begin{bmatrix} \mathbf{x}'_{\mathrm{C}} \\ 1 \end{bmatrix}. \tag{2.4} $$

where $[\mathbf{R} \;\; \mathbf{t}]$ is a 3×4 transformation matrix. Details on coordinate transformations are given in Section 2.3.2. In the scope of this work, the camera coordinate frame C corresponds to the fixed frame F, since we regard each of the cameras of the ego vehicle separately and only transform the detected objects back into the ego vehicle frame. Therefore, we neglect the sub-index F in the following.

Figure 2.2 visualizes the projection of a scene point $\mathbf{x}'$ into the image plane $z = 1$, resulting in the projected point $\bar{\mathbf{x}}$. In the pinhole camera model, the image plane projections are given in the length unit $\mathbb{R}$. The center of the image plane corresponds to the coordinate (0, 0, 1), which is not the case for the actual image of the camera given in the pixel dimension. Typically, the origin is placed in the upper left corner of a camera image. The mapping between the image plane and the camera image is done using a 3×3 matrix $\mathbf{K}$, termed the inner or intrinsic parameters of the camera. The inner parameters include the focal length f, the principal point (x_0, y_0) describing the center of the image, the aspect ratio γ and the skew s such that

$$ \mathbf{K} = \begin{bmatrix} \gamma f & sf & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.5} $$

For more information on the inner parameters, please refer to Chapter 11 of [30].

Together with the transformations described earlier, the camera intrinsics complete the so called camera equation:

$$ \begin{aligned} \lambda \tilde{\mathbf{x}} &= \mathbf{K} [\mathbf{R} \;\; \mathbf{t}] \, \mathbf{x}' \\ &= \tilde{\mathbf{P}} \mathbf{x}' \end{aligned} \tag{2.6} $$

where $\tilde{\mathbf{x}}$ is the projected point in the camera image, λ is the projective depth of the scene point and $\tilde{\mathbf{P}}$ is termed the camera matrix.

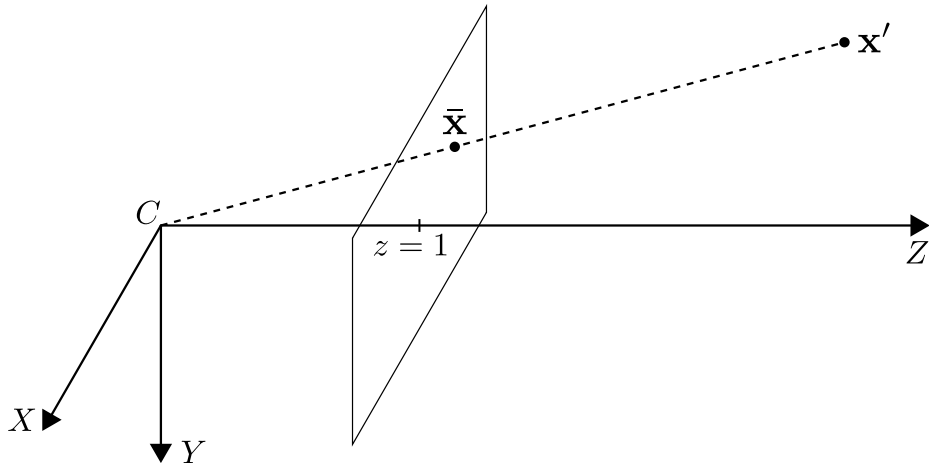


Figure 2.2: Visualization of the camera pinhole model. $\mathbf{x}_\mathrm{s}$ represents a scene point in the 3D surroundings of the camera, $\mathbf{x}_\mathrm{p}$ the corresponding point in the image plane. The origin C represents the camera center, also called pinhole. The cameras coordinate system is depicted through the arrows.

3D to 2D projection

The derived camera equation can be used to project a 3D scene point $\mathbf{x}'$ into the image plane. This is achieved by first calculating

$$\tilde{\mathbf{x}}_i = \tilde{\mathbf{P}}\mathbf{x}' \tag{2.7}$$

where the sub-index i states that the coordinates are in-homogeneous. To receive homogeneous coordinates, $\tilde{\mathbf{x}}_i$ has to be divided by its third coordinate $\tilde{\mathbf{x}}_{i,3}$:

$$\tilde{\mathbf{x}} = \frac{\tilde{\mathbf{x}}_i}{\tilde{\mathbf{x}}_{i,3}} \tag{2.8}$$

The resulting projection $\tilde{\mathbf{x}}$ in homogeneous coordinates is not uniquely defined since all scene points on the line between the camera center and the projected point correspond to this projected point. Additionally, one can determine whether a given 3D scene point is in the field of view (FoV) of the camera. If the third coordinate $\tilde{\mathbf{x}}_{i,3}$ of the in-homogeneous coordinates is negative, the scene point does not lay within the FoV and therefore can't be projected into the image.

2D to 3D projection

Assuming the projective depth λ of the scene point is known, the 3D scene point can be reconstructed from the projected point in the image using the camera equation. For easier notation, the first 3 columns of $\tilde{\mathbf{P}}$ are denoted as $\tilde{\mathbf{P}}_{3\times3}$ and the last column as $\tilde{\mathbf{P}}_4$. The 3D scene point $\mathbf{x}'$ is reconstructed by

$$\mathbf{x}' = \lambda\mathbf{R}^\mathrm{T}\mathbf{K}^{-1}\tilde{\mathbf{x}} - \tilde{\mathbf{P}}_{3\times3}^{-1}\tilde{\mathbf{P}}_4 \tag{2.9}$$

where $\mathbf{R}$ is the rotation matrix and $\mathbf{K}$ the camera intrinsics matrix. For a derivation, please refer to the literature mentioned above.

2.3 Sensor fusion

Traditionally, sensor fusion in object detection for different kinds of sensors is performed on object-level, i.e. each of the sensor modalities has its own detection pipeline. The resulting object detections are then combined, classic fusion methods like Kalman filters have been used to tackle this [32].

While object detection on a single sensor modality has already shown promising results, recent research also focuses on multi-modal object detection using deep machine learning. The results of multi-modal object detection are often not only more precise, but also more robust due to the complementary properties of different sensor modalities [4]. Since this work tackles the fusion of camera and radar sensors, the following summary also focuses on this part of the research field and only gives a brief summary on different modalities where camera-radar fusion could benefit from concepts used with these sensors. The concepts introduced in the following, e.g. in Section 2.3.1 also apply for sensor modalities different from camera and radar.

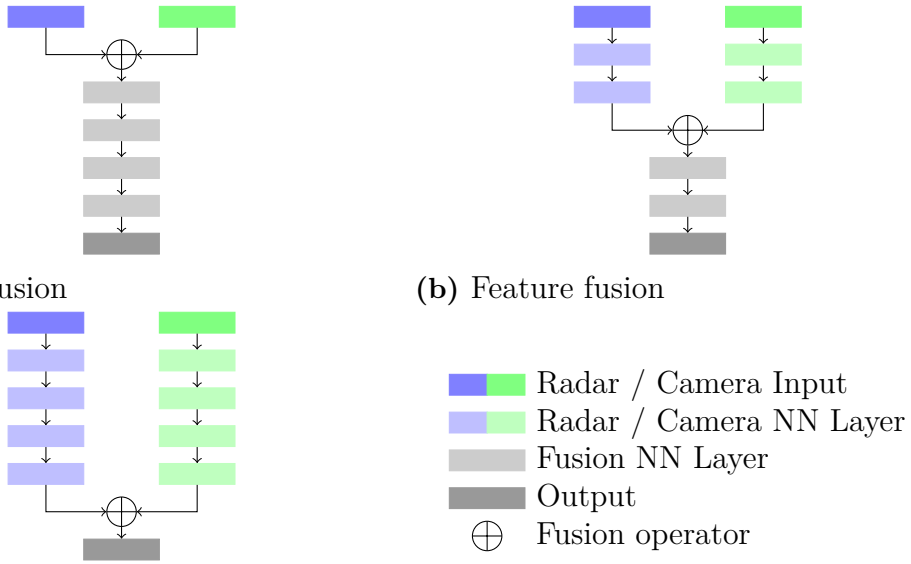
2.3.1 Fusion Depth

In general one can distinguish between three levels of sensor fusion – sensor fusion on *data-level*, on *feature-level*, and on *object-level* [33]. This classification is based on the location of the fusion operation in the fusion network architecture, see Figure 2.3. Fusion on *data-level* is also referred to as early fusion, similarly fusion on *object-level* is referred to as late fusion. This subsection gives an overview over the different fusion depths and discusses the advantages and drawbacks of each of them. However, authors in literature comparing architectures with differing fusion depths do not find evidence for one method being superior to others in general. The performance depends on a great deal on sensors, the dataset and the rest of the network architecture itself [4].

Early fusion

Early fusion outlines the fusion of sensor data at a stage in the pipeline where the data has not been processed by a neural network yet. The data of the different sensor modalities can be pre-processed based on their physical properties, e.g. to handle noise or different sensor coordinate frames. Some papers, e.g. [34], propose the calculation of regions of interest (ROIs) using the radar data before applying a neural network to the corresponding parts in the camera image. Merging the raw data instead of extracted features however enables the network to fully exploit its information [28]. In addition, the reduction of the network size due to the joint processing of multiple sensor modalities results in lower computational requirements as well as a low memory budget [4].

These advantages however also come with some downsides: the merging of raw data introduces inflexibility to the model, especially when it comes to integrating new sensors into the setup. New modalities or even updated sensor technology of the same modality might result in the necessity to retrain the whole network from scratch due to adapted input channels. Moreover, misalignment between sensors in



(a) Early fusion

(b) Feature fusion

(c) Late fusion

Figure 2.3: Overview over the three levels of fusion depth used in sensor fusion using deep machine learning. The network architecture is not further specified since it varies between different approaches.

the form of calibration errors, sampling rates or sensor defects has a greater negative influence on the results in early fusion [4]. The authors of [35] combine minimally processed radar data with camera images in an early fusion approach for ADAS, the authors of [36] propose a low level fusion scheme for camera, radar and LiDAR sensors.

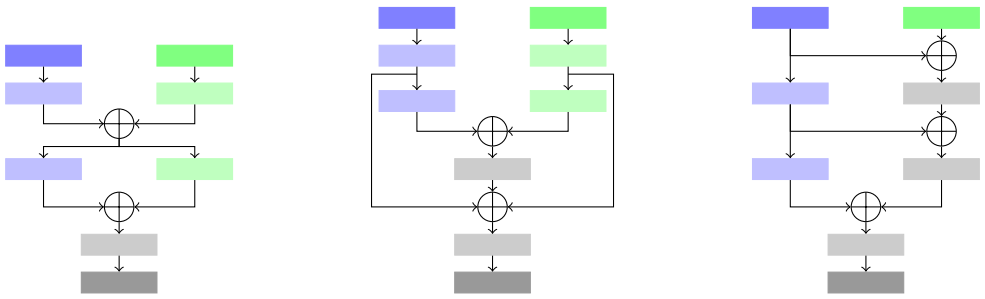
Late fusion

Late fusion combines the detected objects of each individual sensor modality at the end of their pipelines. In a pure late fusion approach, there is no neural network layer after the fusion operator is executed. Late fusion is done on object-level, i.e. in the scope of object detection, each individual sensor unit estimates objects in its surroundings before they are jointly associated and combined. This enables standardizing interfaces between different fusion algorithms which in turn requires no in-depth understanding of the signal processing involved [28] and makes it easier to introduce new sensors since only parts of the network have to be retrained. Also, existing architectures for individual sensors can easily be evaluated.
On the downside, late fusion can't take potentially information-rich, intermediate features of individual sensors into account [4]. Further, it suffers from high computational cost and memory requirements [4]. Regarding Camera-Radar-Fusion, [37, 38] propose two fusion architectures using a late fusion approach.

Feature fusion

Feature fusion, also termed middle fusion, combines the information at a stage in between early and late fusion using the feature representations that result from

applying NNs to the occurring modalities. This introduces the possibility to learn across different sensors' information-rich features at several depths in the pipeline. Feature fusion does not constrain itself to the fusion on one particular layer in the network but can also apply a fusion operator multiple times. Some variants of this are displayed in Figure 2.4. Feature fusion can also contain elements of early and



(a) Deep feature fusion **(b)** Short-cut feature fusion **(c)** Early feature fusion

Figure 2.4: Some variants of Feature level Fusion. For a legend, please see Figure 2.3. Figure 2.4a combines the information of the two modalities at different stages of the fusion pipeline. Figure 2.4b introduces the extracted features of an early stage at a later point in the pipeline again. Last, Figure 2.4c combines early fusion with feature fusion.

late fusion in the architecture, see Figure 2.4c. This may lead to less rich feature extraction from at least one modality but can introduce some of the advantages of early and late fusion. Feature fusion is highly flexible with the biggest potential for precision whilst the complexity makes it difficult to find the optimal way to fuse intermediate feature layers together. Most networks can not be classified as a definite member of either early fusion or late fusion due to some intermediate layers and are therefore rated among feature fusion. The authors of [39] introduce a feature fusion approach as a single shot detection network, [40] introduces *CenterFusion*, which is explained in detail in Chapter 4. CRF-Net [36] combines an early fusion approach with fusion layers reaching deeper into the network, similar to the concept visualized in Figure 2.4c.

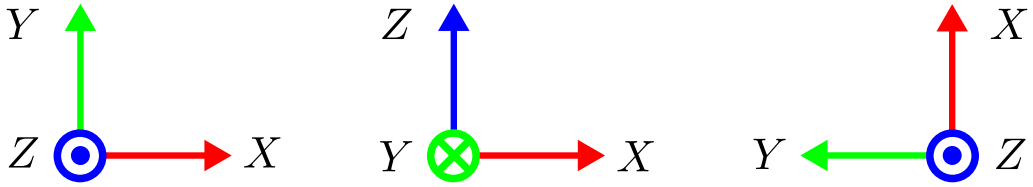
2.3.2 Data Association

One key element of a fusion algorithm is the association of data from different sensors. In the scope of this work, this involves the incorporation of a two dimensional camera image and the corresponding points in the point cloud of the radar, expressed in coordinates of the 3D space. Data association is especially important in early fusion since the raw sensor data has to be combined such that the associated data corresponds to the same object in the ego vehicles surroundings. However, this issue also matters in feature fusion approaches since the features extracted from the data have to be combined as well. After a brief description of essential coordinate transformations, the following Section describes and analyzes different methods to perform this association.

Coordinate systems

To simplify the understanding of the data association and further chapters, the coordinate systems used in this work are introduced in the following. They are heavily based on the coordinate systems introduced in the *nuScenes* dataset, see Section 3.1. All of the coordinate systems used follow the right-hand rule, hwever their orientation differs.

The coordinate system of the ego vehicle is a right hand system whose Y-axis points towards the forward driving direction of the car. In contrast, the camera is represented with the Z-axis facing in the main viewing direction while the radar data is given in a coordinate frame with the X-axis facing forward. The three differing coordinate systems are displayed in Figure 2.5, each facing in the same direction. When rotating a camera or a radar, e.g. when dealing with sensors facing backwards, their coordinate system rotates w.r.t. the ego-vehicle's frame.



(a) Ego vehicle coordinate system **(b)** Camera coordinate system **(c)** Radar coordinate system

Figure 2.5: Orientation of the ego-vehicle, camera, and radar coordinate systems. All coordinate systems are facing forward in this figure. A point encapsulated by a circle represents an arrow pointing towards the reader, a cross encapsulated by a circle an arrow in the opposite direction.

Most of the computations in this work are performed in the camera coordinate system since the outputs of the developed fusion architecture are expressed in this frame. These object detections then can easily be transformed to the ego-vehicle frame or even a global frame assuming the pose of the car is known.

Coordinate transformation point cloud

To associate data from different sensor modalities, one has to ensure they are represented in the same coordinate system. Consequently, the point cloud first has to be transformed into the sensor coordinate frame of the camera to make sure there is no spacial misalignment between the data. The matrix

$$\hat{\mathbf{P}}_{\mathrm{R}} = \left[\hat{\mathbf{p}}_{1,\mathrm{R}}, \hat{\mathbf{p}}_{2,\mathrm{R}}, \ldots, \hat{\mathbf{p}}_{N,\mathrm{R}}\right] \in \mathbb{R}^{4\times N} \tag{2.10}$$

represents the homogeneous coordinates of $N \in \mathbb{N}$ points, where R expresses the coordinate frame of the radar and the i-th column of $\hat{\mathbf{p}}_{1,\mathrm{R}}$ denotes the coordinate vector

$$\hat{\mathbf{p}}_{i,\mathrm{R}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \in \mathbb{R}^4 \tag{2.11}$$

of the i-th point. The transformation is expressed as

$$\hat{\mathbf{P}}_{\mathrm{C}} = \mathbf{T}_{\mathrm{CR}}\hat{\mathbf{P}}_{\mathrm{R}} \tag{2.12}$$

where C denotes the coordinate frame of the camera and

$$\mathbf{T}_{\mathrm{CR}} = \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}_{1\times 3} & 1 \end{array}\right] \in SE(3) \tag{2.13}$$

is a matrix describing the transformation between the coordinate frames of radar and camera consisting of a rotation $\mathbf{R} \in SO(3)$ and a translation $\mathbf{t} = \left[t_X, t_Y, t_Z\right]^{\mathrm{T}} \in \mathbb{R}^3$. $\mathbf{T}_{\mathrm{CR}}$ is a 4×4 matrix out of the special euclidean group $SE\,(3)$ while $\mathbf{R}$ is a matrix of size 3×3 in the special orthogonal group $SO\,(3)$, see [41], Chapter 3. The radial velocities in the point cloud of the radar do not have to be translated but only rotated into the correct coordinate system using the rotation matrix $\mathbf{R}$ in Equation (2.13).

In general, a point in the radar point cloud consists of additional information like the RCS value ρ, see Section 2.2.1, which does not have to be transformed in the same manner. In addition, most radar sensors only output a planar point cloud, such that one dimension in Equation (2.11) consists of zeros only.

Projection of point cloud to 2D

Subsequently to the transformations described above, the data from the point cloud and the image are expressed in the same coordinate system which allows a 3D to 2D projection as introduced in Section 2.2.2.

Since the radar point cloud lacks information in height its points will not get projected to the exact position in the image plane of the reflection when only projecting a single point. Therefore, [36] assumes a fixed object-height assigned to each radar point. A single radar point is projected onto a 2D-line in the image plane starting on the ground of the bird's-eye view (BEV) plane.

Figure 2.6 illustrates this concept using two separate radar points in the 3D-plane. One of the radar detections is closer to the camera frame than the other. Although both have the same artificial height in the 3D-plane, they appear to be of different size in the image plane. This difference is intentional and represents the height of a typical object in the environment of autonomous driving regarding its distance to the sensors.

To complete the association of camera and radar data, the projection is performed for all of the radars point cloud features, specifically RCS and radial velocity. The resulting features-images have the same resolution as the original input images and contain the value of a point cloud feature at the projected 2D-lines and zero elsewhere. These radar feature images are then concatenated with the 3 existing RGB-channels of the original image resulting in the association of radar and camera. Figure 2.7 shows an example for this method, where Figure 2.7a is the original image from the camera and Figure 2.7b shows the projection of one radar channel into the 2D image along with the original RGB image. Notice that the 2D-lines corresponding to radar points are smaller the further away the radar points are.

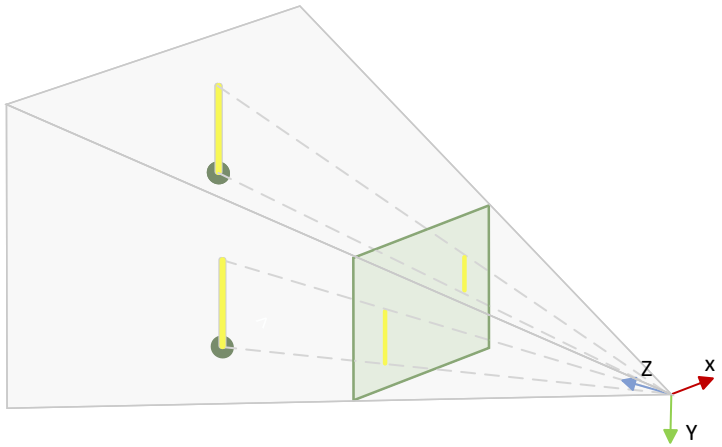


Figure 2.6: Projection of planar radar points into the image plane. The radar points (green dots) on the ground of the 3D plane are associated with a fixed height and projected into the green image plane. The coordinate system represents the camera's origin.

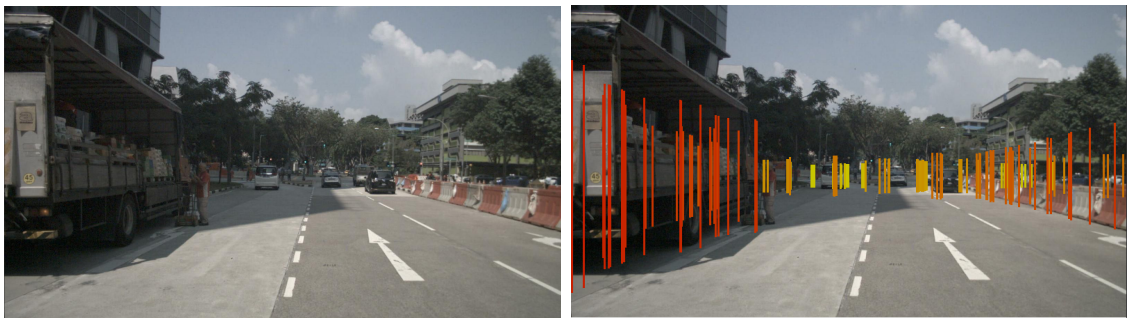
Frustum based association

The authors of [40] follow a different approach to associate the data of the two sensor modalities. Objects are initially predicted using an image-based detection network architecture. The outputs of the network are the 2D- and 3D-bounding boxes (BBs). The 2D-BB is predicted in the image plane while the 3D-BB also contains the depth and orientation information of the object. In contrast to the method described earlier, the association between camera and radar is done in the camera's XZ-plane.

To make up for the missing height information of the radar point cloud explained earlier, the authors of [40] extend the points not only into lines but into pillars. Figure 2.8 illustrates the pillar expansion. The pillars have a square cross section of $20\,\text{cm} \times 20\,\text{cm}$ with a height of $1.5\,\text{m}$ [40]. The radial velocity and RCS measurements are assigned equally to the whole pillar. The points are extended to pillars and not just lines because of the uncertainty in the measurement of radar points. Their position can differ from their corresponding objects, especially since they are aggregated over time. The expansion to pillars makes the association more robust against such inaccuracies [42].

Using the corners of the 3D-BB, a region of interest encapsulating its whole surface is created. This ROI is depicted in BEV in the cameras XZ-plane as the dark blue polygon in Figure 2.9. In the following, the ROI is termed frustum in consequence of its geometrical shape in a three dimensional space. In general, the geometric object of a frustum is by definition the basal part of a cone formed by cutting off the top by a plane parallel to the base.

The frustum is created by casting the camera projection lines into the 3D space such that the 3D-BB is tangent to the projection lines. The planes limiting the



(a) Original camera image in RGB from *nuScenes* dataset [1].

(b) Radar point cloud projected into the 2D image plane (RGB+radar features).

Figure 2.7: Visualization of the projection of radar information into a 2D image. For Figure 2.7b red lines represent close radar points, lighter colors display points with increasing distance.

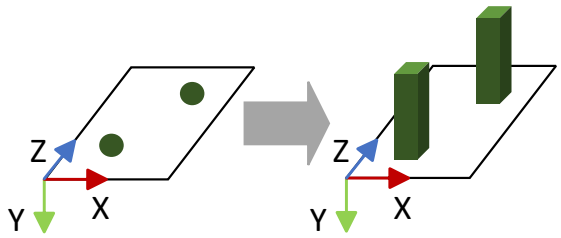


Figure 2.8: Radar points are expanded into pillars due to the lack of height information in the radar measurement.

frustum along the projection lines are defined by the minimal and maximal depth of the BB. Both the base and the top surface of the frustum are parallel to the image plane.

The pillars are considered to be associated to the frustum if any part of the pillar is inside the area of the frustum. The radar point corresponding to the pillar is therefore associated to the same preliminary object detection the frustum is constructed from. Finally, the authors of [40] propose to select the closest pillar of all associated pillars to be used for the fusion step. This is determined by the depth only, not by the euclidean distance. The association is therefore done object-wise, in contrast to the raw-data association through the projection of the point cloud to the image plane described earlier. Figure 2.9 visualizes the association.

Checking if the pillars are inside the frustum is computationally expensive. This motivates why the creation of the frustum is only for demonstration and is not used in the implementation. To determine whether a pillar can be associated to an object or not, the pillars can also be projected into the image using the camera projection matrix as explained in Section 2.2.2. In the projected plane they are checked for intersection with the 2D-BB that is predicted by the detection network. The projected pillars are approximated by axis-aligned BBs to easily check for intersection with the axis-aligned 2D-BB of the object. Figure 2.10 illustrates the projection of the pillars. Brighter color corresponds to greater depth regarding the camera's coordinate system. The object's 2D-BB is shown in red in Figure 2.10. If multiple

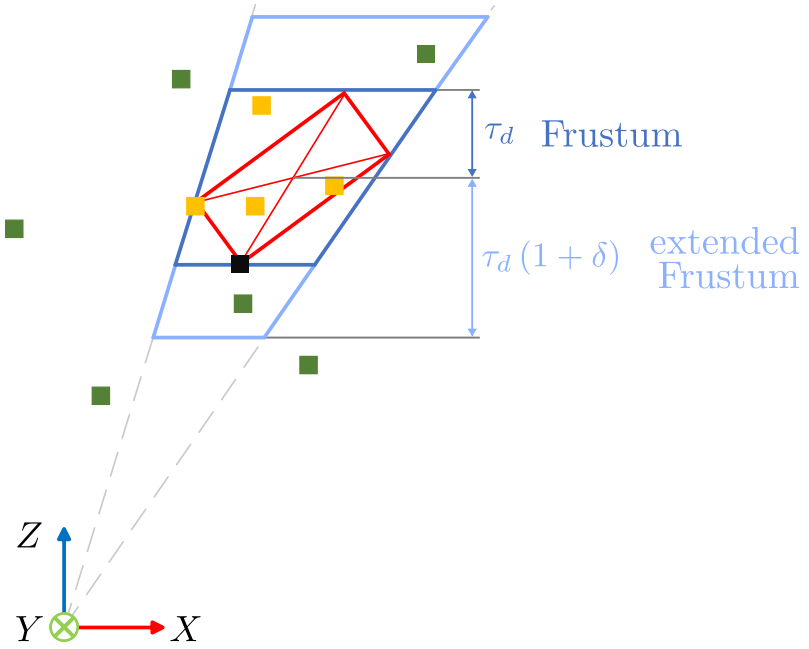


Figure 2.9: BEV of a 3D-BB (red rectangle) from mono-vision. The dark blue polygon displays the frustum which is defined depth-wise by the threshold τ_d, the light blue polygon is the frustum extended by a factor δ. The radar pillars within the cameras FoV are depicted as squares with different color. The green square represent the radar pillars not associated to the object, the orange squares overlap with the frustum and are therefore associated to the object. Associated means here, the pillars and the frustum share the object they are assumed to belong to. The black square is the closest pillar – the one with the smallest depth – and is therefore the only pillar used for the fusion itself. The displayed coordinate system is from the camera.

pillar's BBs overlap in the image the closest one, i.e. the one with the smallest depth, dominates. If any part of each pillar's BB is intersecting with the object's BB this pillar is stored but not yet associated.

So far, the pillars have been filtered by their angle to the camera. If the filtered pillars additionally match with the object depth-wise, they can be associated to the object. The depth of the pillars is the Z-coordinate and is extracted from the measurement of distance and angle. In contrast, the depth estimation to the object has to be learned by the image-based object detector. All radar pillars within a threshold τ_d are associated to the object. This threshold represents the depth of the frustum. The threshold τ_d can be constructed with the predictions of the network by using the 3D-BB. The minimum and maximum values of the BB corners $c_{\text{3D-BB}}$ in the depth or Z-dimension are taken to calculate the threshold

$$\tau_d = \frac{\max\left(c_{\text{3D-BB}_Z}\right) - \min\left(c_{\text{3D-BB}_Z}\right)}{2}. \tag{2.14}$$

The depth boundary for association starts from the center of the 3D-BB reaching towards the positive and negative Z-direction in magnitude of the threshold τ_d. If there are multiple candidate pillars the one with the smallest depth, i.e. the one

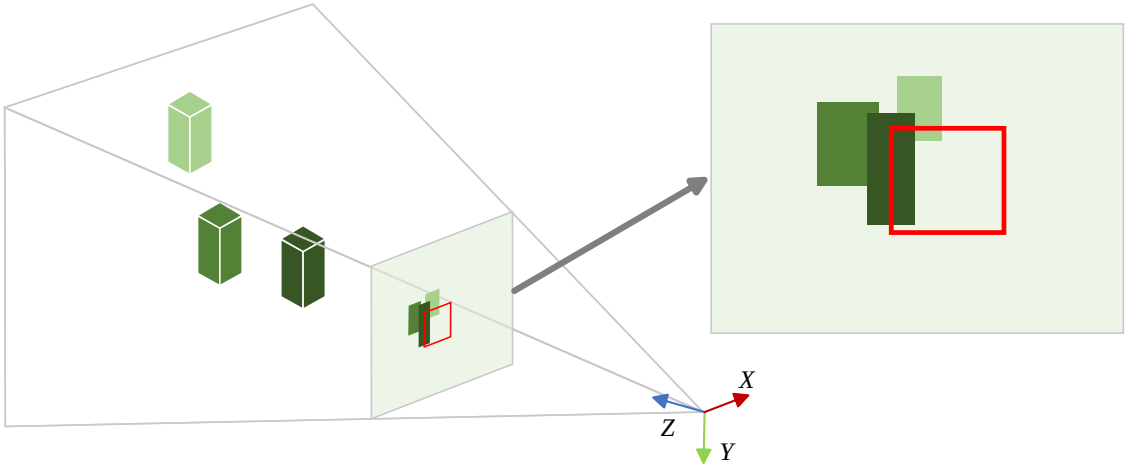


Figure 2.10: Projection of radar pillars into the image. The projection would be a distorted rectangle which is enclosed by an axis-aligned BB. The BBs of the pillars are shown in the image. The darker the pillar is the closer it is to the coordinate system origin. The red rectangle is the 2D-BB predicted by the network. The image is displayed again on the left side to make the overlapping of the pillar BBs better visable.

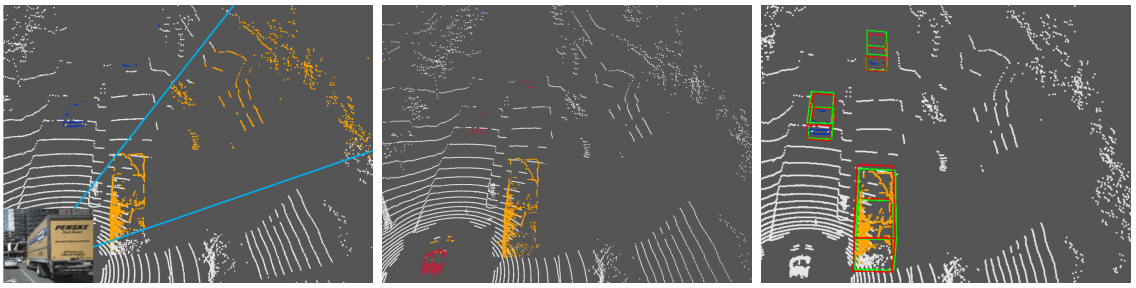
with the smallest Z-coordinate, is used for the fusion step.

Due to uncertainty in the depth estimation by a monocular camera, the prediction of the 3D-BBs from the network is inaccurate. Therefore, a relaxation factor δ is introduced to enlarge the frustum and generally associate more points to the object. Figure 2.9 displays this parameter δ which increases the depth threshold τ_d proportionally. It controls the size of the frustum to robustly associate correct radar pillars to the object even if the estimated depth of the 3D-BB is imprecise. However, this parameter can also cause false positive associations when an object is occluded by another. Thus, it has to be chosen carefully.

Out of the radar pillars in the frustum, in *CenterFusion* the one with the smallest depth is the only pillar to be further processed. This is a simple approach on extracting information from the associated pillars. We propose to extract more information by using a neural network that receives the associated points as input, which is further explained in Section 5.1. The process of fusing the multi-modal information is explained in detail in Chapter 4.

Point-Painting association

The authors of [43] introduce the approach of fusing a camera image with a point cloud involving semantic segmentation and 3D object detection called *FusionPainting*. The framework consists of three main parts: first, the 2D image and 3D point cloud are processed using semantic segmentation networks. The 3D semantic segmentation classifies each point in the point cloud as part of an object in a class occurring in the cars environment, e.g. a car, truck or pedestrian, directly. This results in the segmentated point cloud shown in Figure 2.11b, red points represent misclassified areas.



(a) 2D point cloud segmentation. **(b)** 3D point cloud segmentation. **(c)** *FusionPainting* results.

Figure 2.11: Overview over segmentation and results in *FusionPainting* [43]. 2.11a shows the point cloud segmentated using the 2D segmentation method. The orange points are segmentated as part of the truck, the blue points correspond to other objects. 2.11b displays the point cloud segmentated using the 3D segmentation method. Misclassified points are colored red. Finally, 2.11c shows the results of *FusionPainting* [43]. 3D-BBs are colored in green, ground truth BBs in red. Images are taken from [43] © 2021 IEEE, Figure 1.

To improve the accuracy and precision in the classification of the point cloud, the image corresponding to the point cloud is segmentated as well. The results are projected into the 3D space using the camera's intrinsic properties, see Section 2.2.2, which results in a frustum-like shape in 3D-space. All points in this volume are classified using the result of the 2D camera segmentation, the projection is displayed in Figure 2.11a.

The segmentated point clouds of both sensor modalities come with their own pros and cons. The 2D segmentation generally achieves better performance due to the detailed textural structure but suffers from the *boundary blurring effect*, due to which points in a frustum behind the objects are segmentated as well. This effect is a result of the 2D-3D projection. The 3D segmentation in contrast performs much better on the boundary of obstacles but gives worse results in category classification due to sparsity in the point cloud.

The segmentated point clouds of both, 3D and 2D segmentation are then the input of a neural network termed *Adaptive Attention Module*. This module is designed to suppress the disadvantages of both segmentations and combine their advantages. The architecture is based on an attention mask masking voxels in the point cloud. A detailed explanation is available in [43].

As the last step, the fused and segmented point cloud serves as the input to a state-of-the-art 3D detector which outputs the pose and category of detections. Figure 2.11 shows the detections and the annotations for one example.

Although this approach results in promising scores for the fusion of camera and LiDAR point clouds, it is questionable whether it is adequate when using radar point clouds instead due to their sparseness. With the dataset used in this work, see Section 3.1, the segmentation and especially detection of objects on the point cloud only would suffer from the lack of data. However, assuming improvement in radar sensor technology and available datasets, this approach could become of higher interest in the future.

3
Data

The data used in a machine learning approach plays a crucial role in the performance of the network. This chapter gives an overview of the data that has been chosen for this work as well as the preprocessing and augmentation it is handled with.

3.1 Dataset

With the increasing popularity of machine learning algorithms in the automotive sector, more public datasets are released and contributed to in recent years. However, most of the datasets available currently focus on a sensor-setup involving camera and LiDAR and do not include radar data. Examples are the popular datasets from the University of Karlsruhe KITTI [44] and the Waymo dataset [45]. Since this work depends on radar data, these popular datasets are not included in the following comparison. Table 3.1 gives an overview of the public datasets which contain both, camera and, radar data and compares them by several categories.

Table 3.1: Comparison of publicly available datasets including radar and camera data. # is an abbreviation for *number* in this context, *Res* for *Resolution*.

Dataset	Size	Radar			Cameras	Sensors	Annotations		
		#	Doppler	Res			Type	# annotated images	# Categories
nuScenes [1]	++	5	✓	0	6	IMU, GPS, LiDAR	3D	240 000	23
Oxford Radar RobotCar [46]	+	1	✗	+	4	IMU, GPS, LiDAR	–	–	–
CARRADA [29]	-	1	✓	+	1	–	Spectral	7 193	3
CRUW [47]	+	2	✗	+	2	–	Spectral	76 000	3
Zendar [48]	+	1	✓	+	1	IMU, GPS, LiDAR	2D	11 000 objects	1
RADIATE [49]	+	1	✗	+	1	LiDAR	2D	44 000	8

Several requirements are imposed on a dataset for deep learning. First of all, there has to be a big amount of data to be able to train a robust network. Out of the six datasets in Table 3.1, *nuScenes* [1] is the biggest dataset by far, containing about 1 400 000 images and radar sweeps. CRUW [47], the Oxford Radar RobotCar dataset [46], Zendar [48] and RADIATE [49] still contain a reasonable amount of data in varying scenarios while CARRADA [29] is too small for the purposes of this work. Additionally, CARRADA is recorded on a test track and therefore lacks varying, realistic scenarios including different weather conditions. In this regard,

the other datasets are more diverse although *nuScenes* and the Oxford dataset in contrast to RADIATE do not contain any data recorded in fog or snow [49].

Besides the pure size of the dataset, the amount and quality of available annotations is vitally important. The Oxford dataset does not come with any annotations, Zendar only contains 11 000 annotated cars in the whole set. Both datasets are therefore not considered further. To make use of the complementary properties of radar and camera, both sensors have to be of sufficient quality. The radars in *nuScenes* are sensors used in the automotive industry that output a pre-processed point cloud directly. This point cloud is sparser compared to the data provided by other datasets, but since five sensors are placed around the car, this disadvantage is partly compensated for. Neither CRUW nor RADIATE contain velocity measurements based on the Doppler effect that are crucial for the Camera-Radar sensor fusion approach of this work and therefore also have to be crossed off for our work.

To sum up, the *nuScenes* dataset matches the requirements on a dataset in the best manner and is therefore selected for this work.

nuScenes

The *nuScenes* dataset [1] is currently one of the most popular datasets in literature regarding the fusion of camera and radar data, see e.g. [36, 39, 40]. In total, it provides $n_{\text{scenes}} = 1000$ scenes from Boston and Singapore, where every scene is of the length of $\Delta t = 20$ seconds. Within the dataset, 23 different object classes are annotated with 3D-BBs at a rate of $f_{\text{ann}} = 2\text{Hz}$. The rest of the data is also available but not annotated. In total, the dataset therefore contains

$$\begin{aligned} n_{\text{keyframes}} &= n_{\text{scenes}} \cdot \Delta t \cdot f_{\text{ann}} \\ &= 1000 \cdot 20 \cdot 2 = 40\,000 \end{aligned} \tag{3.1}$$

keyframes. One keyframe contains the data of all the sensors, i.e. six camera images, for one timestamp. Out of the 1000 scenes available to public, only 850 are intended to be used for training and validation. For the remaining 150 scenes, only the data, excluding the annotations, is available to public. These scenes are used in the *nuScenes Object detection task* [50], see below.

Sensors

Figure 3.1 shows the position and coordinate system used in the *nuScenes* dataset. Both, the five radars and six cameras are from the same type each, however the lenses differ between the camera facing backwards and the five other cameras. The camera facing backwards has a FoV of 110° while the other cameras work with a FoV of 70°. The technical details of the said sensors and the inertial measurement unit (IMU) are given in Table 3.2.

Sensor calibration

To obtain precise object detections within the ego vehicle's surroundings, the calibration of both, intrinsic and extrinsic sensor parameters is essential. Extrinsic

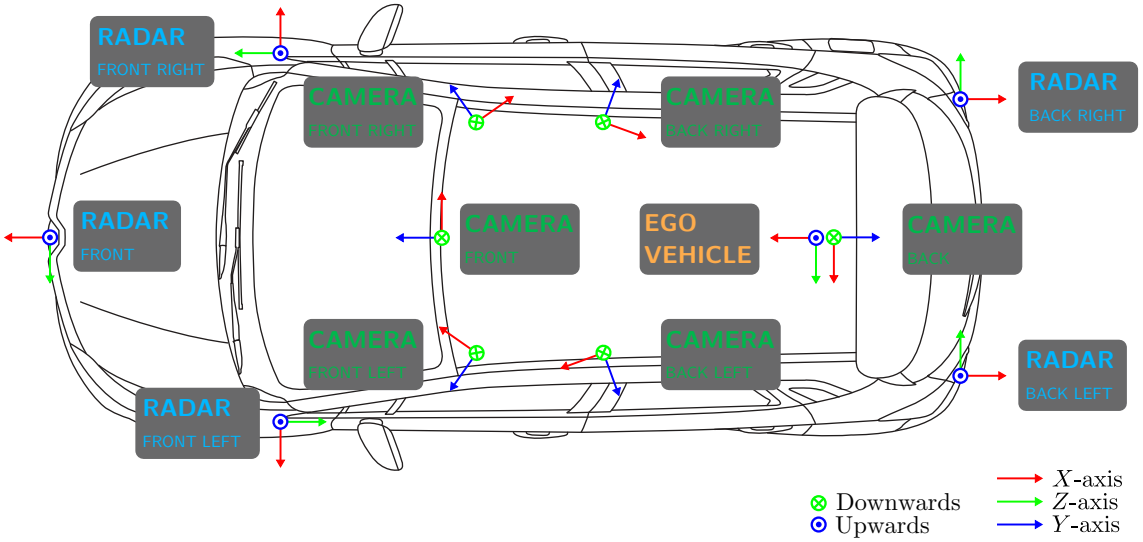


Figure 3.1: Sensors used and their corresponding coordinate systems in the *nuScenes* dataset[1]. The ego vehicle coordinate systems X-axis points towards the forward driving direction and is located at the midpoint of the rear axle projected onto the ground. The top-down view of the car is from [51].

parameters describe the transformation, rotation and translation, between a sensor and the reference, in this case the midpoint of the rear axle of the ego vehicle. Intrinsic calibration is important for the camera and describes the calibration of the camera matrix K, see Section 2.2.2. The calibration is described in the supplementary material of [1] and summarized in the following.

The radar is mounted horizontally on the vehicle to obtain the translation of the extrinsic calibration. After collecting data from driving on public roads, the yaw angle respective to the ego coordinate system is computed using a brute force approach to minimize the compensated range rates for static objects in the environment of the car to obtain the rotation matrix.

The camera extrinsics are computed using a cube-shaped calibration target in front of the camera using the LiDAR reference measurements. The camera intrinsics are obtained using a calibration target board. Further details are described in Chapter 7 of [31]. All of the calibration is done by the authors of *nuScenes*, the calibration matrices are available for each measurement.

Sensor synchronization

Besides accurate sensor calibration in spacial dimensions, the synchronization in time is crucial for good cross-modality data alignment. In *nuScenes*, the synchronization between LiDAR and cameras is performed by triggering the exposure of a camera when the LiDAR sweeps across the center of the camera's FoV. However, since the dataset does not use a sweeping radar, this is not possible for synchronization between radar and camera. Instead, all sensor measurements within one keyframe contain individual timestamps. Therefore it is possible to compensate for some errors induced by ego-vehicle motion in the radar point cloud, see Section 3.2.

Table 3.2: Technical details of the sensors used in the *nuScenes* dataset. Source: [52, 53, 54].

	Camera	**Radar**	**IMU / GPS**
Capture frequency [Hz]	12	13	1000
Model	Basler acA1600-60gc	Continental ARS 408-21	Advanced Navigation Spatial
Sensor details	Resolution: 1600 × 1200 cropped to 1600 × 900 ROI Backwards facing: 110°FoV, others: 70°FoV	77GHz FMCW radar Measures distance and velocity independently in one cycle Up to 250m distance Vel. accuracy of $\pm 0.1 \mathrm{km\,h^{-1}}$ 120° FoV near range, 18° FoV far range	Position accuracy of 20mm Heading accuracy of 0.2° with GNSS Roll and pitch accuracy of 0.1°

Data diversity and splits

Diversity of data is an important factor for a generalized and robust NN. In the context of autonomous driving, this imposes the requirement of a dataset consisting of measurements and annotations in different weather conditions, e.g. sun, rain, snow, night, fog, diverse driving scenarios and locations. Although *nuScenes* does not consist of data for all the mentioned weather conditions, the 1 000 scenes open to public are selected to cover rare classes, e.g. animals, ambulances or police vehicles, difficult driving maneuvers and multiple environmental conditions.

The dataset is divided into three splits: *training*, *validation* and *test set*. This is common practice in machine learning to avoid testing on the same data that is used for training. Table 3.3 shows the division for the *nuScenes* dataset. Out of the 1 000 scenes, 70% are used for training, while 15% of the data is used for each – validation and testing. Considering the number of annotated images each split contains, the division is reasonable and follows good practices. The validation set consists of 36 000 images in total, i.e. 6 000 images for each of the six cameras. The division was performed by the *nuScenes* team by hand, therefore the percentage of scenes in difficult environmental conditions such as night or rain are close for the training and validation set, see Table 3.3.

Table 3.3: Division of the dataset into training, validation and test set. Since the test set is used in the *nuScenes* detection task, environmental information is not available to public.

	Training	Validation	Test
Number of scenes	700	150	150
Number of images	168 000	36 000	36 000
Night scenes	12%	10%	–
Rain scenes	19.7%	18%	–

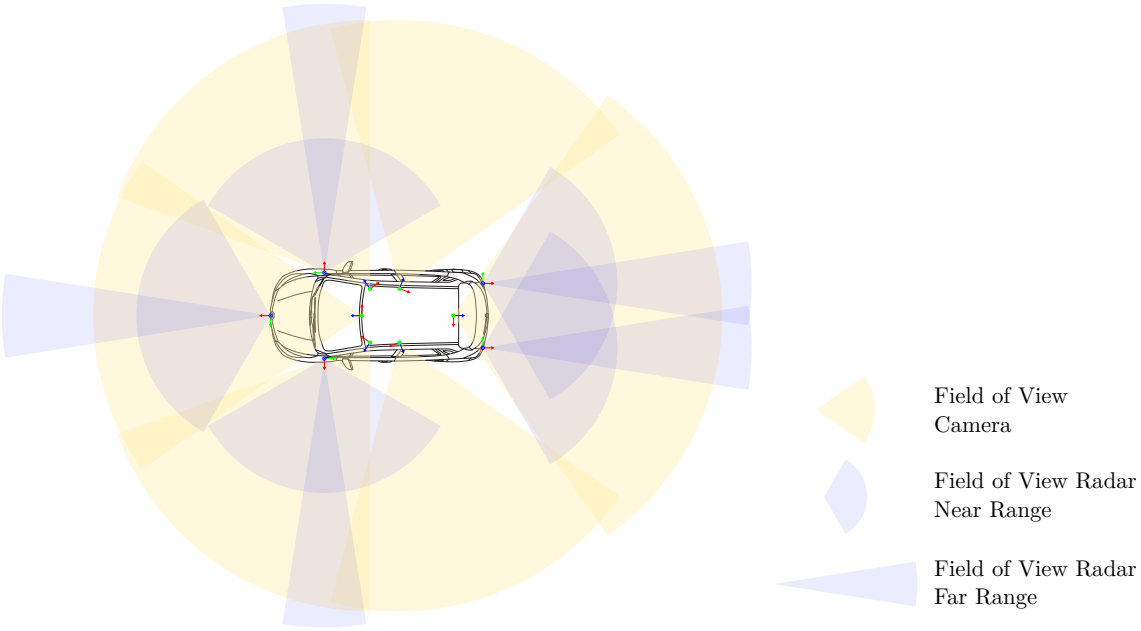


Figure 3.2: Sensors and their corresponding FoVs in the *nuScenes* dataset. The coordinate systems correspond to the ones introduced in Figure 3.1. The FoV of the radar sensors changes between measurements in near and far range. The sizes of the cones do not represent the range at which the sensors operate but are chosen such that the Figure illustrates the overlapping of different sensor modalities. The source for the top-down view of the car is [51].

Data annotations

The objects in *nuScenes* are annotated in the 3D space. Each annotation consists of a three dimensional BB, a global pose, a semantic category, e.g. car, pedestrian, and an attribute, e.g. moving, parked, see Table A.2 for more details. To obtain a 2D-BB in the image, the 3D-BB of the annotation can be projected into the image plane using the camera-specific camera matrix, see Section 2.2.2.

In total, *nuScenes* provides 23 different categories, out of which the ten most relevant are extracted, see Table A.1. Figure 3.3 shows the frequency of annotated objects among the ten categories for both, training and validation. Note that validation and training are scaled on a different Y-axis. One can conclude that the distribution between validation and training set is consistent even when it comes to annotated categories.

In order to develop and analyze the results of a deep learning network, it is crucial to understand the data used for training. Figure 3.4 shows the average number of radar points that overlap with 3D-BBs of objects within the ten categories introduced, analyzed over both training and validation split.

Large objects like buses, trailers or trucks have a larger surface and therefore reflect more radar points compared to smaller objects as pedestrians or traffic cones. The annotations for the categories motorcycle, bicycle, barrier, pedestrian and traffic cone consist of less than one radar point in average. Due to the lack of data, it is harder for the sensor fusion to profit from the radar input in these categories which

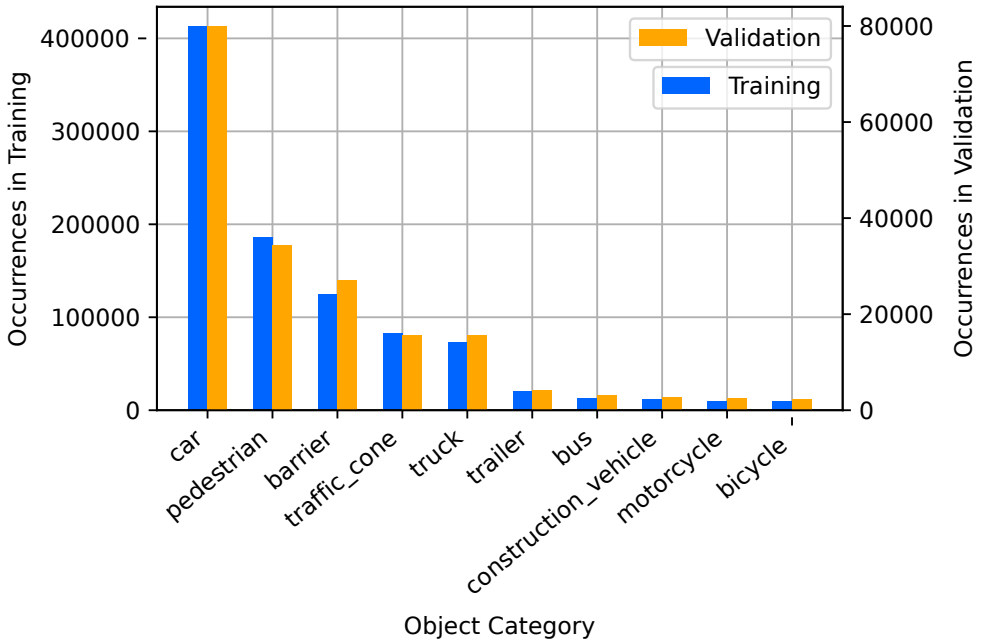


Figure 3.3: Occurrences of object categories in the annotations of both, validation and training set. The validation set is colored in orange, the training set in blue. Note that the Y-axis is different for the two subsets.

has to be kept in mind when analyzing the results.

nuScenes object detection task

The authors of the *nuScenes* dataset provide a detection task on their website [50]. The detection task makes use of the 150 scenes in the test-split and evaluates the network using the metrics mAP and NDS, see Section 3.4. The detection task also includes a leader-board for various sensor setups allowing an easy comparison with state-of-the-art fusion algorithms in literature.

3.2 Data preprocessing

Before inputting the data of both, camera and radar, into the training and validation process, some preprocessing has to be performed on it. This sub-chapter explains these steps along with the format of the ground truth annotations used in this work.

Since *nuScenes* nominates all the data of one time stamp (six camera images, five radar point clouds) as a *sample*, in the following we will refer to the smallest data unit as a mini-sample. Each mini-sample used for the training consists of one camera image, the radar points in the FoV of this camera and the corresponding annotations for the objects captured in the image. Additional information such as the camera calibration matrix is stored in each mini-sample as well. In the following, a *radar sweep* is defined as the point cloud corresponding to a radar measurement at a single timestamp t_R.

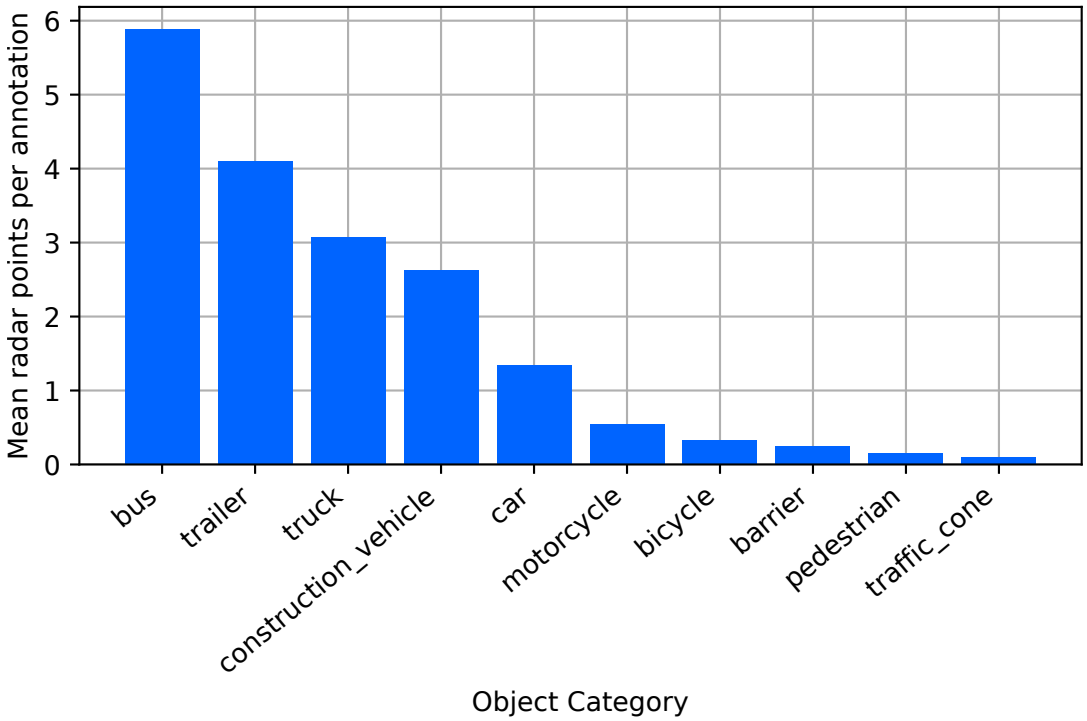


Figure 3.4: Mean number of radar points within object annotations in the *nuScenes* dataset for each category. All annotations further than 60 meters away from the center of the car are disregarded.

Camera preprocessing

To reduce the computational time and the amount of weights in the neural network, the image provided by the dataset is down-scaled from its original resolution of 1600×900 to an input resolution of 800×448 where the first value corresponds to the width and the second to the height of the image.

Since the network architecture of *CenterNet*, see Section 4.1 and specifically the backbone deep layer aggregation (DLA)-34, requires the image to be completely divisible by 32 the input image height of 448 is used, although this results in a slight distortion of the image. The effect however is minimal and therefore can be disregarded.

Radar preprocessing

A single point $\bar{\mathbf{p}}_{i,\mathrm{R}}$ out of the radar point cloud with the index i in the radar's coordinate frame R can be expressed as

$$\begin{aligned} \bar{\mathbf{p}}_{i,\mathrm{R}} &= \begin{bmatrix} \tilde{\mathbf{p}}_{i,\mathrm{R}}^{\mathrm{T}} & \bar{v}_X & \bar{v}_Y & \rho & \Delta t \end{bmatrix}^{\mathrm{T}} \\ &= \begin{bmatrix} \bar{x} & \bar{y} & \bar{z} & \bar{v}_X & \bar{v}_Y & \rho & \Delta t \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^7, \end{aligned} \tag{3.2}$$

where $\tilde{\mathbf{p}}_{i,\mathrm{R}} = \begin{bmatrix} \bar{x} & \bar{y} & \bar{z} \end{bmatrix}^{\mathrm{T}}$ describes the euclidean coordinates of the radar point in the radar frame, $\bar{v}_X$ and $\bar{v}_Y$ are the projection of the radial velocity in their X- and Y-components respectively and $\Delta t = t_{\mathrm{C}} - t_{\mathrm{R}}$ is the difference in time between the

capturing of the image at time t_C and the radar sweep at time t_R. As mentioned earlier, the trigger for the cameras is not time-synchronized with the radar sweeps and therefore $\Delta t \neq 0$ for most cases. Since the output of the radar is only 2-dimensional, the height information in the coordinate z of the radar frame is not of relevance. However it is not removed from $\bar{\mathbf{p}}_{i,\mathrm{R}}$ since it is relevant in the following, e.g. in the transformations regarding early fusion.

Spatio-temporal error compensation

To precisely match radar points with features or a position in the image plane, the radar points have to be expressed in the cameras coordinate frame. Further, since the sensor modalities are not time-synchronized, the ego vehicles movement during the time-frame Δt introduces a spatio-temporal error between the measurements of camera and radar that can be compensated for using the transformations provided by the *nuScenes* dataset. Both of these problems can be tackled with the transformations described in the following.

Each sensor measurement comes with the pose of the ego vehicle at the time instance t the measurement was triggered. The pose is calculated using the ego vehicles GPS and IMU sensor units and is subsequently transformed into a transformation matrix $\mathbf{T}_{\mathrm{EG},t}$, see Section 2.3.2, that describes the rotation and translation of the ego vehicle frame E towards a fixed, global coordinate frame G at the time instance t of the sensor measurement. Since the global frame is fixed in time, the transformation of the ego vehicle coordinate system describing its movement during Δt is

$$\mathbf{T}_{\mathrm{E},\Delta t} = \mathbf{T}_{\mathrm{EG},t_\mathrm{C}} \mathbf{T}_{\mathrm{GE},t_\mathrm{R}} \in SE(3) \tag{3.3}$$

where $\mathbf{T}_{\mathrm{GE},t_R}$ refers to the inverse transformation from global to ego vehicle coordinates. In consequence, the spatio-temporal compensation is performed via

$$\mathbf{T}_{\mathrm{E},\Delta t} \mathbf{T}_{\mathrm{ER}} \begin{bmatrix} \tilde{\mathbf{p}}_{i,\mathrm{R}} \\ 1 \end{bmatrix} \tag{3.4}$$

for the euclidean coordinates $\tilde{\mathbf{p}}_{i,\mathrm{R}}$ of the radar point $\bar{\mathbf{p}}_{i,\mathrm{R}}$. The calculation of the euclidean coordinates corresponding to the radar point $\bar{\mathbf{p}}_{i,\mathrm{C}}$ in the cameras frame matching the image in both euclidean transformation between the sensors themselves as well as in spatio-temporal compensation due to the ego vehicles movement, is expressed as

$$\begin{bmatrix} \tilde{\mathbf{p}}_{i,\mathrm{C},t_\mathrm{C}} \\ 1 \end{bmatrix} = \mathbf{T}_{\mathrm{CE}} \mathbf{T}_{\mathrm{E},\Delta t} \mathbf{T}_{\mathrm{ER}} \begin{bmatrix} \tilde{\mathbf{p}}_{i,\mathrm{R},t_\mathrm{R}} \\ 1 \end{bmatrix} \quad \in \mathbb{R}^{4\times 1} \tag{3.5}$$

where $\mathbf{T}_{\mathrm{CE}}$ describes the transformation from the ego vehicle's coordinate frame to the camera's frame.

Besides the euclidean coordinates of the radar points, the components of the radial velocity have to be transformed as well. However, the spatio-temporal transformation of the radial velocities $\bar{\mathbf{v}} = \begin{bmatrix} \bar{v}_X & \bar{v}_Y & \bar{v}_Z \end{bmatrix}^\mathrm{T}$ reduces to a rotational compensation for time as

$$\bar{\mathbf{v}}_{\mathrm{C},t_\mathrm{C}} = \mathbf{R}_{\mathrm{CE}} \mathbf{R}_{\mathrm{E},\Delta t} \mathbf{R}_{\mathrm{ER}} \bar{\mathbf{v}}_{\mathrm{R},t_\mathrm{R}} \quad \in \mathbb{R}^{3\times 1} \tag{3.6}$$

where $\mathbf{R}$ is the rotation matrix out of a transformation matrix $\mathbf{T}$, see Equation (2.13), $\bar{\mathbf{v}}_{\mathrm{C},t_\mathrm{C}}$ are the radial velocities of the radar points expressed in the camera frame compensated for the spatio-temporal error and $\bar{\mathbf{v}}_{\mathrm{R},t_\mathrm{R}}$ expresses the radial velocities in the radar frame at the time-stamp of the radar sweep. Since the radial velocity is captured on a 2D plane as well, the velocity component in Z-direction expressed in the radar frame is 0 at all times. Due to the different coordinate systems, see Figure 2.5, in the camera frame the Y-axis velocity component becomes 0 and can be neglected.

An error-compensated and transformed radar point in the cameras coordinate frame therefore is expressed as

$$\bar{\mathbf{p}}_{i,\mathrm{C}} = \begin{bmatrix} \bar{x}_{C,t_\mathrm{C}} & \bar{y}_{C,t_\mathrm{C}} & \bar{z}_{C,t_\mathrm{C}} & \bar{v}_{X,\mathrm{C},t_\mathrm{C}} & \bar{v}_{Z,\mathrm{C},t_\mathrm{C}}\rho & \Delta t \end{bmatrix}^\mathrm{T} \in \mathbb{R}^7 \tag{3.7}$$

In the following, the indices C and t_C will be removed for the purpose of readability and the point in Equation (3.7) will be represented as:

$$\bar{\mathbf{p}}_i = \begin{bmatrix} \bar{x} & \bar{y} & \bar{z} & \bar{v}_X & \bar{v}_Z & \rho & \Delta t \end{bmatrix}^\mathrm{T} \in \mathbb{R}^7 \tag{3.8}$$

Further, the radial velocities should be compensated for the ego vehicle's velocity as well since the radar sweep does not account for this. In consequence the radar measures a velocity for static objects if the ego vehicle is moving. This compensation is performed by the *nuScenes* dataset and radial velocities compensated for ego-vehicle movement are used.

Camera-radar matching

The FoV of every camera in the *nuScenes* dataset overlaps only with some of the five radars FoVs. The overlapping areas are displayed in Figure 3.2. Each mini-sample consists of one image and the radar points

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_i, \ldots, \mathbf{p}_N \end{bmatrix} \in \mathbb{R}^{7\times N} \tag{3.9}$$

in the camera coordinate frame overlapping with the camera's FoV. To further filter for these points, the points $\bar{\mathbf{r}}$ are projected into the 2D image plane and removed if they do not lie in the cameras FoV, see Section 2.2.2 for a detailed explanation. As a last step the radar points with $z > d_\mathrm{max} = 60\mathrm{m}$ are removed due to less precise measurements with increasing distance.

Radar aggregation over time

The radar point cloud in the *nuScenes* dataset is extremely sparse. Smaller but safety critical objects like motorcycles contain less than one radar point on average, see Figure 3.4, even cars are often represented by one or two points only.

To partly compensate for this lack in data the point clouds are aggregated over time. The decision upon how many radar sweeps to add up has to be made as a trade-off between the increase in data points one gets and the error introduced by doing so. The merging introduces two main errors introduced by ego vehicle motion and external object's motion. The ego vehicle motion error describes the

translational and rotational errors between the points of two radar sweeps introduced through the ego vehicles movement. This error can be compensated for as described above.

However, the external object's motion error refers to the error in measurement that occurs due to movement of objects in the surroundings of the ego vehicle. This error can not be compensated for in consequence of the lack of information on the velocity of surrounding objects and has to be kept in a reasonable value by choosing a small Δt_{sweeps} over which to merge the radar data.

In the experiments performed in the scope of this work, the radar data of $n_{\text{sweeps}} = 3$ radar sweeps is aggregated while compensating for the spatio-temporal error as described above. With a radar capture frequency of $f_{\text{radar}} = 13\text{Hz}$, see Table 3.2, Δt_{sweeps} is approximately

$$\Delta t_{\text{sweeps}} = \frac{n_{\text{sweeps}}}{f_{\text{radar}}} = \frac{3}{13\text{Hz}} \approx 0.23\text{s} \tag{3.10}$$

which is still a reasonable time frame.

Figure 3.5 visualizes the aggregation of radar sweeps over time for one example from the *nuScenes* dataset. While the objects enumerated in the Figures are represented by around 2 radar points only for $n_{\text{sweeps}} = 1$, more radar points get associated with each object when aggregating radar sweeps from the past. However, this also introduces radar points that can not be associated to the object anymore. These points occur due to the external vehicles movement and are encapsulated by yellow ellipses in Figure 3.5c. While other authors choose $\Delta t_{\text{sweeps}} = 1\text{s}$ [42], a smaller number and therefore a smaller error is more suitable for the sensor fusion approach of this work.

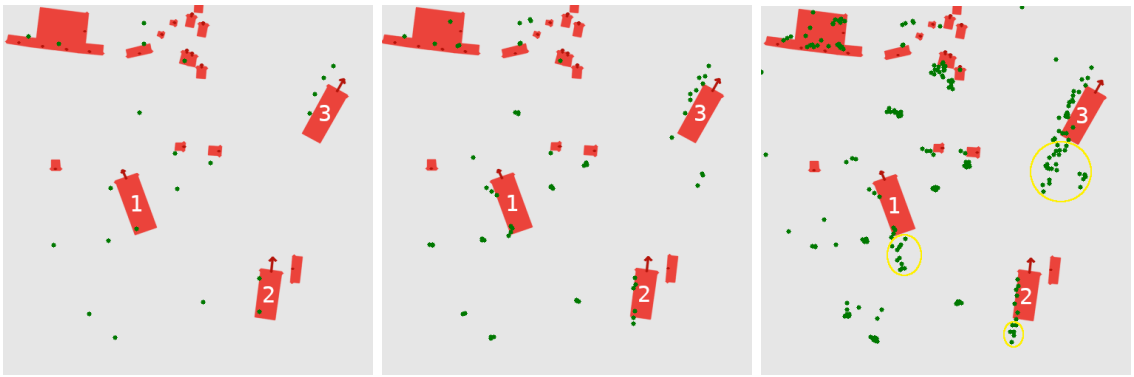
Annotation preprocessing

Besides the input data of radar and camera, the annotations provided by *nuScenes* have to be converted into the format necessary for the network structure of this work. The *nuScenes* object annotations include the information displayed in Table 3.4.

To be able to calculate the metrics introduced in Section 3.4 and the regressions described in Chapter 4, the annotations need to provide the information listed in Table 3.5.

The 3D object center is transformed into the camera frame using a simple transformation matrix $\mathbf{T}_{CG} \in SE(3)$ from the global to the camera frame. The local orientation θ_l is calculated as explained in Section 4.1. The 2D object center is obtained by projecting the 3D object center into the camera plane using the camera matrix, see Section 2.2.2. Similarly, the 2D dimensions are calculated by projecting the corners of the 3D-BB into the image plane and calculating its width and height.

The ground truth velocity v_k of the annotation with index k is calculated by a centered difference between the previous $k-1$ and next frame $k+1$ of the dataset as



(a) $n_{\text{sweeps}} = 1$, $\Delta t_{\text{sweeps}} \approx 0.08\text{s}$ **(b)** $n_{\text{sweeps}} = 3$, $\Delta t_{\text{sweeps}} \approx 0.23\text{s}$ **(c)** $n_{\text{sweeps}} = 13$, $\Delta t_{\text{sweeps}} = 1\text{s}$

Figure 3.5: Visualization of the radar sweep aggregation over time for an example from the *nuScenes* dataset. The radar points are displayed with green dots, ground truth object annotations using red rectangles. The red arrows visualize the absolute value and direction of the object's velocity. Three of the objects are enumerated with indices 1, 2 and 3 for easier comparison. All radar points are compensated for spatial-error due to ego vehicle motion. Figure 3.5a shows a single radar sweep, Figure 3.5b the aggregation of 3 and Figure 3.5c the aggregation of 13 radar sweeps from the past. The yellow ellipses in Figure 3.5c encapsulate the error induced by the external object's motion.

$$v_k = \frac{\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{bmatrix} - \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ z_{k-1} \end{bmatrix}}{t_{k+1} - t_{k-1}} \tag{3.11}$$

where $\begin{bmatrix} x_k & y_k & z_k \end{bmatrix}^{\mathrm{T}}$ expresses the 3D center point of the annotation with index k and t_k the corresponding time stamp. If there is no annotation for the next/previous time stamp, the velocity is approximated using the previous/next annotation only.

3.3 Data augmentation

The performance of deep neural networks heavily depends on big datasets to avoid overfitting the network, see Section 2.1. A common method applied to further enhance the size of the dataset is data augmentation. Literature shows that data augmentation can help to avoid overfitting and make the network more robust towards unknown input data, see [55] and, Section 7.1 of [19].

The type of data augmentation has to be chosen according to the data it is applied to. In the context of this work, e.g. scaling an image is trivial while scaling a radar point cloud changes the 3D measurements which makes it hard to guarantee the same radar points are still matched to the same features in the transformed input image [40, 56].

Table 3.4: Information contained in a single nuScenes object annotation.

Information	Description	Data type
3D Object Center	Euclidean coordinates of the 3D-BB center in global frame	$\mathbb{R}^3$
3D Dimensions	Dimensions of the 3D-BB as width, length, height	$\mathbb{R}^3$
Rotation	Quaternion describing the orientation relative to the global frame	$\mathbb{R}^4$
Category	ID corresponding to the category according to Table A.1	$\mathbb{N}$
Attribute	ID corresponding to the attribute assigned to annotation, IDs are described in Table A.2	$\mathbb{N}$
Number of radar points	Number of radar points within the dimensions of the annotation	$\mathbb{N}$
Number of LiDAR points	Number of LiDAR points within the dimensions of the annotation.	$\mathbb{N}$

Flipping

Flipping the image refers to a flip around the vertical axis. This augmentation can easily be applied to the radar point cloud by inverting the sign of both, the X-component of the euclidean position and radial velocity expressed in the camera coordinate system. The flipped image is visualized in Figure 3.6b. Flipping during training occurs with a default probability of 50%.

Scaling

For scaling, the image pixel coordinates are multiplied with a random scalar $s_{\text{aug}} \in (1 - s_{\text{max}}, 1 + s_{\text{max}})$ where $s_{\text{max}} \in [0, 1)$ can be specified as a hyperparameter. For $s_{\text{aug}} < 1$ the image is *zoomed out*, see Figure 3.6e, for $s_{\text{aug}} > 1$ the contrary occurs, see Figure 3.6f. Scaling is only applied when no radar data is used in the network, e.g. when training the backbone only.

Shifting

Augmentation by shifting describes an offset in both, X- and Z-direction of the image. The offset is applicable in both, negative and positive direction and limited by a hyperparameter as well. The offset calculates as:

$$c_{\text{aug}} = w \; c_r \; c_{\text{max}}, \tag{3.12}$$

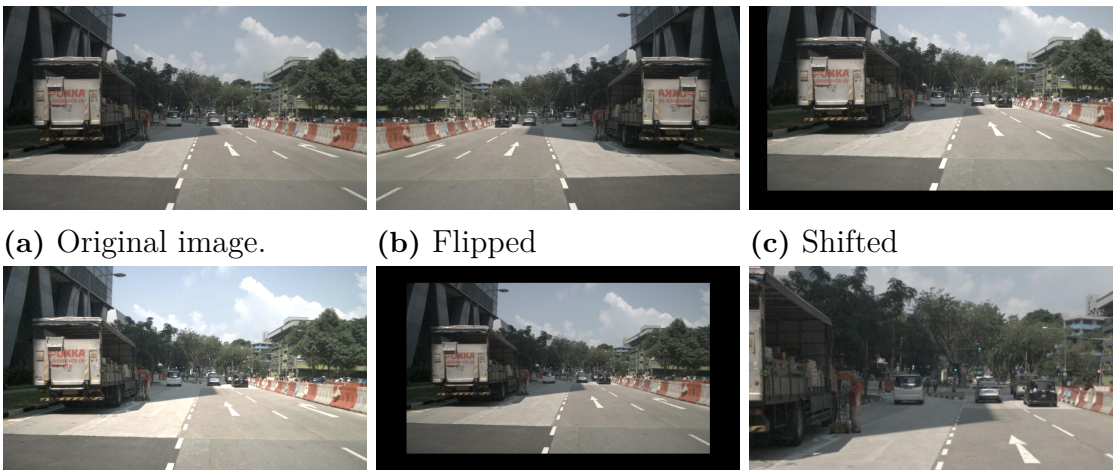
where w refers to the width of the original or not augmented image, $c_{\text{max}} \in [0, 0.5]$ is a hyperparameter defining the amount of possible offset and $c_r \in (-2, 2)$ is a random number sampled from the cropped standard normal distribution. The offset in X and Z is determined separately.

Table 3.5: Object annotations required to calculate the metrics and regressions within this work.

Information	Description	Data type
3D Object Center	Euclidean coordinates of the 3D-BB center in **camera frame**	$\mathbb{R}^3$
3D Dimensions	3D Dimensions of the 3D-BB as width, length, height	$\mathbb{R}^3$
θ_l Rotation	Local orientation θ_l of the object, explained in Figure 4.7	$\mathbb{R}^4$
2D Object Center	Coordinates of the 3D object center projected into the image plane	$\mathbb{R}^2$
2D Dimensions	2D Dimensions of the 3D-BB projected into the image plane	$\mathbb{R}^3$
Velocity	Velocity of the 3D-BB in X, Y and Z expressed in the camera frame	$\mathbb{R}^3$
Category	ID corresponding to the category according to Table A.1	$\mathbb{N}$
Attribute	ID corresponding to the attribute assigned to annotation, IDs are described in Table A.2	$\mathbb{N}$

Color augmentation

Brightness, contrast and color saturation of the input image are adapted as an augmentation method. The amount of color augmentation is randomly selected within a predefined range. An example of a color augmented image is displayed in Figure 3.6d.



(a) Original image. **(b)** Flipped **(c)** Shifted

(d) Color augmentation **(e)** Negative Scaling **(f)** Positive Scaling

Figure 3.6: Different types of data augmentation applied to the original image in Figure 3.6a.

3.4 Evaluation metrics

To evaluate the performance of object detection models, several metrics have been introduced in the past. The most popular evaluation metric is the mAP, which is explained alongside others used in this work in the following.

The mAP metric is based upon the concepts of precision and recall. Precision describes the fraction of true positives among the output of a network while recall is the fraction of true positives over all positives in the annotations [19], Chapter 11.1. They can be expressed as

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{3.13}$$

and

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{3.14}$$

where TP is the number of true positives, FP the number of false positive and FN the number of false negatives. To distinguish between a true positive and false positive detection, the intersection over union (IoU) is widely used as the matching criterion in object detection [57]. The IoU describes the extent of overlap between two bounding boxes. It is expressed as

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{3.15}$$

where A is the annotated and B the estimated BB. If A and B perfectly match in size and position, the IoU is equal to 1. Two BBs A and B match if their IoU exceeds a certain threshold $\in [0, 1]$. However, in the context of a 3D detection task, this approach comes with a major downside since it heavily depends on the size of the regarded object. For the same absolute error in translation, the IoU for a car's BB is bigger compared to a pedestrians 3D-BB. This issue is visualized in Figure 3.7. The

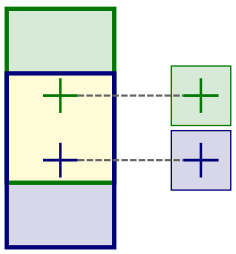


Figure 3.7: Visualization of IoU in BEV for a car (on the left side) compared to a pedestrian (on the right side). Although the centers of the bounding boxes are equally far away from each other, the cars IoU is greater than zero while the pedestrians IoU is 0.0. The overlapping areas are colored in yellow, the predicted BB is depicted in green and the ground truth annotation in blue.

authors of the KITTI dataset [58] tackle this deficit by setting the IoU threshold for larger objects like vehicles to 70% and for smaller objects to 50% [59]. In contrast, the authors of [1] introduce a new matching condition purely based on the 2D BEV

distance between two BBs. This makes the matching independent of object size. The same matching condition is used in this work for a variety of matching distances.

The precision/recall curve describes the curve obtained when plotting precision over recall. The authors of [60] express the average precision (AP) metric as the average over 11 specific recall levels $r \in \{0, 0.1, \ldots, 1\}$ such that

$$\mathrm{AP} = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_{\mathrm{interp}}(r) \tag{3.16}$$

where $p_{\mathrm{interp}}(r)$ describes the interpolated precision for that recall.

The authors of [1] build upon this definition but calculate the AP as the normalized area under the precision recall curve for recall and precision over 10%. The boundaries of 10% are introduced in order to remove the noise in areas of low precision and recall. This definition of the AP is also used in this work since it outputs more precise values. To obtain the final metric mAP, the AP is averaged over different matching thresholds $d \in \mathbb{D} = \{0.5, 1, 2, 4\}$ in meters and the classes $c \in \mathbb{C}$ described in Table A.1 as

$$\mathrm{mAP} = \frac{1}{|\mathbb{C}|\,|\mathbb{D}|} \sum_{c \in \mathbb{C}} \sum_{d \in \mathbb{D}} \mathrm{AP}_{c,d}. \tag{3.17}$$

The mAP metric however comes with two major downsides. First, it does not account for the exact localization error as long as the matching threshold is met. Second, it does not consider the errors in orientation, velocity, BB size and attribute. To evaluate the precision of a network regarding these measures, [1] introduces the NDS metric. It combines the popular mAP metric with five *True Positive* metrics accounting for translation, scale, orientation, velocity and attribute errors, see Table 3.6.

Table 3.6: Overview of the True Positive metrics [1]. All TP metrics are calculated using a matching threshold of 2m and a recall threshold of 10% and are averaged over all classes.

Error	**Description**	**Unit**
mean average translation error (mATE)	Euclidean center distance error in 2D	m
mean average scale error (mASE)	3D IoU after aligning orientation and translation $(1 - \mathrm{IoU})$	–
mean average orientation error (mAOE)	Yaw angle error	rad
mean average velocity error (mAVE)	L2 norm of the velocity difference in 2D	m/s
mean average attribute error (mAAE)	1 minus attribute classification accuracy $(1 - \mathrm{acc})$	–

The overall NDS score is then calculated as

$$\mathrm{NDS} = \frac{1}{10}\left[5\mathrm{mAP} + \sum_{\mathrm{mTP} \in \mathbb{TP}} (1 - \min(1, \mathrm{mTP}))\right] \tag{3.18}$$

where $\mathbb{TP} = \{\text{mATE}, \text{mASE}, \text{mAOE}, \text{mAVE}, \text{mAAE}\}$ includes all True Positive metrics.

4

CenterFusion

When fusing camera and radar data it is difficult to combine the features because it is unclear to which object either information corresponds. The camera could detect one object while the radar only provides data for another. Subsection 2.3.2 gives an overview of approaches tackling this problem. Under these candidates the frustum association is proposed in the work of [40]. The entirety of the architecture in [40], called *CenterFusion*, is a novel solution to fusing camera and radar using deep learning. The authors of *CenterFusion* solve the association problem by creating a proposal with the camera and adjusting it with the point cloud of the radar. For that, the authors introduce the frustum association which is the key idea in this method. It associates a single radar point directly to the detected object. This is important since the learning of the network can then be concentrated on extracting as much information as possible from a single point representing the object and not the whole point cloud. When learning from the entire point cloud, as in [36, 42, 61], there are many measurements from other objects and the surroundings of the ego vehicle on top of the information from the detected object. The additional information makes it harder for the network to extract the important parts about the object that should be detected.

Figure 4.1 shows the high-level network architecture of *CenterFusion*. The idea is to first propose detections by a powerful camera-based object detector that predicts the center points of objects. The detector also predicts a 2D and a preliminary 3D bounding box for each object. The boxes are defined relative to the common center point. This object detection approach is based on the work in *CenterNet* [56]. Both bounding boxes are then used for associating a single radar point from the entire radar point cloud to the object. The associated radar point is used as an additional input to the network's second stage. The 3D-BB proposal is adjusted in the second stage with the information the associated radar point introduces.

The fusion method in *CenterFusion* is on feature level since it uses the predictions of the camera pipeline to influence the radar pipeline. The main drawback of *CenterFusion* is that the detection is reliant on the camera. If the camera does not detect any object in the first stage, e.g. due to environmental conditions or sensor errors, the radar is not used at all. This is because radar points are only used for the prediction when they lie inside the proposal from the camera. When the camera fails to deliver the proposal, none of the radar points are used. The radar only functions as an adjustment of the already detected predictions by the camera. This approach is not robust w.r.t. external interference and does not use the full extent of the fusion potential.

Generally, radar data is relatively sparse compared to camera data. Therefore

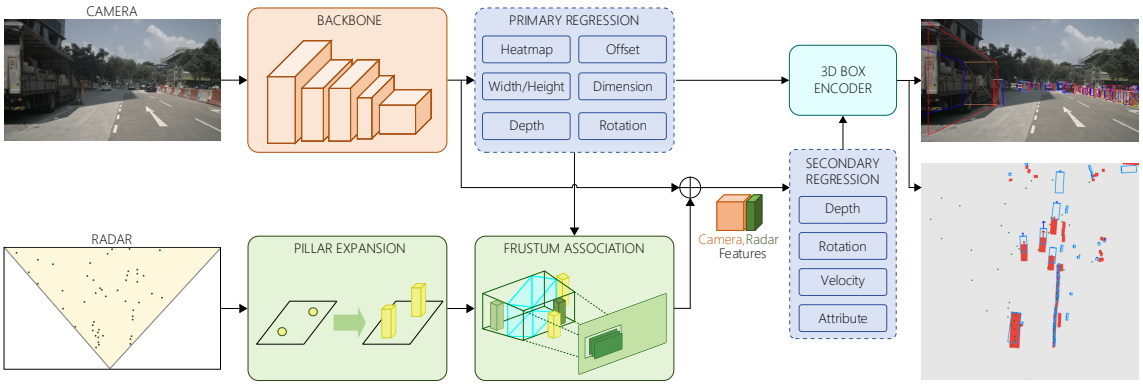


Figure 4.1: Overview of *CenterFusion* where a backbone extracts features to create a 2D-BB and 3D-BB proposal in the primary regression heads [40]. In parallel, the radar points in the camera's FoV, depicted in the lower left image, are expanded over the height information into pillars. The bounding boxes are used in the radar association to find a single radar point that originates from the detected object. The association is based on creating an ROI in the shape of a frustum and only associating pillars that lie inside the frustum to the object. Of the pillars inside the frustum only the closest pillar is used in the rest of the pipeline. The chosen radar pillar introduces additional information to the image-based feature map. The 3D-BB proposal is then fine-tuned in the secondary regression heads with the enriched feature map. Furthermore, additional properties such as velocity and attribute are added as well. The predictions of the primary and secondary regression heads are combined into the final 3D-BB prediction. Results of the pretrained *CenterFusion* model are shown on the right. The upper image shows the projected 3D-BBs while the lower image depicts the BEV of the predictions. In the BEV perspective the red boxes are ground truth, the blue boxes are predictions and the green dots are radar points. The ground truth and predicted velocities are shown as arrows – in the respective color – on all BBs along their axis of orientation.

in *CenterFusion*, the radar data is aggregated in three radar sweeps over time. The position of the points are compensated for ego motion. The aggregation of data increases the chance in finding a radar point for each object and generally improves the radar point association.

The two main ideas of *CenterFusion* – object detection by its center point as in *CenterNet* and frustum based radar point association – are described in this section. First, the method of *CenterNet* is presented in Section 4.1 with which the feature map of the image is created. Second, the regression of the parameters for the 2D-BB and preliminary 3D-BB in the primary heads is explained in Section 4.2. Both bounding boxes are necessary for the frustum association which is explained in detail in Section 2.3.2. Building upon the association, the fusion of the associated radar point to the feature map is explained in Section 4.3. Finally, the secondary regression based on the fused data is presented in Section 4.4.

4.1 CenterNet

CenterFusion builds upon a camera-based 2D and 3D object detector to propose bounding boxes for the frustum creation. For that purpose the work of [56], namely *CenterNet*, an image-based object detector, is used. *CenterNet* can fulfill different tasks such as human pose estimation, tracking as well as 2D and 3D object detection.

The key concept of *CenterNet* is to represent an object as a single point and to predict its center point. In contrary, the classical approach for object detection is to directly regress the object's bounding box such as in classical sliding window approaches [62, 63, 64, 65, 24]. *CenterNet* and sliding window approaches come together in the specific form of anchor-based one-stage detectors where the anchor in *CenterNet* could be seen as a fix shaped 1 pixel anchor. But due to this extreme case of anchor size some parts of the sliding-window approaches are not needed anymore in *CenterNet*. Firstly, the anchors in *CenterNet* are defined only on location and not on overlap and thus there are no manual thresholds that need to be set for classification. Secondly, there is only one anchor per object, due to which there is no need for non-maximum suppression (NMS) in the training process. NMS is first introduced in [66]. Not relying on these parts gives *CenterNet* an advantage in computing speed and enables it to be end-to-end trainable, thus promising higher performance.

For 2D and 3D object detection the network of *CenterNet* is trained to predict a *heatmap* of the input image $I \in \mathbb{R}^{W \times H \times 3}$ where every pixel in the *heatmap* has a certainty value of being an object's center point. W and H are the width and height of the input image respectively. The heatmap $\hat{\mathcal{Y}} \in [0,1]^{\frac{W}{K} \times \frac{H}{K} \times C}$ has C channels to predict the center point of objects in C classes. K determines the output stride w.r.t. the input image I which downsamples the image size by a factor of K. The result of downsizing is the smaller image resolution $|\tilde{I}| = \frac{W}{K} \times \frac{H}{K}$ compared to the original size $|I| = W \times H$. In literature the default output stride is $K = 4$ [67, 68, 69]. A prediction of $\hat{\mathcal{Y}}_{x,y,c} = 1$ means that the object of class c has been detected at the pixel location $\begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}}$. When the network predicts $\hat{\mathcal{Y}}_{x,y,c} = 0$ it assumes this pixel is not an object of class c. If it assumes that for all classes, the pixel is predicted to

be background.

The ground truth center points, called keypoints, $p = \begin{bmatrix} x & y & c \end{bmatrix}^{\mathrm{T}} \in \mathbb{N}^3$ in pixel coordinates are used in a unique way in training as proposed in [70]. They are annotated in the input image with size $|I|$. The ground truth heatmap $\mathcal{Y} \in [0,1]^{\frac{W}{K} \times \frac{H}{K} \times C}$ is generated by laying a 2D Gaussian kernel over the downscaled keypoint image. Figure 4.2 illustrates the Gaussian kernel over a pixel grid for two center points of one class displayed in dark orange and dark green. The rest of the pixels are colored depending on the center point they correspond to and have an opacity proportional to the Gaussian value of either kernel. The downscaled keypoint equivalents are $\tilde{p}_c = \lfloor \frac{p_c}{K} \rfloor = \begin{bmatrix} \tilde{x} & \tilde{y} \end{bmatrix}^{\mathrm{T}} \in \mathbb{N}^2$, where $\lfloor \cdot \rfloor$ is the floor function. Therefore, the ground truth heatmap value at the pixel position $\begin{bmatrix} \tilde{x} & \tilde{y} \end{bmatrix}^{\mathrm{T}}$ in channel c is

$$\mathcal{Y}_{\tilde{x},\tilde{y},c} = \exp\left(-\frac{(\tilde{x} - \tilde{p}_{\tilde{x},c})^2 + (\tilde{y} - \tilde{p}_{\tilde{y},c})^2}{2\sigma_c^2}\right), \tag{4.1}$$

where σ_c is a standard deviation that depends on the typical size of objects in class c. The Gaussian kernel is limited by a variable radius around the corresponding keypoint. If there are multiple objects of one class in an image I the Gaussian kernels of the keypoints overlap. The heatmap value $\mathcal{Y}_{\tilde{x},\tilde{y},c}$ is then computed to be the element-wise maximum of all Gaussian kernels. The overlapping of two Gaussian kernels is displayed in Figure 4.2 as well. Laying a Gaussian kernel over the keypoint image helps the network learn the concept of a heatmap while still providing the necessary information about the actual keypoints.

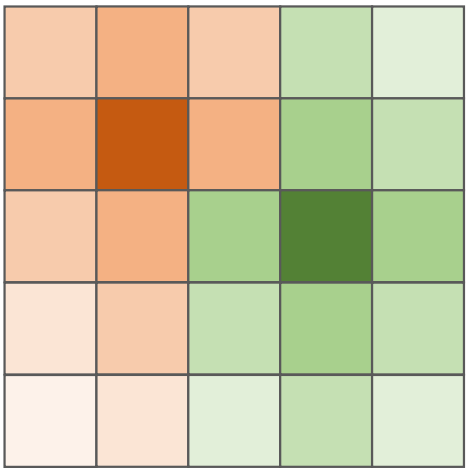


Figure 4.2: The ground truth heatmap $\mathcal{Y}$ shows a grid of pixels with two center points of one class in dark orange and dark green. The rest of the pixel grid is displayed in the color of the center point whose corresponding Gaussian kernel has the maximum value at each pixel.

The advantage of regressing center points of objects is that it enables the network to train without the numerous anchors that are used in sliding window approaches,

this greatly reduces the computational complexity of the training process. Furthermore, the search for center points gets rid of the need to perform NMS. NMS is a post-processing step to determine which anchor gives the best prediction. It uses the IoU of the anchors to each other to pick the prediction that is the likeliest to be true. Although, the calculation of IoU is hard to differentiate and to train the network on. *CenterNet* does not need NMS and thus is end-to-end trainable. This makes *CenterNet* fast in runtime and enables it to achieve a high accuracy [56].

Building upon predicting the center point of objects, other properties of objects such as bounding box dimensions, offset due to striding, depth, orientation, joint locations, etc. can be estimated. The difference between *CenterNet* and sliding window approaches is that the learned properties are parameterized and only the parameters get regressed relative to the common center point and trained in contrast to regressing the whole bounding box at once. In *CenterFusion* the structure of *CenterNet* is used as a 2D and 3D object detector by using a carefully designed head architecture.

In *CenterNet* different encoder-decoder networks are used to extract the feature map which is the input to the head architecture. The authors of *CenterNet* investigated a stacked hourglass network [68, 70], an up-convolutional residual network [71, 72], as well as a DLA network [73]. In *CenterFusion*, as well as in this work, only the latter architecture is used for experiments because it shows the best compromise between speed and accuracy [56]. The DLA architecture is presented in the following section. Since this work is based on DLA, detailed explanations of other candidates are omitted. However, they could be part of further research.

Deep Layer Aggregation

CenterNet needs a powerful encoder-decoder network that can extract rich features out of an image. deep layer aggregation [73] is a deep convolutional image recognition network with specialization on far reaching – or *deep* – skip connections over multiple layer blocks and even stages. In *CenterNet* as well as in this work only the type DLA-34 will be used. There are two versions of DLA-34, one for classification and one for image-to-image tasks. *CenterNet* uses the latter one that consists of a backbone depicted in Figure 4.3 and a neck shown in Figure 4.4. The backbone extracts features by hierarchically connecting – or *aggregating* – blocks of layers and forming larger scale stages. These stages are connected iteratively to each other. The output of each stage is downsampled w.r.t. the preceding stage by a factor of $K = 2$. The neck part of the network builds on top of the backbone to iteratively retrieve the input size of each stage by using a learned bipolar linear interpolation. The result is that the output of the neck is only downsampled by $K = 2$ w.r.t. the original image size $|I|$.

DLA-34 learns to extract the local as well as global information from an input due to the deep aggregation functions. The deep iterative connections help to learn the spatial features and the numerous hierarchical connections improve the semantic learning while also forwarding gradients better to avoid vanishing gradients. The representation of the input is progressively aggregated and condensed inside the network starting from the smallest stride factor iteratively up to the largest.

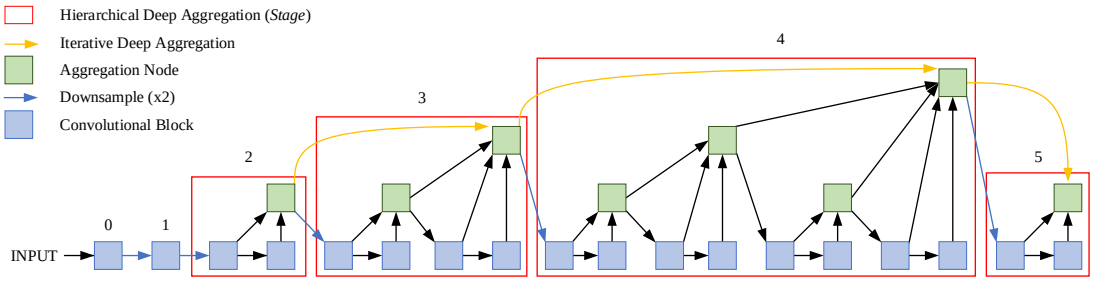


Figure 4.3: The backbone of the DLA-34 network consists of multiple convolutional blocks – depicted as dark blue boxes – which are aggregated hierarchically as well as iteratively [73]. The network therefore learns semantic as well as spatial features from the input image. The nodes in the network that aggregate two streams of data are depicted as green boxes independent on which aggregation type is used. The input image is downsampled by a stride factor of $K = 2$ between each stage – depicted as blue arrows. The hierarchical aggregations are assembled in tree structures that have various depths. These tree structures are defined as *stages* – depicted as red boxes. The stages are numbered 0 through 5. Note that stage 0 and 1 consist of only a single convolutional block. The iterative aggregations – depicted as yellow arrows – are connecting the hierarchically aggregated stages.

The authors in *CenterNet* use the DLA-34 network and adapt the neck structure by omitting the upsampling to the stride factor of $K = 2$ and thus retrieving the image size until a factor of $K = 4$ w.r.t. the original image size $|I|$. Furthermore, they replace the convolutional layers in the neck by deformable convolutional layer (DCL) [21] and add deeper skip connections. The final network used in *CenterNet* is depicted in Figure 4.5.

The concept of the DLA-34 network allows it to work with many different implementations of the backbone and does not restrict it to one specific structure. Note, we refer to the DLA-34 network as a whole in this work by *backbone* although this is not consistent with the definition of the network.

4.2 Primary regression

In the framework of *CenterFusion* the heads of the network play an important part in training the key parameters regressed from the camera and fused – camera and radar – data. They are divided into two stages, primary and secondary regression heads. The primary regression heads extract the parameters for the 2D-BB detection and 3D-BB proposal only from the image and the secondary regression heads adjust and enrich the proposal by using the information from the radar as well.

The primary regression heads are a part of the architecture already proposed in *CenterNet*. The primary heads list the head for the *center point heatmap*, *local offset*, *size* of the 2D-BB as well as the *amodal offset*, *depth*, *dimensions* and *rotation* of the 3D-BB. Figure 4.6 gives an overview over the primary heads. In this section each head with its regressed parameters and its loss function is presented. The primary

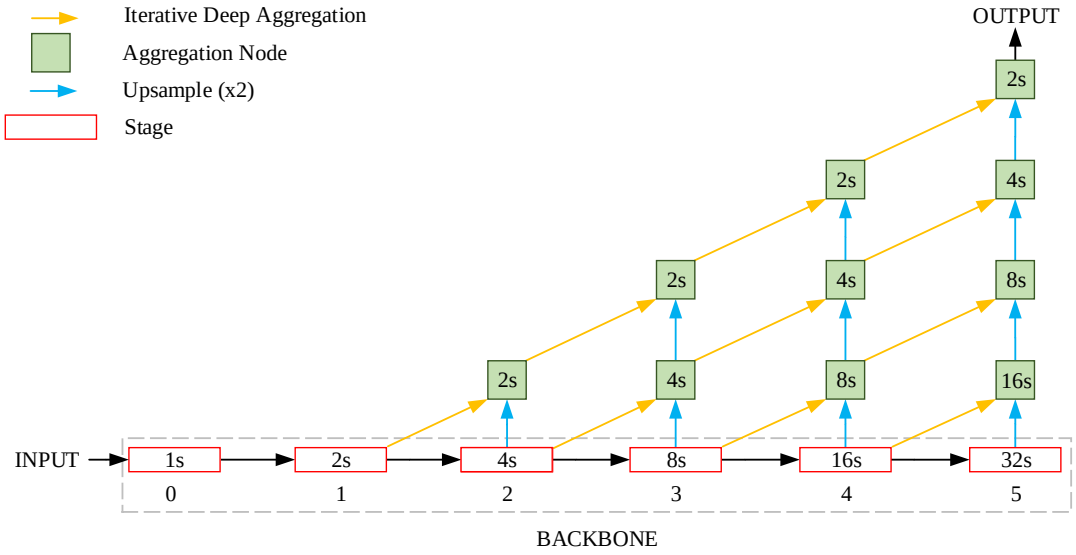


Figure 4.4: The neck of the DLA-34 network builds upon the backbone [73]. The backbone consists of stages numbered 0 through 5 depicted as red boxes. For image-to-image tasks such as object detection the resolution of the downsampled features is retrieved by learned bipolar interpolation. The upsampling is depicted as cyan arrows. The features are then aggregated iteratively – depicted as yellow arrows – with the features of the same resolution from the previous stage. The data streams are aggregated in the nodes depicted in green. The number in each box states the stride K of the input into the node or stage w.r.t. the network input.

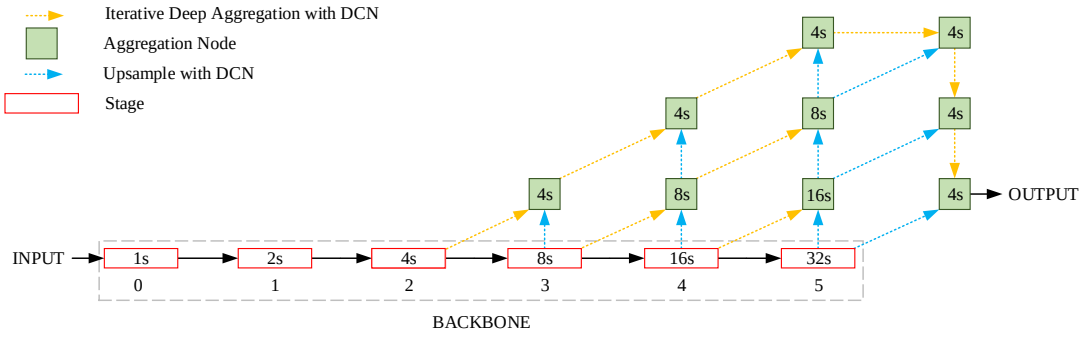


Figure 4.5: In *CenterNet* the neck of the DLA-34 network is modified where the upsampling to the stride factor of $K = 2$ is omitted [56]. The convolutional layers used in the neck are replaced by deformable convolutional layer [21] layers – depicted in dotted arrows. Furthermore, additional skip connections from deep layers to shallow ones are introduced as well.

regression heads all consist of a 3×3 convolutional layer with 256 channels followed by a 1×1 convolutional layer with as many channels as regressed parameters.

The backbone outputs an image-like feature map $FM \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_{FM}}$, with

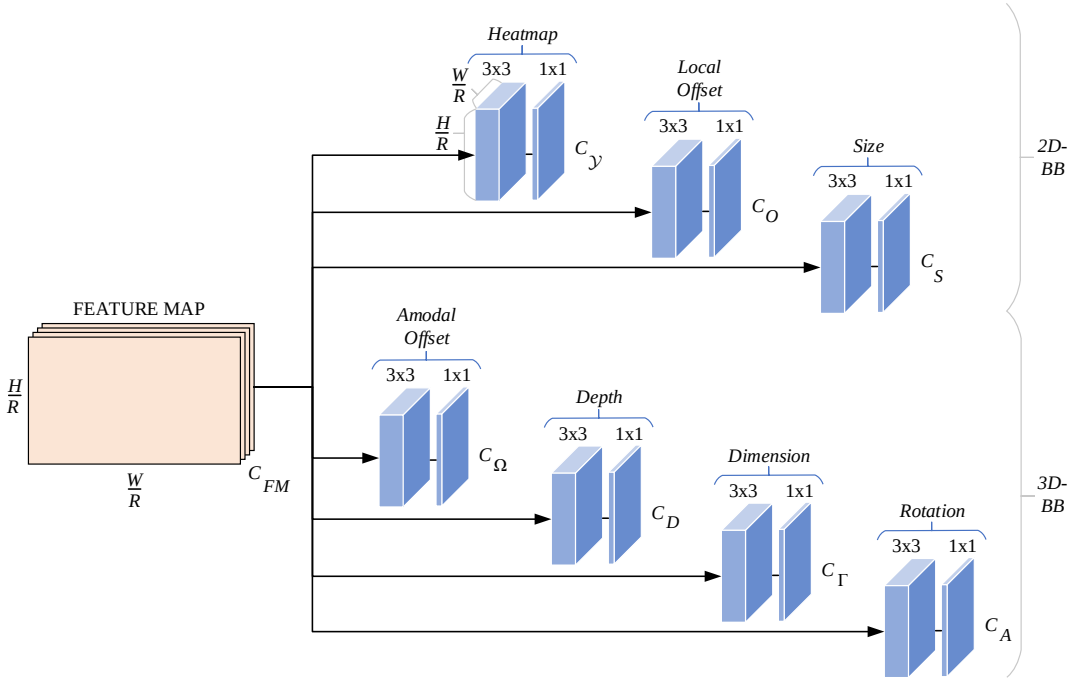


Figure 4.6: The architecture of the primary heads where every head consists of a 3×3 convolutional layer with 256 channels followed by a 1×1 convolutional layer with the number of channels $C_{\mathcal{H}}$ equal to the number of parameters each head $\mathcal{H} \in [\mathcal{Y}, O, S, \Omega, D, \Gamma, A]$ regresses. The feature map FM has C_{FM} channels, which is set to 64, and is the output of the backbone as well as the input for all primary heads. The image size of the feature map is $|\tilde{I}|$ which is the same as the dimensions of the images in and out of the convolutional layers in all heads. At inference the heatmap, local offset and size head are used to assemble the 2D-BB while the amodal offset, depth, dimension and rotation complete the 3D-BB.

$C_{FM} = 64$, for which each head predicts a set of parameters per pixel. During training the predictions are compared to the ground truth values. But only those predictions are used that are located at the same pixel in the image-like output as the downscaled keypoints $\tilde{p}$. In this context, location refers to the pixel position $\begin{bmatrix}\tilde{x} & \tilde{y}\end{bmatrix}^{\mathrm{T}}$ in the image from the backbone which is downsampled by a factor of K compared to the original input image I. Thus, for every keypoint location $\tilde{p}_k$ with $k \in [1, N]$ in the image $\tilde{I}$ only one prediction per parameter is used in training while all other locations are ignored. N is the number of objects in the image $\tilde{I}$. Every image I from the dataset contains a fixed number of objects $M \in \mathbb{N}$ for computational reasons. For many images however, the number $N \in \mathbb{N}$ of annotated objects is smaller than M. The difference $M - N$ is filled up with so-called default annotations. The heatmap head is the only head trained to *not* predict an object if there is none since its prediction is compared over all pixels in the output $\hat{\mathcal{Y}}$ to the ground-truth heatmap $\mathcal{Y}$. Only predictions for which there are corresponding ground truth parameters present are used for all other loss functions, except for the rotation loss.

Center point heatmap

The head for the center point heatmap has $C_{\mathcal{Y}} = C$ channels for simultaneously locating the center point in the image $\tilde{I}$ as well as classifying the object. The heatmap head outputs the heatmap $\hat{\mathcal{Y}} \in [0,1]^{\frac{W}{K} \times \frac{H}{K} \times C}$. The heatmap represents the confidence value $\hat{\mathcal{Y}}_{\tilde{x},\tilde{y},c}$ for each pixel $\begin{bmatrix}\tilde{x} & \tilde{y} & c\end{bmatrix}^{\mathrm{T}}$ in the image $\tilde{I}$ to be a center point of an object. In training, the heatmap is regressed with the focal loss $L_{\mathcal{Y}}$ [24]

$$L_{\mathcal{Y}} = -\frac{1}{N} \sum_{c} \sum_{\tilde{y}} \sum_{\tilde{x}} \begin{cases} \left(1 - \hat{\mathcal{Y}}_{\tilde{x},\tilde{y},c}\right)^{\alpha} \log\left(\hat{\mathcal{Y}}_{\tilde{x},\tilde{y},c}\right) & \text{if } \mathcal{Y}_{\tilde{x},\tilde{y},c} = 1 \\ \left(1 - \mathcal{Y}_{\tilde{x},\tilde{y},c}\right)^{\beta} \hat{\mathcal{Y}}^{\alpha}_{\tilde{x},\tilde{y},c} \log\left(1 - \hat{\mathcal{Y}}_{\tilde{x},\tilde{y},c}\right) & \text{otherwise} \end{cases} \tag{4.2}$$

where α and β are hyperparameters of the focal loss [24]. With the normalization by N all positive losses are normalized to 1. In *CenterNet* the authors chose $\alpha = 2$ and $\beta = 4$ as did the authors of [70].

Local offset

The head for the local offset is introduced to make up for the influence from downsizing the image I inside of the backbone. The reduction of the resolution from $|I|$ to $|\tilde{I}|$ means that any feature at the lowest point in resolution covers a relatively large area in the input image I. The difference of the discrete downscaled keypoint $\tilde{p}_{k,c_k} \in \mathbb{N}^2$ to the continuous keypoint $\frac{p_{k,c_k}}{K} \in \mathbb{R}^2$ is called *local offset*. The prediction of distances in the image plane is due to the local offset error-prone. To help improve the prediction of the center point the local offset $\hat{o} \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_O}$ is learned additionally. The loss L_O for the local offset head is chosen to be the L1-loss

$$L_O = \frac{1}{N} \sum_{k=1}^{N} \left\| \hat{o}_{\tilde{x}_k,\tilde{y}_k} - \left(\frac{p_{k,c_k}}{K} - \tilde{p}_{k,c_k}\right) \right\|_1, \tag{4.3}$$

which is dependent on the keypoint p_{k,c_k} and its downsampled counterpart $\tilde{p}_{k,c_k}$. The head thus tries to predict the error that is introduced by applying the floor function $\lfloor \cdot \rfloor$. Note, the shift of the center point $\begin{bmatrix}\tilde{x} & \tilde{y} & c\end{bmatrix}^{\mathrm{T}}$ by the local offset at inference is therefore not in pixel coordinates anymore. There can be an error in two dimensions, therefore $C_O = 2$.

Size of the 2D-BB

For the radar association, the 2D bounding box is needed additionally to the 3D bounding box. Therefore, the head for predicting the size $\hat{s} \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_S}$ of the 2D bounding box is regressed additionally. The size $\hat{s}$ can be interpreted as the relative width and height to the center of the 2D-BB, thus $C_S = 2$. An axis-aligned 2D-BB can efficiently be stored with only two corners. By convention it is chosen to be the lower left and upper right corner. Thus $\begin{bmatrix}\tilde{x}_k^{(1)} & \tilde{y}_k^{(1)} & \tilde{x}_k^{(2)} & \tilde{y}_k^{(2)}\end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^4$ represents the ground truth 2D-BB of object $k \in [1, N]$, where $\tilde{x}_k^{(l)}$ and $\tilde{y}_k^{(l)}$ represent the Cartesian coordinates of the corner $l \in [1, 2]$ in the X-Y plane. Note, the 2D-BB is not defined

in pixel coordinates. The size of this BB is $s_k = \begin{bmatrix} \tilde{x}_k^{(2)} - \tilde{x}_k^{(1)} & \tilde{y}_k^{(2)} - \tilde{y}_k^{(1)} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^2$. The loss for the size head L_S is chosen to be the L1-loss as well

$$L_S = \frac{1}{N} \sum_{k=1}^{N} \left\| \hat{s}_{\tilde{x}_k, \tilde{y}_k} - s_k \right\|_1 . \tag{4.4}$$

Amodal offset

As the local offset $\hat{o}$ introduces a learnable shift of the center point the amodal offset $\hat{\omega}$ also represents a shift of the center point. While the local offset $\hat{o}$ is applied because of the downsizing of the image, the amodal offset $\hat{\omega}$ is needed because in certain situations objects can only be partially observed by the camera. *Amodal* perception is the detection of entire objects that can only be partially perceived by each individual sensor in a system. Instead of being directly observed by either sensor the object's physical structure is reconstructed. In the context of machine learning, the actual structure is predicted by the network. The reason for not being totally observable by the camera are occlusion by other objects or the environment and the image size restriction – objects can for example extend outside of the image frame. The amodal offset $\hat{\omega} \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_\Omega}$ is trained to fix this issue by learning the offset of the predicted center point to the actual center point. The offset is predicted for the image plane, therefore $C_\Omega = 2$. The amodal offset is trained with the L1-loss

$$L_\Omega = \frac{1}{N} \sum_{k=1}^{N} \left\| \hat{\omega}_{\tilde{x}_k, \tilde{y}_k} - \omega_k \right\|_1 \tag{4.5}$$

where $\omega_k \in \mathbb{R}^2$ is the ground truth amodal offset of the object k.

Depth

For the 3D object detection the image-based network needs to predict depths of objects. The depth head adds the third dimension to the center point predicted in the heatmap head. The depth $\hat{d}$ is generally hard to regress directly which is why the authors of [74] propose an inverse exponential output transformation to

$$\bar{d} = \frac{1}{\sigma(\hat{d})} - 1 = e^{-\hat{d}}, \tag{4.6}$$

where $\sigma(\cdot)$ is the sigmoid function. The transformation has a normalizing effect. The domain of both is $\mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_D}$ but the range of values is smaller for $\hat{d}$. The inverse sigmoidal mapping is applied as the final activation layer which is unique for the depth head. The loss is predicted after the transformation is in the domain where it can be compared to ground truth depths. For the loss of the primary depth head L_D^I the L1-loss is used

$$L_D^I = \frac{1}{N} \sum_{k=1}^{N} \left\| \bar{d}_{\tilde{x}_k, \tilde{y}_k} - d_k \right\|_1 \tag{4.7}$$

with the ground truth depth $d_k \in \mathbb{R}$. Because the depth is scalar also the number of channels is $C_D = 1$.

Dimensions of the 3D-BB

The dimensions of the 3D bounding box are regressed relative to the 3D center point in the X-,Y- and Z-direction with 3 parameters. The dimensions $\hat{\gamma} \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_\Gamma}$ can be interpreted as the height, width and length of the object, thus $C_\Gamma = 3$. The loss for the dimensions head L_Γ uses the L1-loss as well

$$L_\Gamma = \frac{1}{N} \sum_{k=1}^{N} \left\| \hat{\gamma}_{\tilde{x}_k, \tilde{y}_k} - \gamma_k \right\|_1 \tag{4.8}$$

where $\gamma_k \in \mathbb{R}^3$ are the ground truth dimensions.

Rotation of the 3D-BB

The orientation of a 3D bounding box is hard to regress to with camera images. To simplify, only the planar orientation gets predicted, thus it is assumed that all objects are parallel to the ground and that the ground is flat. The authors of [75] divide the orientation into the local orientation θ_l and global orientation θ shown in Figure 4.7. The latter one can be interpreted as the yaw angle of the object or the object's rotation around the camera's Y-axis while the local orientation θ_l is regressed in training. The local orientation θ_l is the angle from the "Ray"-axis, shown in gray in Figure 4.7, to the Z-axis of the object's body fixed coordinate system. Both orientations can be converted from one to another by the angle θ_{ray}, which represents the angle between the object's center point $\begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}}$ and the principal point ψ of the camera and is computed by

$$\theta_{\text{ray}} = \arctan\left(\frac{x - \psi}{f}\right),$$

where f is the focal length of the camera. With the ray angle θ_{ray} the global orientation is

$$\theta = \theta_l + \theta_{\text{ray}}.$$

To further handle the regression, the authors in [76] propose to discretize the planar angle space of the regressed local orientation θ_l into bins and to train with these bins in a similar way as with anchors in sliding window approaches. The relative angle of the bin j to its center angle, denoted by $\Delta\theta_{l,k}^{(j)}$, and the classification score if the angle $\theta_{l,k}^{(j)}$ is in bin j, denoted by $\hat{b}_k^{(j|1)}$, are learnable parameters. The authors of [76] also propose to split up the relative angle into its sine and cosine such that

$$\hat{a}_k^{(j)} = \begin{bmatrix} \hat{a}_k^{(j|1)} & \hat{a}_k^{(j|2)} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \sin\left(\Delta\hat{\theta}_{l,k}\right) & \cos\left(\Delta\hat{\theta}_{l,k}\right) \end{bmatrix}^{\mathrm{T}}$$

is regressed. Thus, each bin can be represented by 3 scalars. In *CenterNet* the authors extend the representation to 4 scalars per bin by introducing another classification parameter $\hat{b}_k^{(j|2)}$ which represents the angle $\theta_{l,k}^{(j)}$ *not* being in bin j. The extension with $\hat{b}_k^{(j|2)}$ is useful in order to let the bins overlap. They also propose to use two bins which amounts to $C_A = 8$ parameters to represent the local orientation

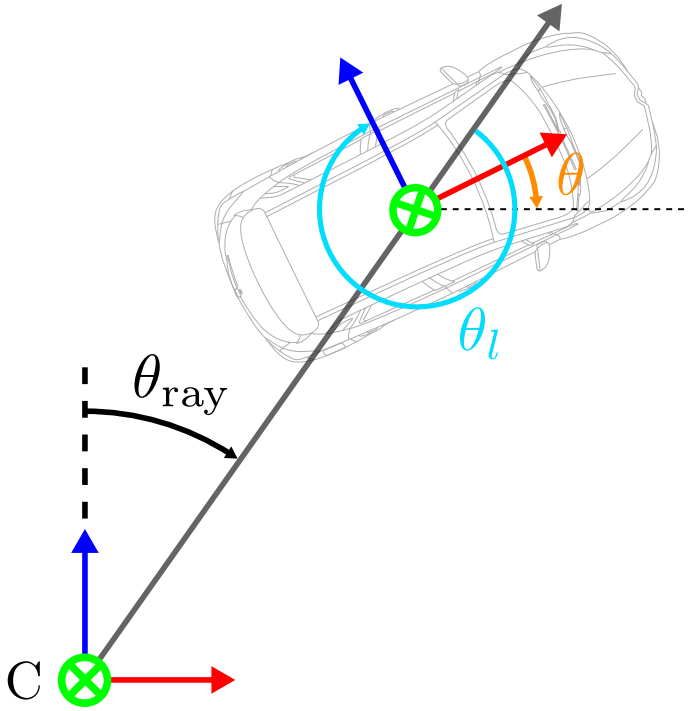


Figure 4.7: The orientation of an object is divided into local orientation θ_l – shown in cyan – and global orientation θ – shown in orange. They can be converted from one to another by the ray angle θ_{ray} – shown in black – which can be derived from the predicted center point in the image plane as well as the camera properties. The global orientation θ is the yaw angle of the object and is computed at inference. In contrary, the local orientation θ_l is regressed in training. The exemplary object shown has a local orientation of $\theta_l = \frac{5}{3}\pi \triangleq -\frac{1}{3}\pi$. The picture of the car in the background is from [51].

θ_l. The parameters are combined in the rotation map $\hat{A} \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_A}$. For the pixel $\begin{bmatrix}\tilde{x}_k & \tilde{y}_k\end{bmatrix}^{\mathrm{T}}$ of the keypoint p_{k,c_k}, the value of the rotation map is

$$\hat{A}_{\tilde{x}_k,\tilde{y}_k} = \begin{bmatrix}\hat{a}_k^{(1|1)} & \hat{a}_k^{(1|2)} & \hat{b}_k^{(1|1)} & \hat{b}_k^{(1|2)} & \hat{a}_k^{(2|1)} & \hat{a}_k^{(2|2)} & \hat{b}_k^{(2|1)} & \hat{b}_k^{(2|2)}\end{bmatrix}^{\mathrm{T}}.$$

Figure 4.8 shows the discretized local orientation angle space $[-\pi, \pi]$ and both bins. The bin 1 is defined by the range $\left[-\frac{7}{6}\pi, \frac{1}{6}\pi\right]$ and bin 2 by $\left[-\frac{1}{6}\pi, \frac{7}{6}\pi\right]$.

The ground truth angle targets are

$$a_k^{(j)} = \begin{bmatrix}\sin\left(\theta_{l,k} - m^{(j)}\right) & \cos\left(\theta_{l,k} - m^{(j)}\right)\end{bmatrix}^{\mathrm{T}}$$

for each bin $j \in [1, 2]$, where the angle $m^{(j)}$ is the mean of the angle range of each bin and $\theta_{l,k}$ denotes the ground truth local orientation of the object k. Note that $m^{(j)}$ is not object dependent. $m^{(1)} = -\frac{\pi}{2}$ and $m^{(2)} = \frac{\pi}{2}$ are the mean values for the chosen bins. The primary rotation loss L_A^I consists of the classification loss, where the binary cross-entropy loss function is used, and the angle loss, which is regressed by the L1-loss. The combined loss is

$$L_A^I = \sum_{k=1}^{M}\sum_{j=1}^{2} -\frac{1}{M}\log\left(\frac{\hat{b}_k^{(j|\xi)}}{\sum_{i=1}^{2}\hat{b}_k^{(j|i)}}\right) + \frac{1}{N}c_k^{(j)}\left\|\hat{a}_k^{(j)} - a_k^{(j)}\right\|_1, \tag{4.9}$$

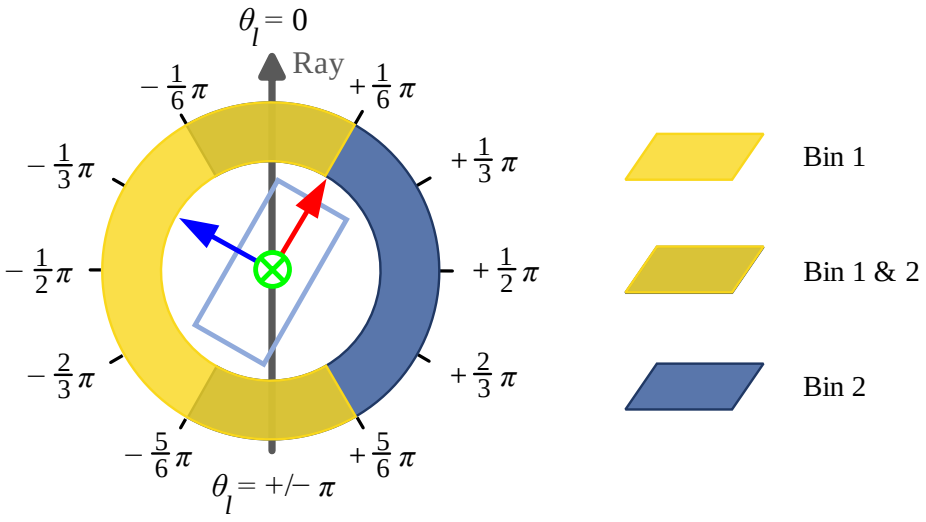


Figure 4.8: The angle space $[-\pi, \pi]$ of the local orientation θ_l is discretized by two bins $j \in [1, 2]$. Bin 1 is defined by the range $\left[-\frac{7}{6}\pi, \frac{1}{6}\pi\right]$ and is displayed in yellow. Bin 2 ranges $\left[-\frac{1}{6}\pi, \frac{7}{6}\pi\right]$ and is shown in blue. The bins are overlapping and the part of the angle space they overlap on is shown in a mixed color. The "Ray"-axis is the line from the origin of the camera coordinate system to the center point of the object and is displayed in gray. The local orientation θ_l is counted from the "Ray"-axis to the Z-axis of the object. Note that the angle θ_l is counted clockwise in this perspective because the Y-axis of the camera is pointing into the plane of projection. An exemplary 3D-BB with its body fixed coordinate system is shown in light blue which has a local orientation of $\theta_l = -\frac{1}{3}\pi$.

where the index $\xi = 1 + c_k^{(j)}$ specifies which bin classification should be penalized. The classification targets are $c_k^{(j)} = 1$ if the angle $\theta_{l,k}$ is inside bin j and $c_k^{(j)} = 0$ if it is not. Note, the loss is computed here over all possible objects M in the image. Although, for all objects k that have default annotations as ground-truth, the outputs as well as the targets are masked by zeros.

Inference

The value for each pixel in the heatmap represents the probability of the corresponding pixel in $\tilde{I}$ to be a center point. At inference the heatmap $\hat{\mathcal{Y}}$ is condensed to its points with highest center point certainty. Firstly, the pixel at position $\begin{bmatrix}\tilde{x} & \tilde{y} & c\end{bmatrix}^{\mathrm{T}}$ with the highest heatmap value $\hat{\mathcal{Y}}_{\tilde{x},\tilde{y},c}$ in every 3×3 region in channel c of the heatmap is saved as a "peak" $\hat{p} = \begin{bmatrix}\hat{x} & \hat{y} & c\end{bmatrix}^{\mathrm{T}} \in \mathbb{N}^3$. Secondly, the K peaks $\hat{p}$ with the highest certainty values over all classes are considered as center point predictions. In *CenterNet* the authors chose $K = 100$ peaks. Thirdly, only peaks with a certainty value $\hat{\mathcal{Y}}_{\hat{p}} \geqslant \tau_{\mathcal{Y}}$ are used as actual outputs of the network. The prediction of center points by using a heatmap is of special importance since all other regression

parameters for the object detection, human pose estimation or tracking are defined relative to the center point and are only given as an output if the corresponding center point has a certainty above the chosen threshold $\tau_{\mathcal{Y}}$. The peak center point filtering is constructed in this way to be able to implement it efficiently with a 3×3-max-pooling operation.

Let $\mathcal{P}_c$ be the set of all n_c used peaks $\mathcal{P}_c = \left\{\begin{bmatrix}\hat{x}_i & \hat{y}_i\end{bmatrix}^{\mathrm{T}}\right\}_{i=1}^{n_c}$ in class c. The sum over all objects n_c in each class is the total number of objects in the image I, i.e. $\sum_c n_c = N$. The regressed parameters are combined at inference to a 2D-BB and a 3D-BB for each predicted center point $\begin{bmatrix}\hat{x}_i & \hat{y}_i\end{bmatrix}^{\mathrm{T}}$.

The 2D-BB is constructed by using the parameters from the head for the heatmap, local offset and size. For each peak $\hat{p}_i$ that the heatmap head predicts, the local offset head estimates $\hat{o}_{\hat{x}_i,\hat{y}_i} = \begin{bmatrix}\delta_{\hat{x}_i} & \delta_{\hat{y}_i}\end{bmatrix}^{\mathrm{T}}$ and the size head outputs $\hat{s}_{\hat{x}_i,\hat{y}_i} = \begin{bmatrix}\hat{w}_i & \hat{h}_i\end{bmatrix}^{\mathrm{T}}$. Note that here the value of the output maps $\hat{o}$ and $\hat{s}$ are extracted at the pixel position of the predicted peak $\hat{p}$ and not at the keypoint $\tilde{p}$ as in training. The 2D-BB of object i has its center point at $\begin{bmatrix}\hat{x}_i & \hat{y}_i\end{bmatrix}^{\mathrm{T}}$ shifted by $\begin{bmatrix}\delta_{\hat{x}_i} & \delta_{\hat{y}_i}\end{bmatrix}^{\mathrm{T}}$ extending $\begin{bmatrix}\hat{w}_i & \hat{h}_i\end{bmatrix}^{\mathrm{T}}$ in each dimension. Calculating the four corners yields

$$B_{2\mathrm{D}_i} = \begin{bmatrix}\hat{x}_i + \delta_{\hat{x}_i} - \frac{\hat{w}_i}{2} & \hat{y}_i + \delta_{\hat{y}_i} - \frac{\hat{h}_i}{2} & \hat{x}_i + \delta_{\hat{x}_i} + \frac{\hat{w}_i}{2} & \hat{y}_i + \delta_{\hat{y}_i} + \frac{\hat{h}_i}{2}\end{bmatrix}^{\mathrm{T}}. \quad (4.10)$$

The 3D-BB is similarly assembled by using the parameters from the head for the amodal offset, depth, dimensions and rotation. For each peak $\hat{p}_i$ all these heads predict their set of parameters. Firstly, the amodal offset $\hat{\omega}_{\hat{x}_i,\hat{y}_i} = \begin{bmatrix}\Delta_{\hat{x}_i} & \Delta_{\hat{y}_i}\end{bmatrix}^{\mathrm{T}}$ shifts the center point of the 2D-BB such that it is no longer dependent on the image size or occlusions. The shifted center point is

$$\mu_i = \begin{bmatrix}\hat{x}_i + \delta_{\hat{x}_i} + \Delta_{\hat{x}_i} & \hat{y}_i + \delta_{\hat{y}_i} + \Delta_{\hat{y}_i}\end{bmatrix}^{\mathrm{T}}.$$

Secondly, the depth head predicts $\bar{d}_{\hat{x}_i,\hat{y}_i} = \epsilon_i$ which introduces the third dimension to the object's center point μ_i. This is done by projecting the 2D center point μ_i into the 3D space as explained in Section 2.2.2. The 3D center point is $c_{3\mathrm{D}_i} = P(\mu_i, \epsilon_i)$.

Thirdly, the dimensions of the 3D-BB are predicted by the dimensions head $\hat{\gamma}_{\hat{x}_i,\hat{y}_i} = \begin{bmatrix}\hat{\gamma}_{X_i} & \hat{\gamma}_{Y_i} & \hat{\gamma}_{Z_i}\end{bmatrix}^{\mathrm{T}}$. The eight preliminary corners of the 3D-BB $B'_{3\mathrm{D}_i} \in \mathbb{R}^{3\times 8}$ are

$$B'_{3\mathrm{D}_i} = \begin{bmatrix} \begin{bmatrix}-\frac{1}{2}\hat{\gamma}_{X_i}\\ \hat{\gamma}_{Y_i}\\ \frac{1}{2}\hat{\gamma}_{Z_i}\end{bmatrix} & \begin{bmatrix}\frac{1}{2}\hat{\gamma}_{X_i}\\ \hat{\gamma}_{Y_i}\\ \frac{1}{2}\hat{\gamma}_{Z_i}\end{bmatrix} & \begin{bmatrix}\frac{1}{2}\hat{\gamma}_{X_i}\\ \hat{\gamma}_{Y_i}\\ -\frac{1}{2}\hat{\gamma}_{Z_i}\end{bmatrix} & \begin{bmatrix}-\frac{1}{2}\hat{\gamma}_{X_i}\\ \hat{\gamma}_{Y_i}\\ -\frac{1}{2}\hat{\gamma}_{Z_i}\end{bmatrix} \dots \\ \begin{bmatrix}-\frac{1}{2}\hat{\gamma}_{X_i}\\ 0\\ \frac{1}{2}\hat{\gamma}_{Z_i}\end{bmatrix} & \begin{bmatrix}\frac{1}{2}\hat{\gamma}_{X_i}\\ 0\\ \frac{1}{2}\hat{\gamma}_{Z_i}\end{bmatrix} & \begin{bmatrix}\frac{1}{2}\hat{\gamma}_{X_i}\\ 0\\ -\frac{1}{2}\hat{\gamma}_{Z_i}\end{bmatrix} & \begin{bmatrix}-\frac{1}{2}\hat{\gamma}_{X_i}\\ 0\\ -\frac{1}{2}\hat{\gamma}_{Z_i}\end{bmatrix} \end{bmatrix}.$$

The rotation head predicts the classification score $\hat{b}_i^{(j)}$ and transformed angle $\hat{a}_i^{(j)}$ for each bin j. The predicted local orientation $\hat{\theta}_{l_i}$ can be reconstructed from these eight parameters

$$\hat{\theta}_{l_i} = \begin{cases} \operatorname{atan2}\left(\hat{a}_i^{(1|1)}, \hat{a}_i^{(1|2)}\right) + m^{(1)}, & \hat{b}_i^{(1|2)} > \hat{b}_i^{(2|2)} \\ \operatorname{atan2}\left(\hat{a}_i^{(2|1)}, \hat{a}_i^{(2|2)}\right) + m^{(2)}, & \hat{b}_i^{(1|2)} \leq \hat{b}_i^{(2|2)}. \end{cases} \quad (4.11)$$

Note that even though $\hat{b}_i^{(1|1)}$ and $\hat{b}_i^{(2|1)}$ are not used in inference they are still influencing the training process. The reconstructed local orientation $\hat{\theta}_{l_i}$ is mapped to the global orientation $\hat{\theta}_i$, or yaw angel, by

$$\hat{\theta}_i = \hat{\theta}_{l_i} + \hat{\theta}_{\text{ray}_i},$$

with the ray angle $\hat{\theta}_{\text{ray}_i}$. Then the preliminary corners of $B'_{3\text{D}_i}$ can be rotated around the Y-axis by $\hat{\theta}_i$

$$B''_{3\text{D}_i} = \begin{bmatrix} \cos(\hat{\theta}_i) & 0 & \sin(\hat{\theta}_i) \\ 0 & 1 & 0 \\ -\sin(\hat{\theta}_i) & 0 & \cos(\hat{\theta}_i) \end{bmatrix} B'_{3\text{D}_i}. \tag{4.12}$$

Finally, the 3D-BB is constructed by shifting the origin of the preliminary BB $B''_{3\text{D}_i}$ by the 3D center point $c_{3\text{D}_i}$

$$B_{3\text{D}_{l_i}} = B''_{3\text{D}_{l_i}} + c_{3\text{D}_i}.$$

Here each corner $l \in [1, 8]$ is translated.

4.3 Radar association

In this section the association of a single radar point from the point cloud to the preliminary camera proposal is described. The frustum association is explained in detail in Section 2.3.2. The input to the frustum association are the 2D-BB, 3D-BB and the point cloud. The bounding boxes are computed as they would be at inference. This means the training process includes the computational effort of inference. In the frustum association the points in the ROI around the 3D-BB are associated to the object. If there are multiple associated points only the closest point is chosen for the fusion. The fusion is explained in detail in the following.

The chosen point r has the characteristics depth $d^{(r)}$, velocity in X-axis $v_X^{(r)}$ and in Y-axis $v_Y^{(r)}$. These characteristics are mapped as a *bounding box heatmap* $\Upsilon \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_\Upsilon}$ into the image plane. Since three characteristics of the radar measurement are chosen it follows $C_\Upsilon = 3$. Figure 4.9 shows the depth channel of the bounding box heatmap Υ on the right 4.9b and the 2D-BB on the left 4.9a. The depth $d^{(r)}$ of the detected objects in the images is represented through color-mapping, where a lighter color corresponds to a bigger depth value. The white areas represent areas where no objects were detected by the camera, and their corresponding heatmap values are set to zero. Note that Figure 4.9 is generated by hand to illustrate the concept and does not contain all objects in the image for the purpose of clarity. The 2D-BBs are shrunk by a scaling factor η and then filled with the radar point's characteristics. The scaling factor η is applied to shrink the area the bounding boxes are covering because the bounding boxes in practice often exceed the objects limits. The shrunken bounding boxes are then filled with the information from the selected radar point. Therefore, the pixels of the bounding box heatmap Υ consisting of the shrunken bounding boxes are set to the chosen point's characteristic $d^{(r)}$, $v_X^{(r)}$ and $v_Y^{(r)}$. Each characteristic is filled into a different channel of the

bounding box heatmap Υ. The mapping of the features $f = \begin{bmatrix} d^{(r)} & v_X^{(r)} & v_Y^{(r)} \end{bmatrix}^{\mathrm{T}}$ for the object j can also be formulated by

$$\Upsilon_{x,y,i}^{j} = \frac{1}{M_i} \begin{cases} f_i, & \left|x - c_X^j\right| \leq \eta w^j \text{ and } \left|y - c_Y^j\right| \leq \eta h^j \\ 0, & \text{otherwise} \end{cases} \tag{4.13}$$

where $\begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}}$ is the pixel position in the image-like bounding box heatmap Υ. w^j and h^j are the width and height of the 2D-BB of object j. The center of the 2D-BB is $\begin{bmatrix} c_X^j & c_Y^j \end{bmatrix}^{\mathrm{T}}$. The parameter M_i is a normalization factor for each channel $i \in [1, 2, 3]$. When there are multiple BBs covering a pixel the closest one – the one with the smallest depth – dominates because closer objects are prioritized. The bounding box heatmap Υ is forwarded to the secondary heads.

(a) RGB image with 2D-BBs. Source of the picture is the dataset [1].

(b) Camera-radar associated bounding box heatmap Υ representing the depth of the object. Lighter color corresponds to larger depth.

Figure 4.9: Visualization of the camera-radar bounding box heatmap Υ association using the frustum approach. Note that this figure is generated by hand to illustrate the concept.

Because it is generally better to train neural networks by normalizing the inputs of a layer [77] the bounding box heatmap Υ is also normalized since it is an input for the secondary heads. But only the depth channel of the bounding box heatmap Υ is normalized by the maximal depth $M_1 = d_{\max}$ allowed for the point cloud, i.e.

$$d_{\text{norm}}^{(r)} = \frac{d^{(r)}}{d_{\max}}.$$

Points that have a higher depth value than the maximal depth are discarded from the point cloud. Thus, the normalization by this threshold $d_{\max}$ is advantageous over batch normalization [77] since it depends on a dataset wide constant in contrast to batch wide constant. The velocity channels are not normalized at all since there exist no boundary on them as with the depth channel and batch normalization would possibly discard important information about the velocities. Therefore, $M_2 = M_3 = 1$.

The generated bounding box heatmap Υ is then concatenated channel-wise with the features of the image, creating a feature map and are further processed in the secondary heads. Figure 4.10 shows the concatenation of the image feature map channels with the radar bounding box heatmap Υ channels.

The fusion of the backbone features and the chosen radar point enables the preliminary proposal from the object detector to be adjusted and additional properties such as velocity to be regressed. This is done in the secondary regression heads.

The procedure presented for the association until here is using the prediction of the network. This holds for inference and therefore validation and testing. When using the network's outputs the predicted 2D- and 3D-BB are possibly not precise which is why the relaxation factor τ_d is introduced. In contrary, while training the ground truth 2D- and 3D-BB are used for association which are accurate. Therefore, in training the relaxation factor τ_d is set to zero to avoid false positive associations.

4.4 Secondary regression

The secondary regression heads are only used in *CenterFusion*. They contain the head for the *velocity* and *attribute* as well as the secondary stage of the *depth* and *rotation* heads. The depth and rotation heads have the same output and loss function – L_D^{II} and L_A^{II} – as in the equivalent primary stage. These two heads are adjusting the preliminary 3D-BB prediction by using the radar information. A depth and rotation head needs to be in the primary stage as well, because they are necessary in the frustum association. The Figure 4.10 shows the structure of the secondary stage including the head architecture. The secondary regression heads all consist of a 3×3 convolutional layer with 256 channels followed by three consecutive 1×1 convolutional layers. The two first also have 256 channels and the last layer has as many channels $C_{\mathcal{H}}$ as parameters regressed in each head $\mathcal{H} \in [V, T, D, A]$. Note that in the publication of *CenterFusion* [40] the authors defined the head architecture to be three consecutive 3×3 convolutional layers and a single 1×1 layer. But the model they achieved the benchmark performance with is the version with three 1×1 layers.

Velocity

The velocity head adds a new dimension to the prediction since velocity can hardly be predicted by the camera network without temporal information. In contrary, the radar directly measures velocity due to the Doppler effect in a single measurement. The head predicts the velocity $\hat{v} \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C_V}$ of the 3D-BB in all three spatial coordinates and thus $C_V = 3$. The fused radar information is especially useful for the prediction. The loss function of the head is the L1-loss

$$L_V = \frac{1}{N} \sum_{k=1}^{N} \left\| \hat{v}_{\tilde{x}_k, \tilde{y}_k} - v_k \right\|_1 \tag{4.14}$$

where $v_k \in \mathbb{R}^3$ is the ground truth velocity.

Attribute

The attribute head predicts additional attributes of objects specific to the dataset used. The dataset worked with in *CenterFusion* is *nuScenes* [1] which is explained in detail in Section 3.1. Attributes describe, among other things, whether a vehicle is moving or standing or if a person is standing, sitting or laying down. A complete list of all possible attributes in *nuScenes* each class of objects has, is given in Table A.2. The head predicts a classification score for the object to be in each class $\hat{t} \in \mathbb{R}^{\frac{W}{K} \times \frac{H}{K} \times C}$. The number of classes in the *nuScenes* dataset is $C = 8$. The loss for the attribute head is the multi-label cross-entropy loss

$$L_T = -\frac{1}{N} \sum_{k=1}^{N} \sum_{c=1}^{C} t_{k,c} \log \left(\frac{e^{\hat{t}_{\tilde{x}_k, \tilde{y}_k, c}}}{\sum_{i=1}^{C} e^{\hat{t}_{\tilde{x}_k, \tilde{y}_k, i}}} \right) + (1 - t_{k,c}) \log \left(1 - \frac{e^{\hat{t}_{\tilde{x}_k, \tilde{y}_k, c}}}{\sum_{i=1}^{C} e^{\hat{t}_{\tilde{x}_k, \tilde{y}_k, i}}} \right) \quad (4.15)$$

with the ground truth $t_{k,c}$ being 1 if the class of object k is c and zero otherwise. This loss function combines the cross-entropy loss with a sigmoid layer $\sigma(\cdot)$ as activation function of the output of the head.

Inference

The prediction of the 3D-BB from both the primary and secondary heads are combined in a 3D box decoder. The decoder uses the secondary rotation and depth prediction combined with the amodal offset and 3D-BB dimensions of the primary heads to estimate the complete 3D-BB for every detected object. Additionally, the prediction is enhanced by the velocity and attribute of the object. The velocity is projected onto the orientation axis of the object

$$\hat{v}_{\text{proj}_{X,Z}} = \|\hat{v}_{X,Z}\|_2 \begin{bmatrix} \cos(\hat{\theta}) \\ -\sin(\hat{\theta}) \end{bmatrix} \quad (4.16)$$

for all predicted velocities $\hat{v} \in \mathbb{R}^3$. Since the orientation is only a planar angle, only the X- and Z-components are projected while the Y-component is not modified, i.e. $\hat{v}_{\text{proj}_Y} = \hat{v}_Y$.

Total Loss

The total loss is the weighted sum of all partial losses from all heads

$$\begin{aligned} L = {} & \lambda_{\mathcal{Y}} L_{\mathcal{Y}} + \lambda_O L_O + \lambda_S L_S + \lambda_\Omega L_\Omega + \lambda_D \left(L_D^I + L_D^{II} \right) + \lambda_\Gamma L_\Gamma \\ & + \lambda_A \left(L_A^I + L_A^{II} \right) + \lambda_V L_V + \lambda_T L_T. \end{aligned} \quad (4.17)$$

The weighting factors $\lambda_{\mathcal{H}}$ with $\mathcal{H} \in [\mathcal{Y}, O, \Omega, D, \Gamma, A, V, T]$ are set to 1. For the size loss the weighting factor is $\lambda_S = 0.1$ which has resulted in the best performance [56].

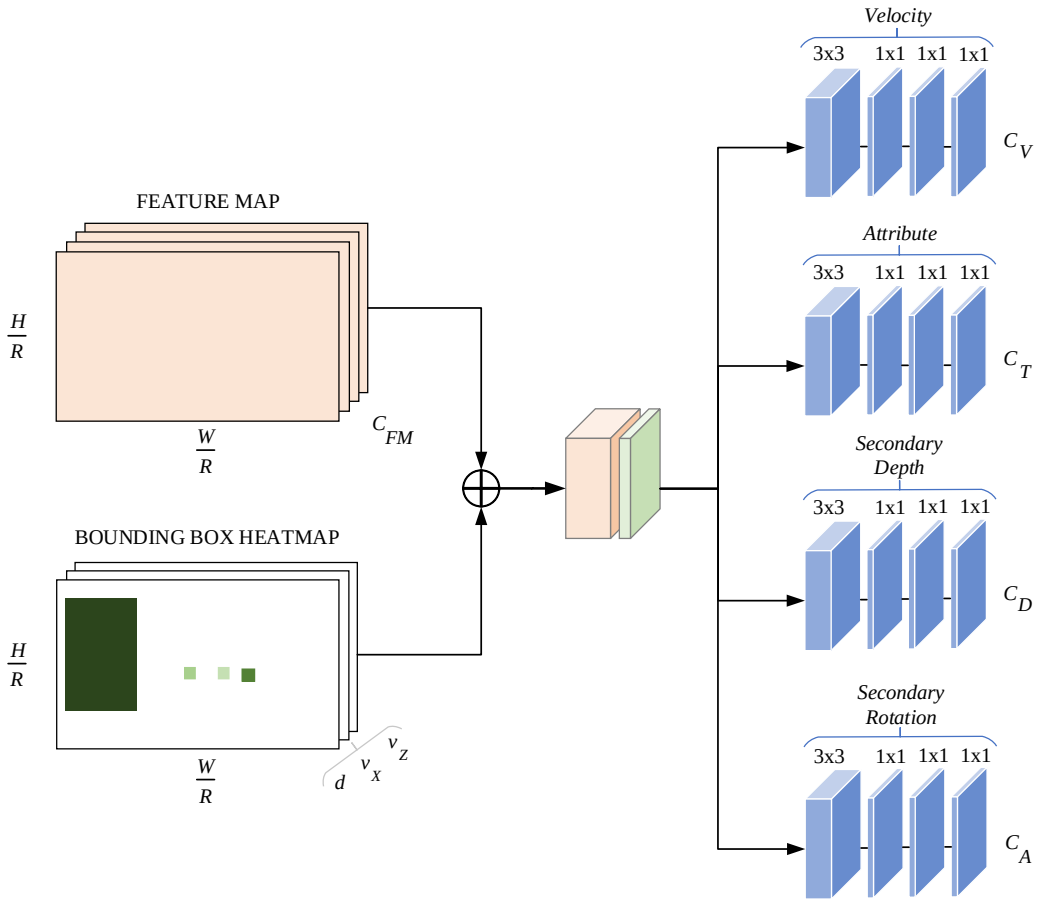


Figure 4.10: Overview of the fusion step and the head architecture of the second stage. For the fusion step, the camera FM is concatenated channel-wise with the radar bounding box heatmap. The camera FM is the output of the image backbone which has $C_{FM} = 64$ channels. The radar bounding box heatmap Υ is the result of the frustum association and mapping of radar data into the image plane and thus has three channels d, v_X and v_Z. The secondary heads take the fused camera-radar feature map as input. The heads are set up with one 3×3 convolutional layer and two consecutive 1×1 convolutional layers with 256 channels. They have a final 1×1 convolutional layer with $C_{\mathcal{H}}$ channels where $\mathcal{H} \in [V, T, D, A]$ at the end. All image-like maps as well as inputs and outputs of the convolutional layers have the resolution $|\tilde{I}|$. At inference the predictions of the secondary heads for depth and rotation adjust the 3D-BB proposal from the primary heads. The heads for velocity and attribute provide additional information for the 3D-BB.

5

CenterFusion++

The deep learning sensor fusion network *CenterFusion* [40], explained in detail in Chapter 4, successfully merges the data of the complementary sensor modalities camera and radar using a novel approach. It is a state-of-the-art network within 3D object detection resulting in promising scores in the metrics introduced in Section 3.4 while also leading the *nuScenes 3D detection task* [50] using camera and radar considering submissions with published papers.

In this work we propose three main improvements that modify the architecture of *CenterFusion*:

1. **RCS**: *CenterFusion* does not include the additional information contained in the RCS channel of the radar. Since it has potential to draw conclusions on the material and shape of a detected object, this might lead to more accurate predictions since false positive detections might be corrected. The inclusion of the RCS feature channel was also experimented with in [78] and resulted in improvements in mAP.

2. **Radar association**: The authors of *CenterFusion* propose to choose the radar point with the smallest depth to the camera in the detected ROI for the fusion step. In the case that multiple radar points are inside the ROI, the information of the remaining points is lost. The association can be improved by considering all radar points in the ROI. To achieve this, a neural network can be designed to handle the information of all points and create an artificial radar point that optimally represents the rest of the points.

3. **Robustness**: The 3D detections of *CenterFusion* depend on a preliminary detection using the camera input only. The complementary information contained in the radar modality is therefore not used for the primary detection but only to adjust the camera's prediction. Although *CenterNet* is a reliable image-based detection network, weather conditions as rain or fog make detections depending on camera less reliable. To tackle the lack of robustness, the radar information can additionally be introduced at an earlier stage of *CenterFusion*.

The proposed ideas motivate the two main adaptions *LFANet* and *EarlyFusion*, bundled under the name "CenterFusion++". They are integrated in the architecture of *CenterFusion* and explained in the following.

5.1 LFANet

In this section we first motivate a new approach to learning from the associated radar points in the frustum. Secondly, the input of the proposed NN is constructed and finally the chosen architecture for the NN itself is explained.

5.1.1 Motivation

The approach of *CenterFusion* is to choose the closest radar point r' for the fusion step which is a rules-based system for association. To compare whether the artificial radar point r^* from *LFANet* improves this rules-based system or not, we analyze their performance w.r.t. the ground-truth depth d of the center point and projected radial velocity v_{rad}. The metrics for this comparison are the depth error $e_d^{(r')}$ and velocity error $e_v^{(r')}$ for the chosen radar point r', which are computed by

$$e_d^{(r')} = \left| d^{(r')} - d \right| \tag{5.1}$$

$$e_v^{(r')} = \left\| v^{(r')} - v_{\text{rad}} \right\|_2 . \tag{5.2}$$

The relative errors to the best alternative of all points r in the frustum are

$$\Delta e_d^{(r')} = e_d^{(r')} - \min_r e_d^{(r)} \tag{5.3}$$

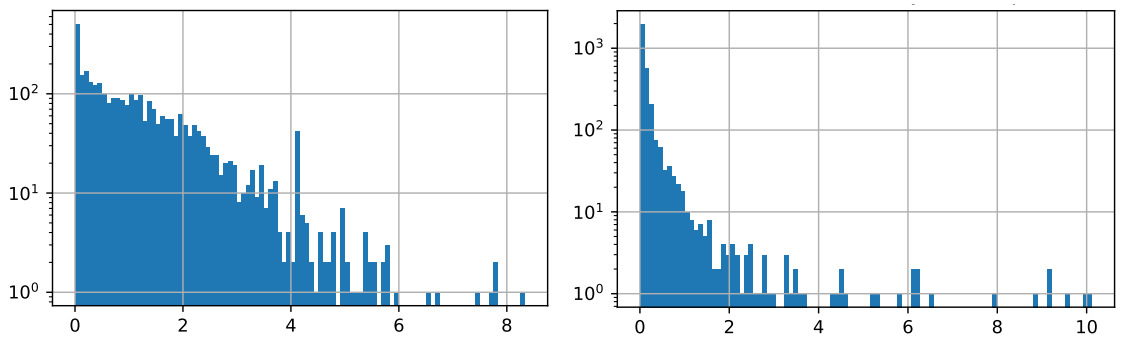
$$\Delta e_v^{(r')} = e_v^{(r')} - \min_r e_v^{(r)}. \tag{5.4}$$

The errors are then normalized by setting them into contest w.r.t. the best alternative over all points r in the frustum. The relative errors e are

$$|e|_d^{(r')} = \frac{\Delta e_d^{(r')}}{\max_r \Delta e_d^{(r)}} \tag{5.5}$$

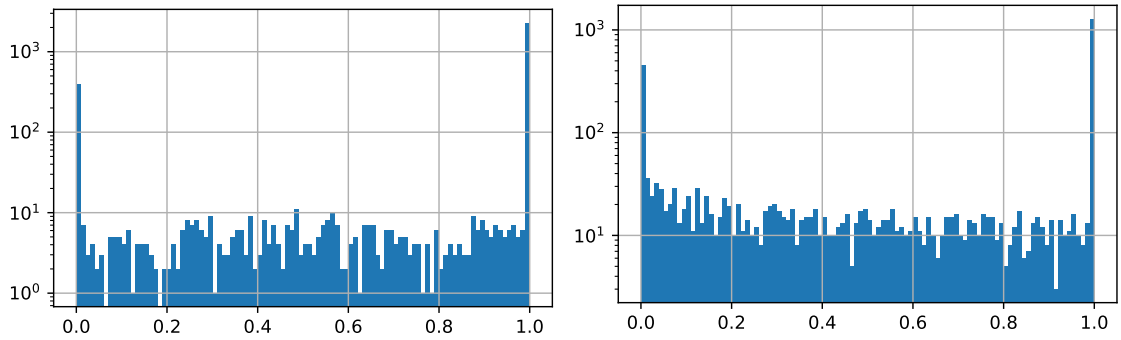
$$|e|_v^{(r')} = \frac{\Delta e_v^{(r')}}{\max_r \Delta e_v^{(r)}}. \tag{5.6}$$

The choice of the closest radar point r' results in the performance as presented in Figure 5.1, where the histogram of all frustums – that have at least one point inside them – is shown. For every frustum the relative errors $\Delta e_d^{(r')}$ and $\Delta e_v^{(r')}$ as well as the normalized errors $|e|_d^{(r')}$ and $|e|_v^{(r')}$ are computed. It is obvious to see that the method of choosing the closest point r' is not always the best option w.r.t. the errors. Especially for scenarios where the closest point is a measurement failure or outlier, for example detections of the environment or other objects. Indeed the peaks at 1 in Figure 5.1c and Figure 5.1d show the cases where the choice of r' was the worst choice that could be made of all possible frustum points r. Additionally, the outliers in Figure 5.1a and Figure 5.1b at around 6-8 m in depth error and 6-10 $\frac{m}{s}$ in velocity error indicate the severity of the choice in extreme situations. The Figure 5.2 shows special cases where the choice of r' leads to bad representations of the object even though with the majority of other points r inside the frustum a more accurate representation is possible. This validates the motivation for another rules-based



(a) Relative depth error $\Delta e_d^{(r')}$.

(b) Relative velocity error $\Delta e_v^{(r')}$.



(c) Normalized depth error $|e|_d^{(r')}$.

(d) Normalized velocity error $|e|_v^{(r')}$.

Figure 5.1: Histograms of the relative depth and velocity errors as well as normalized depth and velocity errors. The errors are indicating the optimality of choosing the closest point r' as proposed in *CenterFusion* compared to all alternative points r in the frustum.

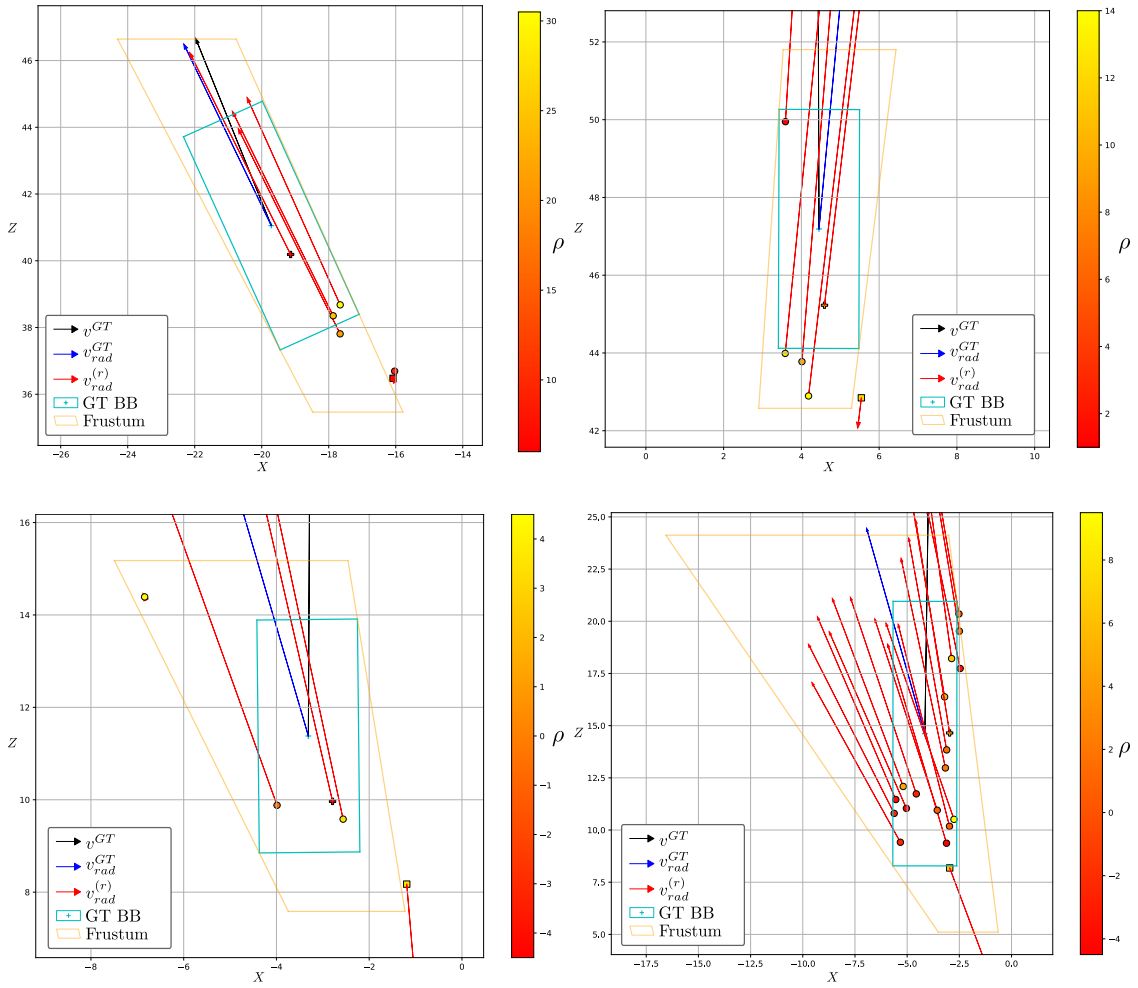


Figure 5.2: Special cases where the choice of the closest radar point r' leads to wrong representation of the object's depth and velocity even though the majority of the alternative points r in the frustum leave a better conclusion to be made. In the figures the dots are the radar points, while special points such as the selected point is marked by a square and the optimal point w.r.t. the error metrics $|e|_d^{(r)}$ and $|e|_d^{(r)}$ is marked by a plus. The color of the markers are set by a scale over the RCS values. The velocity vectors extend from the markers. The object's 3D-BB in BEV is shown in turquoise with its center point. The directional velocity v and radial velocity v_{rad} are depicted in black and blue respectively. The frustum with an expansion ratio of $\tau_d = 0.5$, as chosen in *CenterFusion*, is shown in yellow. For readability the velocity vectors are at some cases not fully shown in the figures.

system as well as the use for a neural network, namely *LFANet*, to extract the most accurate information possible out of the frustum.

The radar association implemented in *CenterFusion* can be improved by adjusting the way the point for the fusion step is chosen out of the constructed frustum ROI. When considering only the closest point to the camera a lot of information is lost while using all points inside the frustum can lead to a more robust use of the radar measurements. The method for filtering the point cloud and associating all points that lie inside the frustum is explained in Section 2.3.2.

Extracting information from all associated points can be achieved in several ways, e.g. with a rule-based system, a stochastic approach or a neural network that predicts the information. We propose to design a neural network that receives a *snapshot* of the frustum and its points as input and outputs an artificial radar point r^* that combines the information of all the radar points contained in the frustum. The snapshot generation is explained in Section 5.1.2. The artificial radar point r^* can be interpreted as the ideal radar measurement of the object, given the actual measurements. Ideal in this context refers to a radar point that represents the depth of the center of the object and is not affected by measurement noise. Therefore, the secondary stage of the *CenterFusion* network can extract more information out of the radar point cloud ROI and adjust the preliminary 3D-BB accordingly.

The network predicting the artificial radar point r^* is termed "Learned Frustum Association Network", in short *LFANet*, which is a novel approach to extract features from a point cloud. Its network architecture is presented in Section 5.1.3.

In the big picture, *LFANet* replaces the *Frustum Association* block in Figure 4.1. The adapted *CenterFusion* network is visualized in Figure 5.3.

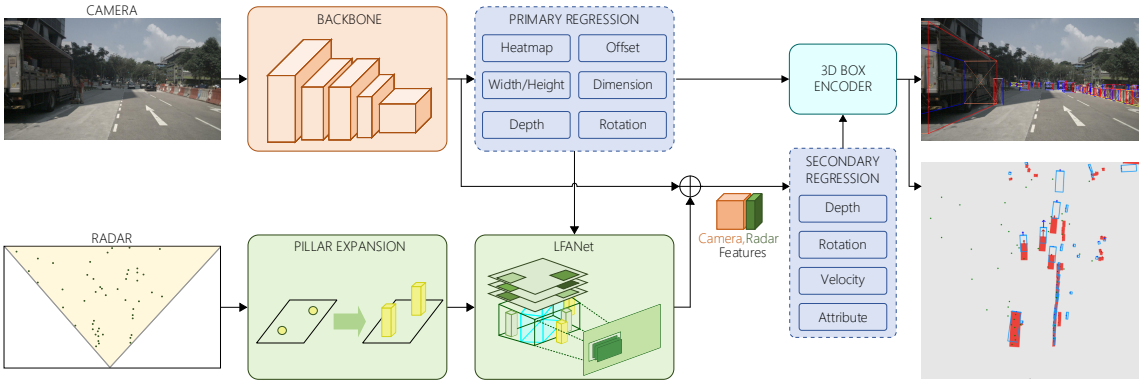
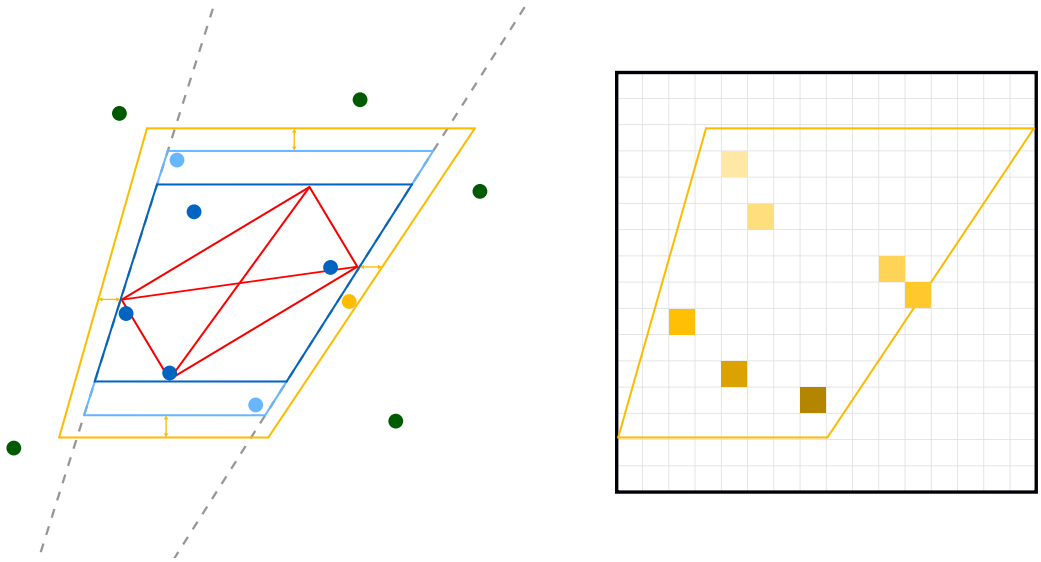


Figure 5.3: *CenterFusion* network architecture including the changes introduced by *LFANet*.

5.1.2 Snapshot

The input to *LFANet* contains all radar points in the ROI, namely the frustum generated by projecting the 2D image detection into the 3D space. Since the network consists of 2D convolutional layers, the radar points in the frustum are expressed as an input image $\mathcal{X} \in \mathbb{R}^{\kappa \times \kappa \times n_{\text{feat,LFA}}}$ in BEV comprising $n_{\text{feat,LFA}}$ channels, where $n_{\text{feat,LFA}}$ is the number of radar features used for *LFANet* and κ is the channel resolution in both dimensions.

Figure 5.4 shows the visualization of the snapshot generation. The initial frustum, displayed in dark blue, is calculated by encapsulating the 3D bounding box of the 2D image object detection by its four corners. The boundaries on the bottom and top are parallel to the X-axis of the camera's coordinate system. The boundaries on the left and right direct towards the origin of the camera's coordinate system. To compensate for uncertainties in the initial estimation, the frustum is enlarged in depth by a factor δ, see Section 2.3.2, Figure 2.9 corresponding to the light blue frustum.



(a) Frustum in BEV.

(b) Single snapshot channel displaying the depth feature

Figure 5.4: Visualization of the snapshot generation. Figure 5.4a shows the BEV of the frustum ROI in the camera's coordinate frame, the gray, dashed lines meet in its origin. The red rectangle corresponds to the 3D object estimation obtained from the 2D image detection. The dark blue frustum expresses the initial ROI obtained through the BEV corners of the detection. This initial frustum is further enlarged by applying the distance threshold expansion δ, see Section 4.3, resulting in the light blue frustum. The *reverse pillar expansion*, explained in Section 5.1.2, results in the final frustum defining the ROI displayed in yellow color. The yellow arrows correspond to half the pillar width w_{pillar}. The dots display the radar points, their colors correspond to the frustum they are assigned to. The green radar points are outside of all frustums and therefore not associated to the object.
Figure 5.4b shows the generated snapshot for a resolution of $\kappa = 16$. The yellow squares are the pixels matching the radar points in dark blue, light blue and orange visualized in Figure 5.4a. Their brightness displays the value assigned to the pixel location, brighter pixels corresponds to bigger depth, i.e. bigger Z-coordinate in the camera's coordinate frame. The yellow frustum is not contained in the actual snapshot but included for visualization only.

Additionally, the authors of [40] introduce three dimensional pillars to account for noise in the measurement of points in the radar point cloud. Analogously, *reverse*

pillar expansion is introduced to the frustum for the generation of the snapshot. Instead of expanding the radar points to pillars, the size of the frustum is adapted in two dimensions such that a radar pillar with width w_{pillar} corresponding to a specific radar point next to the boundaries of the frustum would still be included. The small yellow arrows in Figure 5.4a visualize the extension of the light blue frustum by a distance of $\frac{w_{\text{pillar}}}{2}$ describing the width of the 3D pillar used within *CenterFusion*.

The final frustum, colored in yellow in Figure 5.4a, resembles the ROI used to select the radar points associated with the 2D image object detection. The radar points outside of this ROI are neglected in the further steps. To generate the input image to *LFANet*, namely the snapshot $\mathcal{X}$, the coordinates associated with the radar points $\mathbf{r}_f$ have to be transformed from euclidean coordinates in the camera frame, described as $\begin{bmatrix} x_C & z_C \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}$ into pixel coordinates $\begin{bmatrix} x_s & z_s \end{bmatrix}^{\mathrm{T}} \in \mathbb{N}$.

The coordinates representing the radar points are firstly translated towards the origin such that the center of the frustum, in both width and height, lays in the center of the generated snapshot as well using a transformation matrix

$$\mathbf{T}_{sC} = \begin{bmatrix} 1 & 0 & x_{sC} \\ 0 & 1 & z_{sC} \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.7}$$

The translated coordinates are then scaled by a factor s_{sf} describing the scaling from the euclidean frustum dimensions into snapshot dimensions of resolution $\kappa \times \kappa$ as

$$s_{sf} = \frac{\kappa}{l_{\text{frustum}}}, \tag{5.8}$$

where l_{frustum} is the absolute maximum of width and height of the euclidean frustum. In total, the translation and scaling can be expressed as

$$\begin{bmatrix} x_s \\ z_s \\ 1 \end{bmatrix} = s_{sf} \begin{bmatrix} 1 & 0 & x_{sC} \\ 0 & 1 & z_{sC} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_C \\ z_C \\ 1 \end{bmatrix} \in \mathbb{N} \tag{5.9}$$

describing the pixel coordinates of the radar points $\mathbf{r}_f$ in the frustum.

The $n_{\text{feat,LFA}}$ channels of the the snapshot are then filled at the pixel coordinates with the corresponding information from the radar points within the frustum. To further enlarge the data-density in the snapshot and therefore simplify the training process, we introduce an artificial pillar corresponding to a radar point with $\hat{w}_{\text{pillar}} \in \mathbb{N}$ describing the width of a pillar in pixels. If two pillars overlap, the values corresponding to the closer radar point are dominant.

Figure 5.5b shows a snapshot comprising $n_{\text{feat,LFA}} = 5$ input features, namely the euclidean Z-coordinate in the camera's coordinate frame, the components v_X and v_Z in X- and Z-direction of the radial velocity expressed in the camera's coordinate frame, the RCS value ρ and the time difference Δt between the radar sweep and the time stamp of the corresponding camera image. The latter is included to allow the network to put different focus on the latest radar measurements in order to reduce the effect of errors introduced by external objects movement.

Input normalization

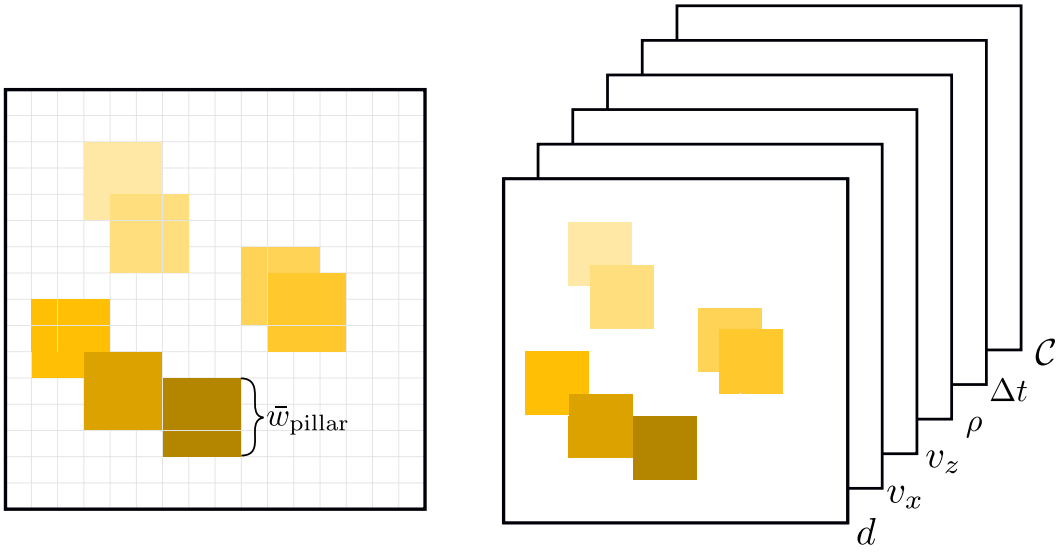
The input channel d corresponding to the depth measurement is normalized by the radar's maximum range d_{max} such that

$$\mathcal{X}_{d,x_s,z_s} = \frac{d_C}{d_{\text{max}}}, \tag{5.10}$$

where $\mathcal{X}_{Z,x_s,z_s}$ corresponds to the value of the input channel Z at pixel coordinates $\begin{bmatrix} x_s & z_s \end{bmatrix}^{\text{T}}$ and d_C is the corresponding depth value of the radar point in the camera's coordinate system. The time difference channel Δt is normalized by a factor

$$\Delta t_{\text{max}} = \frac{n_{\text{sweeps}}}{f_{\text{radar}}} = \frac{n_{\text{sweeps}}}{13} \tag{5.11}$$

describing the biggest time frame possible due too the aggregation of radar sweeps. The other snapshot input channels are not normalized.



(a) Single snapshot channel with $\kappa = 16$ and $\overline{w}_{\text{pillar}} = 3$

(b) Snapshot with $n_{\text{feat,LFA}} = 5$ channels.

Figure 5.5: Visualization of the snapshot corresponding to the frustum and snapshot channel displayed in Figure 5.4 with $n_{\text{feat,LFA}} = 5$ input channels and a resolution of $\kappa = 16$. Figure 5.5a shows the radar points enlarged by the pillar pixel size $\overline{w}_{\text{pillar}} = 3$. Note that the values corresponding to radar points with smaller depth are dominant in case pillars overlap each other.
Figure 5.5b visualizes the complete snapshot used as input for *LFANet*, the 5 channels are concatenated to one single input image.

5.1.3 Network architecture

LFANet receives the snapshot $\mathcal{X}$ of the frustum and outputs the features of the artificial radar point r^*. The structure chosen to process the snapshot is a staged

convolutional network that downsamples the snapshot with the resolution κ to a single pixel with n_{targets} channels. The artificial radar point r^* is defined by the predicted scalars in the n_{targets} channels. In *CenterFusion* the radar point r used for fusion had the characteristics depth $d^{(r)}$, velocity $v_X^{(r)}$ in X-axis and velocity $v_Z^{(r)}$ in Z-axis, thus $n_{\text{targets}} = 3$. The artificial radar point r^* tries to represent the ideal radar point originating from the detected object. Therefore, its properties are the depth d^* of the object's center point as well as the object's radial velocity v^* split up in its X- and Z-component, thus $n_{\text{targets}} = 3$ as well.

The snapshot $\mathcal{X}$ contains semantic as well as global information about the point cloud. Additionally, it inherits a high amount of noise in the set of associated radar points. The noise either comes directly from the uncertainty of the radar or from the approximation of the object by a frustum. The frustum covers a larger area than the object such that points that originate from other objects or the environment around the object are also part of the set. Therefore, the network has to distinguish the outliers and only use information from those points that are close to the object, i.e. center point, and have a velocity vector coherent with the other points in the set.

We propose two approaches that deal with the semantic as well as with the global importance of the radar points in separate ways. The first consists of concatenated convolutional layers that extract features from the snapshot $\mathcal{X}$ as if it would be an image. This approach is focused on extracting semantic features. It iteratively uses convolutional together with pooling layers to downsample the snpashot $\mathcal{X}$ to a single pixel. The second version of *LFANet* is on the contrary focused on extracting global features. It applies two convolutional layers that downsample the snapshot $\mathcal{X}$ in only two steps.

The output of both versions is the prediction for the artificial radar point r^* that represents the set of associated points optimally. The artifical radar point r^* is used in the rest of *CenterFusion's* pipeline as presented in Section 4.3 by projecting the radar properties to the point cloud bounding box heatmap Υ. The regressed properties of the *LFANet* is therefore fused to the feature map FM from the backbone and forwarded as the input to the secondary heads which regress the depth, rotation, velocity and attribute of the detected object.

All convolutional layers used are DCL [21], which means the offset between the kernel pixels are additional learnable parameters.

Iterative approach

The following describes the first approach mentioned above. The iterative setup is inspired by classical image-recognition networks, e.g. [79], therefore it is termed *LFANet*$_{IMG}$. The architecture of the image-based version *LFANet*$_{IMG}$ is shown in Figure 5.6. It downsamples the resolution κ of the snapshot to a single pixel with n_{targets} channels. The network consists of blocks which – except for the two last blocks – are set up by a 3×3 convolutional layer, a ReLU activation function and a $q \times q$ max-pooling operation. A block downsamples the input image exponentially depending on the network depth l. The first size reduction is by a factor of $q = 2$.

All following blocks reduce the size of the input into each block by

$$q(l) = \left\lfloor \min\left(\frac{\kappa}{\Pi_{j=1}^{l-1} 2^j}, \; 2^l \right) \right\rfloor, \quad l > 1,$$

starting with $l = 2$ for the second block. The network's maximal depth l_{max} is therefore in turn dependent on the size κ of the snapshot $\mathcal{X}$. There are as many blocks as downsampling steps are necessary such that the output is a single pixel in size. Therefore, the snapshot size κ can be chosen arbitrarily, although it is practical to choose a resolution that is a power of two. When the final downsizing stage is defined, the output has dimensions of $1 \times 1 \times n_f$, where n_f are the number of filters used in the convolutional layers. Motivated by multilayer perceptrons, see [17], the final stages of *LFANet IMG* are two fully connected linear layers. The first one has n_f nodes while the second maps the number of filters n_f to the n_{targets} parameters needed to predict the artificial radar point r^*. The first layer is additionally followed by a ReLU activation layer.

Note, the convolutional layers are set up in a way that the input size is equal to the output size. They have the parameters given in Table 5.1 where "ConvLayer" followed by the index of network depth and ended by the network version specifies the layer. For *LFANet IMG* every convolutional layer at block depth l has the same parameters.

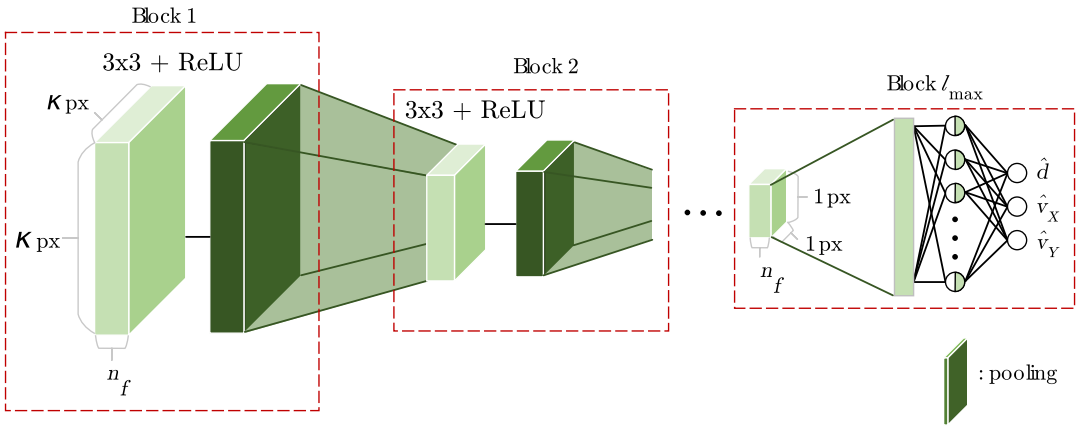


Figure 5.6: The *LFANet IMG* consists of $l_{\max}$ blocks – depicted in red – that are build up from a 3×3 convolutional layer, a ReLU activation function and a max-pooling layer, or by a linear layer in the case of the last two. The first fully connected layer also is followed by a ReLU activation function. The max-pooling layers in dark green. Each block downsamples the size of its input by $q(l)$. The amount of downscale increases with the network depth l until the last convolutional block downsamples it to a scalar with n_f channels. The fully connected blocks extract the n_{targets} parameters to predict the artificial radar point r^*. The depicted downsampling is not to scale but for demonstration purposes only.

Table 5.1: Hyperparameters of the convolutional layers in each version of the *LFANet*. The layers are named by "ConvLayer" followed by the depth of the block the layer is in and its version. The padding is applied with zeros.

Convolutional Layer	nr. filters n_f	kernel size	padding	stride	dilation
ConvLayer-l-IMG	64	3×3	1	1	1
ConvLayer-1-PC	256	7×7	2	3	2
ConvLayer-2-PC	256	8×8	0	1	1

Hierarchical approach

The second version of *LFANet* is using two well-defined convolutional layers for downsampling the snapshot $\mathcal{X}$. It condenses the information in the set of associated points by covering the snapshot $\mathcal{X}$ with a relatively large kernel and striding in relatively large steps along the image to reduce the output size of each layer. This version is motivated by the fact that *LFANet* extracts information from a sparse point cloud. The snapshot $\mathcal{X}$ contains widespread radar points that together contain information about the object they are assumed to originate from. Therefore, the convolutional layers should cover a large area and extract relative information from each point to the others. The network should learn to weigh the importance of points relative to each other, therefore hierarchically differentiating them. Based on this principle this approach is titled *LFANet* $_{PC}$. This version only works for the fixed snapshot resolution of $\kappa = 32$. The architecture of *LFANet* $_{PC}$ is shown in Figure 5.7. The layers are carefully designed such that the first kernel downsamples the snapshot $\mathcal{X}$ to a reduced size that can be completely covered by a second kernel. Therefore with no padding applied, the output of the second layer is scalar. To achieve this, the layers "ConvLayer-1-PC" and "ConvLayer-2-PC" are set up with the parameters given in Table 5.1. The first layer reduces the input size from 32×32 to 8×8 due to the high stride and dilation. The padding is chosen to be 2 to allow the highest coverage of the most pixel in the input while maintaining the given output size. The high dilation enables the kernel to have a larger receptive field without needing a high number of parameters. The dilation still allows the kernel to always match with at least one pixel of the pillars in the snapshot $\mathcal{X}$, even when the pillars have the minimal pixel width of $\overline{w}_{\text{pillar}} = 3$. Some points may be covered more often from the kernel and thus creating an imbalance in weighting of the pillars. The second layer has a kernel the same size as its input, thus – without padding the input – it results in a scalar with n_f channels. After both convolutional layers is a ReLU activation function.

The output of the second layer has the dimensions $1 \times 1 \times n_f$. Therefore, the same concept inspired by multilayer perceptrons can be applied as in the iterative approach. Two fully connected linear layers are added to the pipeline such that the *LFANet* $_{PC}$ extracts the n_{targets} parameters of the artificial radar point r^* as well. The fully connected layers are set up in the same manner as in the iterative approach.

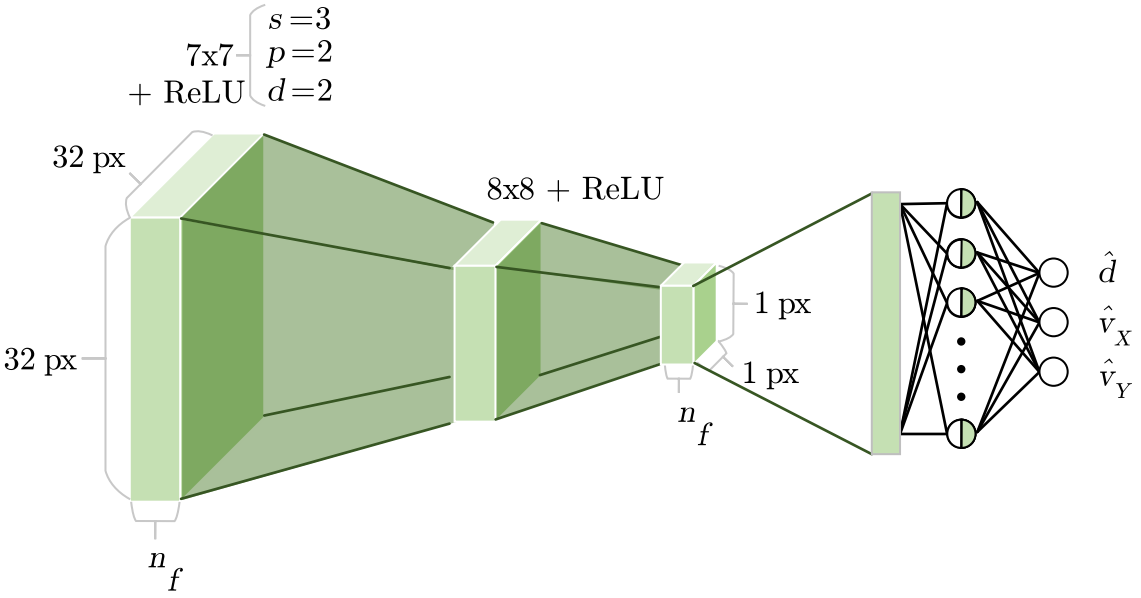


Figure 5.7: The *LFANet* $_{PC}$ contains two convolutional layer and a ReLU activation function. The first layer has a kernel size of 7×7, striding of 3, zero-padding of 2 and dilation of 2. The second convolutional layer has the same kernel size as its input size. Therefore, it returns a scalar with n_f channels which are fed through a fully connected layer including a ReLU activation function. The last linear layer outputs the regressed n_{targets} parameters. The depicted downsampling is not to scale but for demonstration purposes only.

Target & Loss

The *LFANet* is trained by supervised learning where the network makes a prediction for each snapshot given and a loss is computed for these predictions. The loss function returns the error between the predictions and the ground-truth targets. In the training process, the snapshots are created for each labeled object in the dataset. The objects have their 3D location as well as velocity annotated. The velocity of an object is computed in *nuScenes* by comparing its center point over two consecutive keyframes, where only the X- and Z-components are used for the targets. The Z-component of the 3D location is the ground-truth depth d_k of object k. The ground-truth velocity v_k is directed into the object's axis of movement. The *LFANet* would have a straining challenge to predict this velocity vector given only the snapshot of the point cloud. Therefore, we propose that it predicts the projected radial velocity v_{rad_k} instead. The projection of a vector onto an radial axis is shown in Figure 5.8 and is given by

$$v_{\text{rad}_k} = \frac{\overline{OC}_k \cdot v_k}{\left\|\overline{OC}_k\right\|_2^2} \overline{OC}_k \,,$$

where $\overline{OC}_k$ is the vector of direction from the camera's origin to the center point of the object k and $\cdot$ expresses the dot product.

The loss function for the *LFANet* is chosen to be the L1-loss function for all

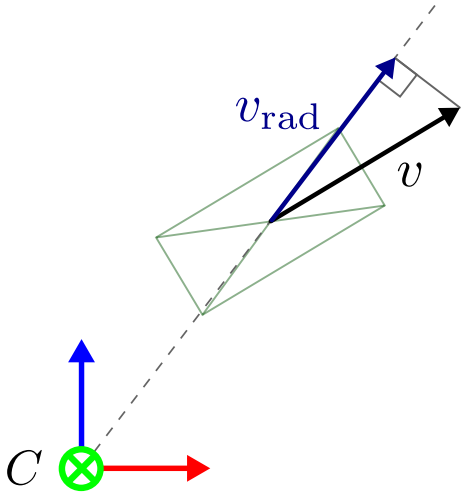


Figure 5.8: Projection of ground truth velocity v into the target velocity v_{rad}.

three parameters, if not mentioned otherwise. The total loss is

$$L_{LFA} = \sum_{k=1}^{N_{\text{snap}}} \lambda_D^{LFA} \left| d_k^* - d_k \right| + \lambda_V^{LFA} \left\| v_k^* - v_{\text{rad}_k} \right\|_1 , \tag{5.12}$$

where $N_{\text{snap}} \leq N$ is the number of snaps that contain a radar point and $v_k^* = \begin{bmatrix} v_{X_k}^* & v_{Z_k}^* \end{bmatrix}^{\text{T}}$. The weighting of the velocity error is chosen to be $\lambda_V^{LFA} = 10$ if not mentioned otherwise, while $\lambda_D^{LFA} = 1$.

5.2 Early Fusion

The outputs of a sensor fusion architecture should ideally not rely on a single sensor modality since this approach does not make use of the full potential of complementary modalities. The radar is more robust regarding challenging environmental conditions, e.g. rain, fog and snow, in which a camera might have difficulties generating robust sensor outputs, see Section 2.2.

The lack of robustness regarding sensor failure and difficult weather conditions in *CenterFusion* described in the introduction to Chapter 5 implies the need for changes in the sensor fusion architecture. Although the term "early fusion" describes a general fusion scheme, see Section 2.3.1, the changes described in the following are referred to as *Early Fusion* within the scope of this work.

To not rely on the camera only for the initial object detection in *CenterNet*, the information contained in the radar point cloud can be projected into the image plane and added as additional channels to the input of *CenterNet*. By doing so, the backbone of the fusion architecture providing the initial object detections is more robust regarding environmental conditions or sensor failure [78].

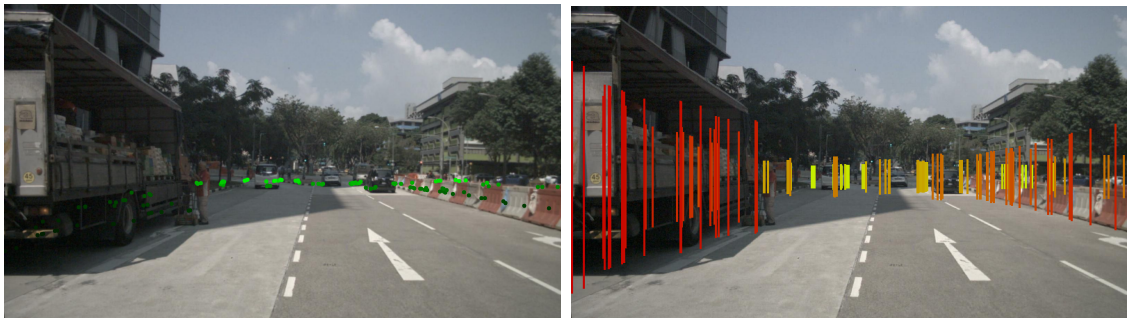
We get inspiration from [42] and implement a similar *Early Fusion* approach. Compared to the camera image, the radar point cloud is very sparse in data. To partly compensate for this problem, we take several measures. First, the point cloud consists of several radar sweeps, see Section 3.2. Second, after projecting the 3D

radar point cloud into the 2D image plane, the points are further enlarged in both dimensions, height and width.

Figure 5.9 shows the comparison of projecting radar points to points or vertical lines in the 2D image plane. By projecting the points to vertical lines, the radar image features become denser in data while not loosing precision in the X-coordinate of the image. The size of the lines is specified in meters such that the height of the lines decreases the further the radar points are from the cameras origin.

The line starts on the ground in the 3D space and ends with the height h_{proj}. The value of h_{proj} assigns a Y value to the radar measurements and therefore also to the associated objects. Bigger values for h_{proj} result in a less sparse input image but also in an increase in imprecision in the Y-coordinate. The authors of [78] found a height of $h_{\text{proj}} = 3\text{m}$ to be the best compromise between a sparse input image and precise associations. Furthermore, an object height of 3m is a reasonable choice for the dynamic objects that have to be expected in the surroundings of an AV.

In addition, the width of each line in the image can be directly specified in pixels which is introduced as a hyperparameter for training, see Chapter 6 for further details. Figure 5.10 visualizes the integration of the *Early Fusion* method into the



(a) Projected radar points **(b)** Projected radar lines

Figure 5.9: Visualization of radar points projected into the 2D image plane. In Figure 5.9a the radar points are projected to a single point in the 2D image only. For visualization, the radius of the points is slightly enlarged, lighter green corresponds to bigger depth. In Figure 5.9b, the radar points are projected onto a line to partly compensate for the sparsity in the radar point cloud, lighter colors correspond to bigger depth. Note that the 2D lines become smaller in height the further the radar points are away from the cameras origin.

CenterFusion architecture.

In order to make use of all the measurements provided in the radar point cloud, namely information on depth, radial velocity and RCS, we introduce a number of $n_{\text{feat},EF}$ feature layers that are concatenated to the RGB input image I as additional image channels. Each image channel contains the corresponding sensor measurement value at the pixel coordinates corresponding to the 2D vertical lines. Analogously to the pre-processing described in Section 5.1.2, the depth values in the Z-coordinate image channel are normalized by the radars maximum range d_{max}. Figure 5.11 visualizes the input I_{EF} image comprising the RGB- feature channels from the camera image as well as the radar feature channels.

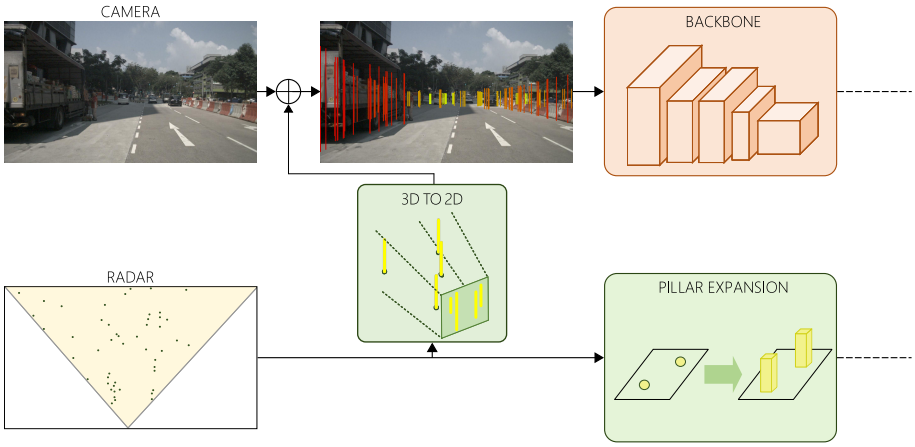


Figure 5.10: Overview of the CenterFusion architecture including the changes introduced through the implementation of *Early Fusion*. Since the remaining fusion parts are not adapted, they are not included in this Figure for better readability.

The newly introduced input I_{EF} replaces the RGB input image in *CenterFusion*. Apart from the size of the input tensor, the network architecture described in Section 4.1 is not further adapted.

Image BlackIn

Dropout is a regularization method for the use in large neural networks in order to reduce overfitting on the training set [80]. The idea is to omit parts of the neural net by randomly deactivating single neurons in the network with a pre-defined probability on every training step. Dropout has proven to make many different kinds of networks more robust [81] and has become a widely used regularization method in the machine learning field.

Inspired by dropouts, the authors of [42] introduce a training strategy that deactivates all input neurons in the input layer, namely *BlackIn*. Effectively, *BlackIn* sets the input data of specific layers to zero for random training steps with a probability of $p_{BI} \in [0, 1]$.

We use the *BlackIn* training method for *Early Fusion*, in the scope of this work the layers affected by *BlackIn* are the channels corresponding to the RGB image of the camera. Consequently, the network trains in the absence of camera input while still receiving the projected radar image channels as input which forces the network to rely more on the input from the radar modality and in turn become less dependent on the camera [42]. This also helps to overcome the bias towards the camera input introduced by transfer learning on a pre-trained, camera-based backbone, see Section 4.1.

5.3 Combination

The two main adaptions to *CenterFusion*, namely *LFANet* and *Early Fusion*, motivated and explained in Section 5.1 and Section 5.2, are combined to use the advan-

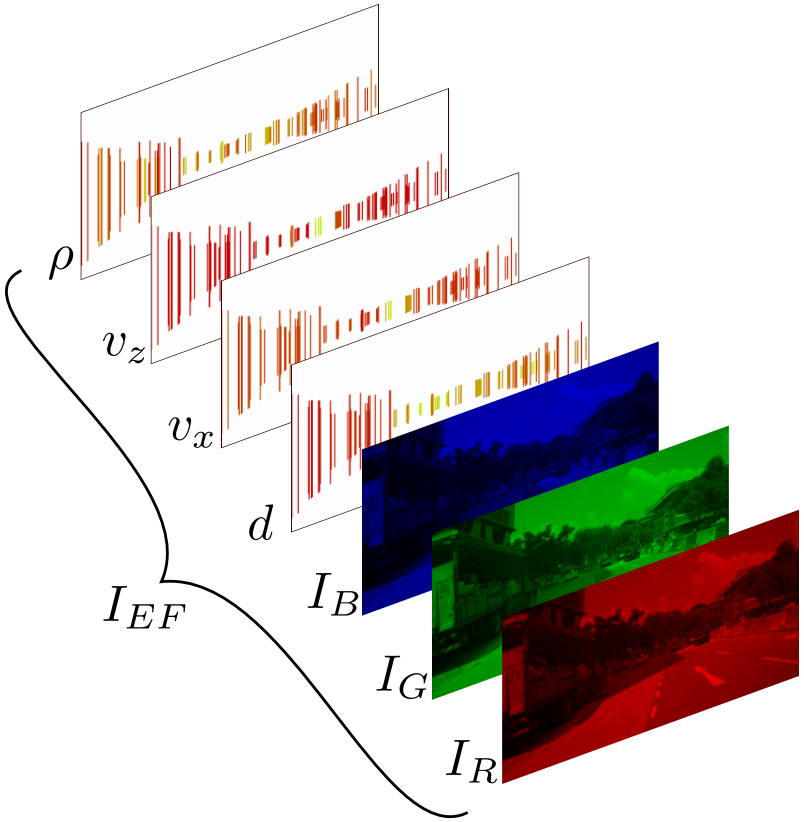


Figure 5.11: Features of the input image I_{EF} used in *Early Fusion* for $n_{\text{feat},EF} = 4$. The additional image channels displayed correspond to the depth, the radial velocity components v_X and v_Z and the RCS value ρ. For all additional image channels, lighter colors correspond to larger values, white displays zeros in the input channels.

tages of both approaches.

The combined network is termed *CenterFusion++* and its fusion scheme is visualized in Figure 5.12.

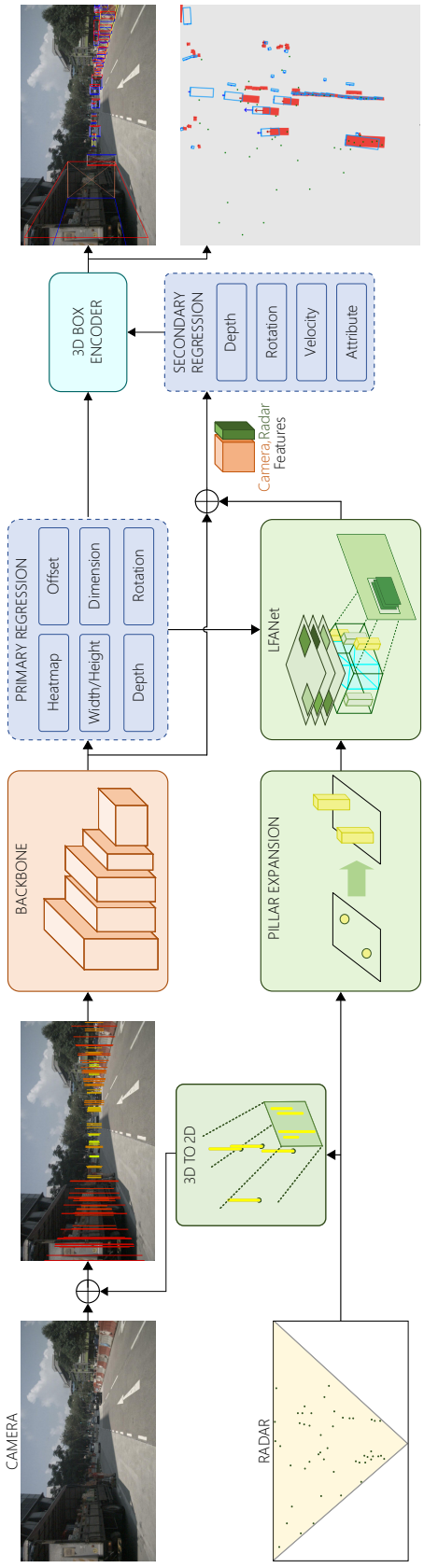


Figure 5.12: *CenterFusion++* overview including the changes introduced by *LFANet* and *Early Fusion*.

6
Results

In this chapter we compare the performances of the adaptions made in *CenterFusion++* to the baselines *CenterNet* and *CenterFusion* separately, to show their standalone effect. Furthermore, we combine the three parts together and validate the complete pipeline, extracting the final performance. The process for all of the following training runs as well as model validations is explained in Section 6.1.

To make sure that we can reach the performances of the original networks *CenterNet* and *CenterFusion* without any modifications, we train them from their corresponding base models as well. The goal is to reproduce the pre-trained benchmark performances of *CenterNet* and *CenterFusion*, which are presented in Section 6.2 and Section 6.3. Assuming we can reproduce the benchmark performances, we can modify the networks and analyze the effect of the modifications.

The introduction of *Early Fusion* mainly increases the robustness of *CenterNet* as well as *CenterFusion* which is further investigated in Section 6.5. The proposed *LFANet* focuses on increasing the accuracy of the pipeline of *CenterFusion* by the revised fusion concept. The analysis of the *LFANet* is given in Section 6.4. Additionally, further ablation studies are provided in Section 6.7.

In the rest of this work, we use the following naming convention, models that were pre-trained by other authors and loaded into the architecture of this work are printed in bold letters, e.g. **CenterFusion$_{60}$**. Models that are trained within the scope of this work are printed in regular letters, e.g. CenterFusion$_{60}$. The subscript corresponds to the number of epochs the model was trained for upon the respective baseline. For example, **CenterFusion$_{60}$** was trained for 60 epochs on the pre-trained **CenterNet$_{170}$** model. Furthermore, in tables comparing models by their metrics, "↑" indicates a higher score is better, while "↓" shows a lower score is better.

6.1 Training and validation process

All training runs for the named models are executed in one of the following modes. The corresponding model can either be trained in mode 1, mode 2, mode 3 or mode 4 while building upon a pre-trained models that often was trained in a different mode. The different modes and the hyperparameters used are summarized in Table 6.1.

Mode 1 is used in the training of the primary heads and the backbone. The authors of *CenterNet* [56] only use mode 1. Mode 2 is divided into two stages: in the first stage, the primary and secondary heads are trained while the backbone is frozen. In the second stage, the backbone is trained as well and therefore the entire model is fine-tuned. *CenterFusion* uses mode 2 while building on top of a

Table 6.1: Differences in the number of epochs and learning rate drops used in the training modes. The table also shows the network architectures for which the individual modes were first constructed.

Mode	# of epochs	learning rate drop at epoch	First proposed for
1	140	90, 120	*CenterNet*
2	Stage 1: 50 Stage 2: 10	50	*CenterFusion*
3	25	20	*LFANet*
4	63	21, 45	*CenterFusion++*

pre-trained model trained in mode 1. Mode 3 describes the training of the *LFANet* architecture on top of a pre-trained model that was trained in mode 2. In mode 4 all heads as well as the *LFANet* are trained together.

The validation of a model means running it at inference over the validation split of the *nuScenes* dataset. We use data augmentation in most validations. The type of validation with data augmentation we use is called "flip-testing". Validating with "flip-test" corresponds to feeding the original and flipped validation split to the network and merging the outputs into one. The images are flipped around the Y-axis, the point clouds around the Z-axis.

6.1.1 Hardware

The training runs described in this chapter were performed on work group stations of the type *NVIDIA DGX A100* [82] using up to four *NVIDIA A100* GPUs [83].

Apart from the training hardware, some validation and inference test runs were executed on a computer powered by a *Intel Core i9-9880H* CPU and a *NVIDIA Quadro T2000 Mobile* GPU termed as the *local* configuration in this chapter.

6.1.2 Hyperparameters

The default hyperparameters used in the training runs are accessible in the `opts.py`[1] script provided in the source code.

6.1.3 Implementation

The implementation of the thesis work is performed in Python3 using the machine learning framework PyTorch [84]. The code is published on GitHub[2].

[1]Default hyperparameters are explained and specified here: https://github.com/brandesjj/centerfusionpp/blob/main/src/lib/opts.py

[2]The code is available on this GitHub repository: https://github.com/brandesjj/centerfusionpp. Some of the models trained within the scope of this thesis work are also available for download.

6.2 Reproducing CenterNet

In *CenterFusion++* we proposed three design changes to the whole architecture, one of them being *Early Fusion*. To evaluate the influence of *Early Fusion*, first we need to validate if we are able to train the original *CenterNet* architecture to the baseline performance. The training of the architecture adapted with *Early Fusion* replaces the *CenterNet* baseline, therefore it is essential to be able to reproduce *CenterNet*'s performance. The baseline is the pre-trained model[3] provided by the authors of *CenterNet*. The authors train on top of the pre-trained DLA-34 network[4] with weights initialized from training 120 epochs on the ImageNet dataset [85]. Our network meant to achieve the same performance as the baseline is trained with the code[5] from *CenterNet* and the same hyperparameters, except we trained with only two GPUs to save resources and thus with a quarter of the batch size to keep the same batch size per GPU. Following the linear relation between batch size and learning rate [86], we also set the learning rate to a quarter of its original value. The deviating parameters are summarized in Table 6.2. Because of the number of epochs trained, namely 140, we term the baseline **CenterNet$_{140}$**.

Table 6.2: Overview of the parameters used in the pre-trained **CenterNet**$_{140}$ training and our reproduction.

Model	# GPUs	batch size	learning rate
CenterNet$_{140}$	8	128	$5 \cdot 10^{-4}$
CenterNet$_{140}$ (Ours)	2	32	$1.25 \cdot 10^{-4}$

The pre-trained model **CenterNet$_{140}$** and our trained model were both trained in mode 1. They achieve the performances in the metrics mAP and NDS given in Table 6.3 when validated with flip-test. Our training, aiming to reproduce the performances of the baseline **CenterNet$_{140}$**, is slightly worse in the mAP but nearly identical to the baseline at the NDS. The mATE and mAP are closely correlated due to the measurements they are representing. Therefore, our training is slightly worse in detecting the correct 3D location of the objects. This discrepancy most likely can be explained due to numerical fluctuations and that the assumed linear relation between batch size and learning rate is not perfect for this architecture. Note, the architecture of *CenterNet* does not contain a head for velocity and attribute. Therefore, default values for the velocity and attribute of an object are returned. Every prediction has a zero velocity vector and the same attribute for each class. Therefore, the metrics mAVE and mAAE do not display *CenterNet*'s performance.

To achieve the result in Table 6.3 we needed to make a key adjustment to the orientation loss function L_A^I. Without it especially the benchmark performance in the mAOE metric could not be reached. The adapted rotation loss is described in

[3]Available at https://github.com/xingyizhou/CenterTrack/blob/master/readme/MODEL_ZOO.md named as "nuScenes_3Ddetection_e140", accessed: 2022/02/24.

[4]The pre-trained model DLA-34 is available at https://dl.yf.io/dla/models/imagenet/dla34-ba72cf86.pth, accessed: 2022/05/23.

[5]The code implementation of *CenterNet* we used is provided at https://github.com/xingyizhou/CenterTrack under the project name *CenterTrack*, accessed: 2022/05/23.

the following and is used in all other training runs as well. In summary, we matched the pre-trained result and we compare the result of our modifications to our baseline model CenterNet_{140}.

Table 6.3: Results of the pre-trained **CenterNet_{140}** and our model CenterNet_{140}. The trained models are validated with flip-test.

			Error ↓				
Model	mAP ↑	NDS ↑	mATE	mASE	mAOE	mAVE	mAAE
CenterNet$_{140}$	**0.3167**	**0.3374**	**0.6886**	0.2600	0.6045	1.4048	**0.6569**
Ours	0.3044	0.3373	0.7133	**0.2582**	**0.5170**	**1.3869**	0.6603
Difference	-0.0123	-0.0001	0.0247	-0.0018	-0.0875	-0.0179	0.0034

Rotation loss

In the original loss function for the rotation loss L_A^I in *CenterNet*, given in Equation (4.9), the error between the prediction and the ground-truth is computed over all M potential objects in the image I. This includes all objects k that are default annotations. The outputs and targets are masked by zeros for all $M - N$ default annotations. However, the bin classifications $\hat{b}_k^{(j|1)} = \hat{b}_k^{(j|2)} = 0$ masked by zero for both bins $j \in [1, 2]$ does *not* result in a suppressed output. Instead, the rotation head is forced to predict with 0% certainty that the angle is in either bin. Crucially, setting the ground-truth classification $c_k^{(j)} = 0$ corresponds to the statement that the angle is in neither bin. The cross-entropy loss function enforces this discrepancy since

$$-\log\left(\frac{e^0}{e^0 + e^0}\right) = -\log(0.5) \approx 0.301 \neq 0. \tag{6.1}$$

The resulting problem in the backward propagation is that the head is optimized by the fact that its classification $\hat{b}_k^{(j|2)} = 0$ is not correct. For the ground-truth $c_k^{(j)} = 0$ the optimal prediction would be $\hat{b}_k^{(j|2)} = 1$. Since this represents the greatest error that the network can produce the optimization focuses on reducing this error, obviously without success. Therefore, the head is trained to mainly predict that the angle is in neither of the two bins. On the other hand, the predicted classifications $\hat{b}_k^{(j|1)}$ for the angle being in bin j are thus relatively small. At inference, the decision of the network whether the angle is in bin 1 or in bin 2, as presented in Equation (4.11), is in some cases decided by numerical fluctuations. The head can still be trained to robustly predict the correct bin but the training process is very sensitive to the choice of hyperparameters and the training process. Our contribution is to skip the loss computation for the default annotations instead of masking their corresponding outputs and targets. The adapted rotation loss

$$L_A^I = L_A^{II} = \frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{2} -\log\left(\frac{\hat{b}_k^{(j|\xi)}}{\sum_{i=1}^{2}\hat{b}_k^{(j|i)}}\right) + c_k^{(j)}\left\|\hat{a}_k^{(j)} - a_k^{(j)}\right\|_1 \tag{6.2}$$

enables the rotation head to train more robustly and increase its performance on the mAOE metric. Note, $\xi = 1 + c_k^{(j)}$. Not only the mAOE benefits from this adaption but also the mAVE, because at inference the velocity is projected onto the orienation axis of the object, as shown in Equation (4.16).

6.3 Reproducing CenterFusion

The pre-trained model **CenterFusion$_{60}$** needs to be replaced by the training of *CenterFusion++*'s architecture which critically changes the point cloud bounding box heatmap Υ. Therefore, the network model must be trained again such that it adapts to the changes in the heatmap Υ. However, we first have to validate that we are able to replicate the *CenterFusion* model, in order to improve it by modifying the association using *LFANet*.

CenterFusion's pre-trained model is trained on top of the *CenterNet* baseline in mode 2. To compare the results of *CenterFusion* to *CenterNet* the authors additionally trained the baseline **CenterNet$_{140}$** for 30 more epochs in mode 1 with an additional velocity and attribute head. Therefore, the velocity and attribute predictions can be compared reasonably. The resulting model is termed **CenterNet$_{170}$**[6]. This pre-trained model is then trained for another 60 epochs in mode 2 on resulting in the *CenterFusion* baseline termed **CenterFusion$_{60}$**[7]. Note, for the training of *CenterFusion* in mode 2 the velocity and attribute heads of **CenterNet$_{170}$** are discarded.

To validate that we are able to reproduce the baseline performance of the original architecture of *CenterFusion*, we train starting from **CenterNet$_{170}$**. We train the network with the same hyperparameters[8] as the authors of *CenterFusion*, although we use only a single GPU. As in Section 6.2 the batch size and the learning rate are divided by two assuming linear dependence between the two [86]. We train the network in mode 2 and therefore in two parts, a first stage for 50 epochs to train the newly initialized secondary heads and the already pre-trained primary heads with a larger learning rate while freezing the backbone and a second stage with 10 epochs to fine-tune the whole network with a smaller learning rate. For the smaller learning rate we choose half of the learning rate of the first stage, unlike the authors of *CenterFusion* who use a tenth after 50 epochs instead. The parameters that differ are summarized in Table 6.4. The results of validating the baseline and our trained model with flip-test is given in Table 6.5.

The results in Table 6.5 show that our training CenterFusion$_{60}$ managed to reproduce the **CenterFusion$_{60}$** model, although the mAP and mATE are sightly worse. The difference likely leads from the reduction in the number of GPUs used, specifically the corresponding change in learning rate w.r.t. the batch size as well as from numerical fluctuations. Note that, similar to the reproduction of the results

[6]The corresponding model "centernet_baseline_e170" is available for download at https://github.com/mrnabati/CenterFusion#pre-trained-models, accessed: 2022/03/01.

[7]The pre-trained model of *CenterFusion* called "centerfusion_e60" is available at https://github.com/mrnabati/CenterFusion#pre-trained-models, accessed: 2022/02/24.

[8]The training parameters are given at https://github.com/mrnabati/CenterFusion/blob/master/experiments/train.sh, accessed: 2022/05/23.

Table 6.4: The hyperparameters differing from the baseline **CenterFusion$_{60}$** and our training CenterFusion$_{60}$. The learning rate is denoted by lr, while lr_{50} represents the learning rate used for fine-tuning after 50 epochs.

Model	# GPUs	batch size	initial lr	lr_{50}
CenterFusion$_{60}$	2	32	$2.5 \cdot 10^{-4}$	$2.5 \cdot 10^{-5}$
CenterFusion$_{60}$	1	16	$1.25 \cdot 10^{-4}$	$6.25 \cdot 10^{-5}$

Table 6.5: Results of validating the pre-trained **CenterFusion$_{60}$** and our replicated model CenterFusion$_{60}$ with flip-test.

			Error ↓				
Model	mAP ↑	NDS ↑	mATE	mASE	mAOE	mAVE	mAAE
CenterFusion$_{60}$	**0.331**	**0.452**	**0.6478**	0.2631	0.5396	0.5418	**0.1426**
CenterFusion$_{60}$	0.3223	0.4502	0.6588	**0.2623**	**0.5008**	**0.5403**	0.1475
Difference	-0.0087	-0.0018	0.0110	-0.0008	-0.0388	-0.0015	0.0049

from *CenterNet*, the mAOE and mAVE is improved because of the modifications on the rotation loss. In conclusion, we were able to approximate the pre-trained result and we will compare the performance of our modifications to our baseline model CenterFusion$_{60}$.

6.4 LFANet

The introduction of *LFANet* to the architecture of *CenterFusion++* for an optimized frustum association should enable the whole network to increase its performance in mAP and NDS compared to *CenterFusion* especially in the location accuracy of the prediction.

The *LFANet* outputs the artificial radar point r^* that represents the depth of the object's center point instead of the closest point as in *CenterFusion*. Although, the secondary heads in CenterFusion$_{60}$ are trained to receive a different information about the object than the *LFANet* provides. Therefore, we need to train a parallel model to CenterFusion$_{60}$ where the adapted heatmap Υ is forwarded to the secondary heads.

During training, the output of *LFANet* is not accurate – in the beginning simply random – and should not be used as the input for the secondary heads while training them. In turn, also the output of the primary heads, which is used for constructing the frustum in the snapshot generation, is not ideal. Therefore, we separate the training of the heads presented in Section 6.4.1 and *LFANet*, presented in Section 6.4.3. During the training process, only the ground-truth information of the object is used as an input for *LFANet* and only the ground-truth bounding box heatmap Υ is forwarded into the secondary heads. When evaluating the model with the ground-truth information as well we get the upper limit of performance for the *LFANet*, which is presented in Section 6.4.2.

6.4.1 Training of secondary heads

For the first part of training *CenterFusion* architecture with the *LFANet*, the secondary heads are trained using an adapted bounding box heatmap Υ. It contains ground-truth radar information of the depth d^{GT} of the object's center point and its projected radial velocity $v_{\mathrm{rad}}^{\mathrm{GT}} = \begin{bmatrix} v_{\mathrm{rad}_X}^{\mathrm{GT}} & v_{\mathrm{rad}_Z}^{\mathrm{GT}} \end{bmatrix}^{\mathrm{T}}$. The difference to the original *CenterFusion* pipeline is that it uses the information of the closest radar point r' in the frustum, which lies most of the times on the edges of the frustum. The heatmap Υ is created as described in Section 4.3, filled in with the ground-truth features. The altered model is trained in mode 2 with the parameters as in *CenterFusion*.

6.4.2 Proof of concept

To evaluate whether the optimized frustum association can indeed improve the network architecture *CenterFusion*, we analyze the network using ground-truth information. The study is based upon the training described in Section 6.4.1. The secondary heads are trained using ground-truth values from the annotations. The ground-truth heatmap Υ can therefore also be used to validate the approach. This can interpreted as measuring the theoretical limit if the *LFANet* would perform optimally. It also functions as a "proof of concept" that the *LFANet* can improve the architecture if trained perfectly. We assume that the optimal *LFANet* always outputs the exact depth of the center-point d^{GT} and radial velocity $v_{\mathrm{rad}}^{\mathrm{GT}}$ of any object, i.e. we assume the *LFANet* to perform with a loss of zero. Table 6.6 shows the result for the ground-truth evaluation of the training described in Section 6.4.1, termed as $\mathrm{CenterFusion}_{60}^{\mathrm{GT}}$.

All metrics except the scale error mASE show large improvements. The mAP is improved by 30.90%, the NDS by 18.94%. The largest improvement in the error metrics is observed in the mATE which decreases to 62.86% of the error in the baseline model. The ground-truth study shows that there is high potential to improve the *CenterFusion* network in the stage of the frustum association. We did not highlight the numbers because we are assuming unrealistic circumstances.

6.4.3 Training of LFANet

The training of *LFANet* builds up on the training of the secondary heads in the model $\mathrm{CenterFusion}_{60}^{\mathrm{GT}}$. For *LFANet*, both network architectures described in Section 5.1.3 are implemented and compared to each other. $LFANet_{IMG}$ and $LFANet_{PC}$ are trained standalone in mode 3.The input of the network in training are the ground-truth values for the object's dimension γ^{GT}, location $c_{\mathrm{3D}}^{\mathrm{GT}}$ and global rotation θ^{GT} as well as the entire point cloud. For validation, the predictions from the primary heads are forwarded instead. The loss L_{LFA} is computed on the regressed depth d^*, projected radial velocity $v_{\mathrm{rad}_X}^*$ in X- and velocity $v_{\mathrm{rad}_Z}^*$ in Z-direction. The input to the secondary heads are the feature map FM concatenated with the bounding box heatmap Υ, where the regressed features are filled in.

6.4.4 LFANet results

Table 6.6 compares the results of the *LFANet* training runs for both network architectures $LFANet_{IMG}$ and $LFANet_{PC}$ as well as the "proof of concept" model $\text{CenterFusion}_{60}^{\text{GT}}$ with the baseline model CenterFusion_{60}. All models were trained multiple times to minimize the effect of numerical fluctuations on the training results. With a mAP value of 0.3343, the *IMG* architecture outperforms the *PC* architecture of *LFANet* by 6.98% which allows the conclusion, that this architecture works better in our use-case. In consequence, we only use the *IMG*-based architecture in the following.

Even more important however, the model $\text{LFANet}_{\text{IMG},60}$ is able to outperform the baseline CenterFusion_{60} with an increase in the mAP metric by 3.73% from 0.3223 to 0.3343. Additionally, the obtained mAP value of 0.3343 is even slightly higher than the score obtained through the validation of the model **CenterFusion$_{60}$** trained by the authors of [40]. However, since we were not able to reproduce these results with the training procedure provided, we take our own training CenterFusion_{60} as a baseline for the performance improvements obtained through the modifications of the network architecture.

However, the results of the $\text{LFANet}_{\text{IMG},60}$ also show a decreased performance in the orientation and velocity errors mAOE and mAVE compared to both, the CenterFusion_{60} baseline and the $\text{LFANet}_{\text{PC},60}$ model. The reason behind this phenomenon is not completely investigated yet and part of future work. With an increased mAP value, the network detects more objects than the baseline CenterFusion_{60}. Some of these objects might contain sparse or inaccurate information from the radar point cloud. While the new network architecture might be able to successfully locate these objects with improved precision, the imprecise information could lead to an overall decreased average precision in some of the errors metrics. To further evaluate this, one could calculate and compare the error metrics for the objects detected by both architectures. Furthermore, we expect the errors especially mAVE to decrease greatly when the backbone is unfrozen in training.

Table 6.6: Comparison of CenterFusion_{60} and the entire *LFANet* training of both versions with the models $\text{LFANet}_{\text{IMG},60}$ and $\text{LFANet}_{\text{PC},60}$. All models are evaluated the validation split.

			Error ↓				
Model	mAP ↑	NDS ↑	mATE	mASE	mAOE	mAVE	mAAE
CenterFusion_{60}	0.3223	**0.4502**	0.6588	0.2623	**0.5008**	**0.5403**	0.1475
$\text{LFANet}_{\text{IMG},60}$	**0.3343**	0.4392	**0.6539**	0.2621	0.5584	0.6603	**0.1450**
$\text{LFANet}_{\text{PC},60}$	0.3125	0.4307	0.6933	**0.2617**	0.5074	0.6406	0.1522
$\text{CenterFusion}_{60}^{\text{GT}}$	0.4219	0.5355	0.4072	0.2708	0.5067	0.4304	0.1393

Figure 6.1 shows six output examples of the $\text{LFANet}_{\text{IMG}}$ model from the validation split of the dataset. While most examples (see Figure 6.1a to Figure 6.1d) show very promising results, some predictions are not as precise, see Figure 6.1e and Figure 6.1f. Figure 6.1a to Figure 6.1d show examples of the output where both, velocity and depth information of LFANet match almost perfectly with the ground

truth values in different scenarios and for different object sizes. In Figure 6.1e the velocity estimation is not correct although the radar points contain correct information, in Figure 6.1f the depth prediction for a small object is off by about 1.5m. Most of the bad examples only contain a single radar point within the frustum. This makes it hard for the network to predict a precise output in general. In the future, a radar point cloud of higher density could improve these scenarios. Figure 6.1e

6.5 Early Fusion

As described in Section 5.2, the focus on the implementation of *Early Fusion* in *CenterFusion++* lays on improving the robustness of the network, especially w.r.t. challenging environmental conditions. We therefore introduce a new validation split including scenes that are captured in rain or during the night only, termed as *rain-night-val*. The split contains 39 scenes and 9 420 camera images, none of which have been used in the training process. Additionally, we introduce a validation split termed *night-val* that only contains the 602 camera images and their corresponding radar data recorded during night.

We train the *Early Fusion* approach from "scratch" and compare it to our own *CenterNet* training runs presented in Section 6.2. To train from scratch here corresponds to a training starting from the weights obtained by training the DLA-34 network for 120 epochs on the ImageNet dataset [85]. From the secondary heads, only the velocity and attribute heads are trained to compare their metrics with the velocity and attribute estimations based on *CenterNet*. The authors of [40] therefore trained the mentioned heads for 30 epochs on top of the **CenterNet$_{140}$** baseline, described in Section 6.2, to obtain **CenterNet$_{170}$**. We performed the equivalent training on our own CenterNet$_{140}$ baseline resulting in the model CenterNet$_{170}$.

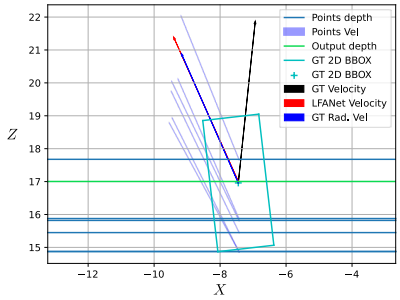
Alongside the standard *Early Fusion* training, termed as EarlyFusion$_{140}$, another run is trained with a *BlackIn* rate of $p_{\text{BI}} = 20\%$, see Section 5.2, we refer to this training as EarlyFusionBI$_{175}$.

Table 6.7: Results of the Early Fusion training runs, evaluated on the *val* split. The metrics mAOE and mAVE were evaluated using the secondary heads with a *BlackIn* rate of $p_{BI} = 0\%$.

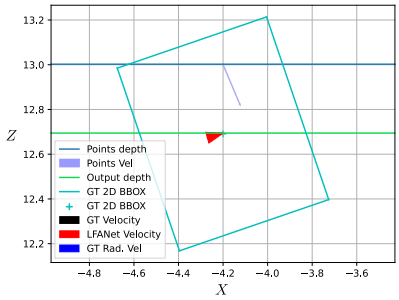
			Error ↓				
Model	mAP ↑	NDS ↑	mATE	mASE	mAOE	mAVE	mAAE
CenterNet$_{170}$	0.3044	0.3373	0.7133	**0.2582**	0.5170	1.3869	0.6603
EarlyFusion$_{140}$	**0.3074**	**0.4446**	**0.7056**	0.2615	**0.4779**	0.5016	**0.1446**
EarlyFusionBI$_{175}$	0.2914	0.4300	0.7216	0.2649	0.5352	**0.4875**	0.1474

The results for the standard *Early Fusion* training run EarlyFusion$_{140}$ are similar to the results obtained from the CenterNet$_{170}$ model in every metric except the orientation, velocity, and attribute errors mAOE, mAVE, and mAAE.

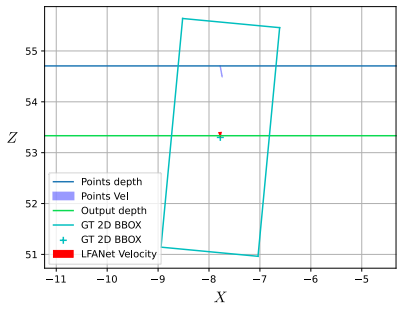
The model EarlyFusionBI$_{140}$ shows a slightly improved mAP value compared to the other two networks. The difference to CenterNet$_{170}$ can be explained by numerical fluctuations in the training progress but also shows that the introduction



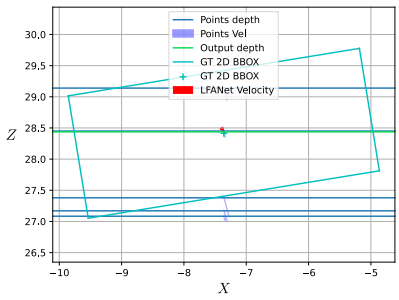
(a) Good prediction, multiple radar points.



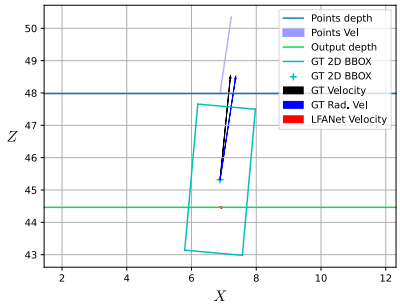
(b) Good prediction, single radar point, small object.



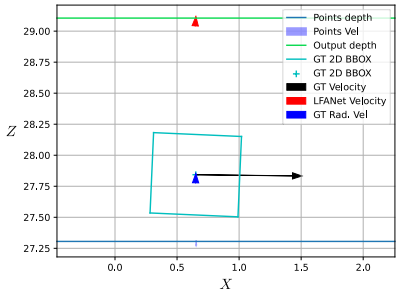
(c) Good prediction, single radar point, big object.



(d) Good prediction, multiple radar points, big, rotated object.



(e) Bad velocity prediction, single radar point, big object.



(f) Bad depth prediction, single radar point, small object.

Figure 6.1: Examples for the output of the $\text{LFANet}_{\text{IMG}}$ model in BEV. The turquoise rectangle depicts the ground truth BB of the object in BEV, the turquoise "+" its center point. The black arrow starting at the center of the BB is the ground truth velocity of the object, the dark blue arrow the radial component of the ground truth velocity towards the camera. The light blue arrows correspond to the velocities measures for each individual point in the radar point cloud. They start at the depth of the corresponding radar point and the center of the BB in X. The red arrow is the velocity prediction as the output of LFANet. The blue, horizontal lines depict the depth information of a single radar point. It it depicted by a line since the LFANet architecture does not depend on the X component of a radar point. The green, horizontal line is the depth estimation predicted by LFANet.

of *Early Fusion* does not lead to any performance decreases. However, the mAP shows slightly worse results for the EarlyFusion training with a *BlackIn* rate of $p_{\text{BI}} = 20\%$ trying to force the network towards the radar data. Since 20% of all images are not available as an input to the network during training, the network is trained for more epochs to allow it to learn from the same amount of input images. Assuming a linear dependency, the training is expanded to 175 epochs to comply with the $175 \cdot \frac{8}{10} = 140$ epochs used in the training runs of *CenterNet*. However, the strict focus on the sparser radar data still leads to a decreased performance in the mAP metric. In general, the EarlyFusion$_{140}$ leads to big improvements in the velocity and attribute errors compared to the CenterNet$_{170}$ model. While the orientation error mAOE is improved by 7.56%, the velocity error mAVE is improved by 63.83% and the attribute error by 78.1$. This implies that the *Early Fusion* actually profits from the radar data introduced through projection in an early step. The improvements also show in a bigger NDS value.

Robustness

To evaluate the robustness improvements gained through the additional radar channels in the input image of *CenterNet*, two validation scenarios are regarded. First, we evaluate the performance on the *night-val* split to investigate potential improvements in difficult environmental conditions. Second, we test the robustness towards sensor failure by introducing artificial camera *BlackIns* with a rate of $p_{\text{BI}} = 50\%$. In the following, the EarlyFusionBI$_{175}$ model is used for all validations. We therefore have to account for the slightly worse, overall performance of the model, especially in mAP, when regarding different splits of the validation set.

Table 6.8 shows the results for the first test case. The attribute and velocity heads are used in the *BlackIn* training for this comparison.

Table 6.8: Comparison of CenterNet$_{170}$ and EarlyFusionBI$_{175}$ evaluated on the *night-rain-val* split with a *BlackIn* rate of $p_{\text{BI}} = 0\%$.

			Error ↓				
Model	mAP↑	NDS↑	mATE	mASE	mAOE	mAVE	mAAE
CenterNet$_{170}$	**0.2784**	0.3104	**0.7677**	**0.2748**	**0.5512**	1.3737	0.6945
EarlyFusionBI$_{175}$	0.2629	**0.3997**	0.7803	0.2791	0.6474	**0.5008**	**0.1098**

Table 6.9: Comparison of CenterNet$_{170}$ and EarlyFusionBI$_{175}$ evaluated on the *night-val* split with a *BlackIn* rate of $p_{\text{BI}} = 0\%$ regarding the car category only.

			Error ↓				
Model	mAP↑	NDS↑	mATE	mASE	mAOE	mAVE	mAAE
CenterNet$_{170}$	0.159	0.375	**0.463**	0.116	**0.05**	3.395	0.418
EarlyFusionBI$_{175}$	**0.168**	**0.440**	0.491	**0.111**	0.06	**0.623**	**0.152**

The mAP of EarlyFusionBI$_{175}$ is around 5% smaller than the mAP of CenterFusion$_{170}$ on the complete *val* split of the dataset. Evaluated on the *night-rain-val* split

of the dataset, the relative difference is almost the same around 5%. However, when evaluated on the night scenes only, the difference reduces to $\approx 3\%$ which implies that EarlyFusionBI$_{175}$ can handle night scenes slightly better. This also shows in some examples from the dataset, see Figure 6.2.

Additionally, when regarding the car category only, there is an improvement of the mAP metric through the introduction of *Early Fusion*, see Table 6.9.

Table 6.10: Comparison of CenterNet$_{170}$ and EarlyFusionBI$_{175}$ evaluated on the *night-val* split with a *BlackIn* rate of $p_{\text{BI}} = 0\%$.

			Error ↓				
Model	mAP↑	NDS↑	mATE	mASE	mAOE	mAVE	mAAE
CenterNet$_{170}$	**0.0646**	0.1418	0.9087	**0.5155**	0.6995	2.4809	0.7812
EarlyFusionBI$_{175}$	0.0624	**0.1669**	**0.8608**	0.5252	**0.6560**	**1.0082**	**0.6008**

(a) Detection using *CenterNet* **(b)** Detection using EarlyFusion$_{140}$

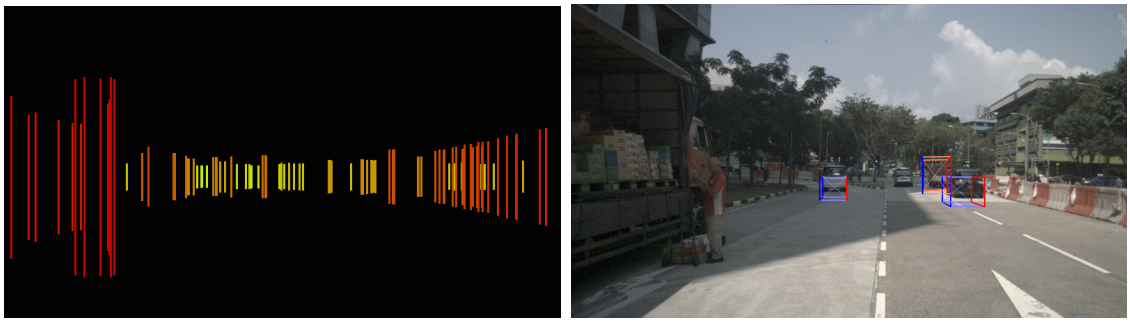
Figure 6.2: Example of the improvements obtained through *Early Fusion*. Figure 6.2a shows a car in a night scene that is not detected by the *CenterNet* network based on camera only. However, the same input image along with the corresponding radar data results in a detection using *Early Fusion*, see Figure 6.2b.

Table 6.11 compares *CenterNet* and *EF BlackIn* with a *BlackIn* rate of $p_{\text{BI}} = 50\%$ in order to simulate temporary sensor failure.

Table 6.11: Comparison of CenterNet$_{170}$ and EarlyFusionBI$_{175}$ evaluated on the *val* split with a *BlackIn* rate of $p_{\text{BI}} = 50\%$.

			Error ↓				
Model	mAP↑	NDS↑	mATE	mASE	mAOE	mAVE	mAAE
CenterNet$_{170}$	0.1415	0.2054	**0.6866**	**0.4547**	0.7041	1.3868	0.8076
EarlyFusionBI$_{175}$	**0.1529**	**0.2951**	0.7443	0.4786	**0.6320**	**0.6730**	**0.2849**

In contrast to the evaluations performed above, the mAP for the EarlyFusionBI$_{175}$ model is bigger than the mAP for the CenterNet$_{170}$ by around 7.4% for the validations with a *BlackIn* rate of $p_{\text{BI}} = 50\%$. This demonstrates the greater robustness of the *Early Fusion* approach regarding sensor failure. In addition, EarlyFusionBI$_{175}$ shows better detection results in some specific examples from the dataset, see Figure 6.3.



(a) Radar depth feature channel. **(b)** Detection using EarlyFusionBI$_{175}$

Figure 6.3: Example of the improvements obtained through *EF BlackIn*. Figure 6.3b shows the detections of the EarlyFusionBI$_{175}$ model with a zero-valued input image. The regular CenterNet$_{140}$ network did not output any detections in this scenario. However, due to the corresponding radar data, displayed in Figure 6.3a, the *EarlyFusion* network handles the simulated camera failure better and still detects objects.

6.6 CenterFusion++

Section 6.5 and Section 6.4 reflect the results of the two main improvements within *CenterFusion++* and justify their use. This section compares the CenterFusion$_{60}$ training, explained in Section 6.3 with a new model termed CenterFusion++$_{60}$.

It is based upon the EarlyFusion$_{140}$ model described in Section 6.5 to obtain more robustness of the object detection in the primary heads. The further training process matches with the one described in Section 6.4, where the **CenterNet$_{170}$** model is replaced by the model EarlyFusion$_{140}$. Table 6.12 shows the comparison of the obtained model to CenterFusion$_{60}$. Both models match each other in performance, the differences are almost neglectable. We were not able to improve the results in a similar manner as the improvements obtained through *LFANet*. One reason might be the EarlyFusion$_{140}$ that is outperformed by the pre-trained **CenterNet**$_{170}$. As described in Section 6.2, we were not able to reproduce the results presented in [56] completely although using the described training procedure. However, we were able to reach the level of performance of CenterFusion$_{60}$ (based on **CenterNet$_{170}$**) with CenterFusion++$_{60}$ suggesting that the introduction of the two main adaptions indeed does improve the network architecture.

Additionally, for further improvements on the robustness of the network, a CenterFusionBI++$_{60}$ model based on the EarlyFusionBI$_{175}$ model was trained, the results are displayed in Table 6.12 too. The worse value in the mAP was to be expected due to the results described in the previous sections.

Run-time analysis

To investigate whether the implemented changes have a large impact on the runtime of the detection pipeline, a run-time analysis was performed. Table 6.13 shows the run-time for three models introduced in this work: CenterFusion$_{60}$ as introduced by Nabati et al. [40], LFANet$_{\text{IMG}}$ trained on top of the CenterNet$_{170}$ architecture, and

Table 6.12: Comparison of the evaluation on the *val* split between the unadapted CenterFusion$_{60}$ architecture, introduced in [40] and the adapted network CenterFusion++$_{60}$.

			Error ↓				
Model	mAP↑	NDS↑	mATE	mASE	mAOE	mAVE	mAAE
CenterFusion$_{60}$	**0.3223**	0.4502	**0.6588**	**0.2623**	0.5008	0.5403	0.1475
CenterFusion++$_{60}$	0.3209	**0.4512**	0.6756	0.2629	**0.4758**	**0.5359**	**0.1423**
CenterFusionBI++$_{60}$	0.3031	0.4373	0.6869	0.2644	0.5070	0.5380	0.1461

CenterFusion++$_{60}$ uniting the two main changes to the network: Early Fusion and LFANet.

Table 6.13: Comparison of the run-times for three network architectures. All evaluations were performed on the *local* configuration described in the introduction of this chapter. The third column expresses the increase in runtime relative to the CenterFusion$_{60}$ model.

Network Architecture	Runtime [ms]	Runtime increase
CenterFusion$_{60}$	244	–
CenterFusion + LFANet$_{\text{IMG}}$	254	3.94%
CenterFusion++	274	10.95%

Introducing LFANet to the network results in an run-time increase of less than 4% which is perfectly reasonable for the performance improvements gained. Additionally, introducing EarlyFusion to the network results in a total run-time increase of about 11%. It is likely that further improvements on the implementation can be made since this was not the main focus of the thesis work.

6.7 Ablation Studies

We experimented a lot with different setups of *CenterFusion++* where we for example trained with different hyperparameters for the shrinking factor η in the bounding box heatmap Υ or the resolution of the snapshot κ. These experiments resulted in worse performances than the ones shown above, although for completeness we provide them in this section. Smaller ablation studies include deviations in the relaxation factor τ_d of the frustum in both validation and training, the features of the snapshot $\mathcal{X}$, the pillar size $\overline{w}$ in the snapshot, the number of filters n_f of the *LFANet* and dilation factor in *LFANet* $_{PC}$. More complex modifications are explained in the following.

RCS input layer

As mentioned in Chapter 5, the authors of *CenterFusion* did not use the RCS measurements provided by the radar sensor in the input to the secondary heads. To

analyze the effects of doing so, we introduced an additional input channel containing the RCS value of the associated radar point in the heat-map input for the secondary heads. The *Bounding box heatmap* in Figure 4.10 therefore is appended by one channel, see Figure 6.4, the rest of the secondary heads does not change.

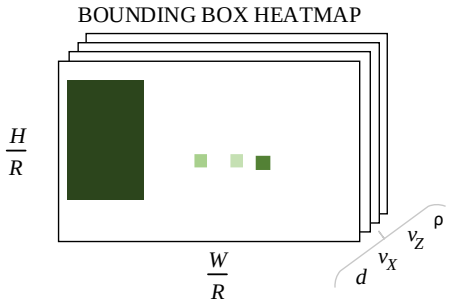


Figure 6.4: Bounding box heatmap as explained in Figure 4.10 extended by the additional radar feature RCS depicted by ρ.

Table 6.14 compares the metrics of the CenterFusion$_{60}$ training with the training including the *RCS* input layer, termed CenterFusionRCS$_{60}$.

Table 6.14: Comparison of CenterFusion$_{60}$ and CenterFusionRCS$_{60}$ using the radar's *RCS* feature as an additional input feature.

			Error ↓				
Model	mAP↑	NDS↑	mATE	mASE	mAOE	mAVE	mAAE
CenterFusion$_{60}$	0.3223	0.4502	0.6588	0.2623	**0.5008**	**0.5403**	0.1475
CenterFusionRCS$_{60}$	**0.3232**	0.4438	**0.6548**	**0.2604**	0.5453	0.5723	**0.1452**

The results are very similar to each other. There are negligible differences in mAP and the additional error metrics. This leads to the conclusion that the RCS measurement can not lead to noticeable improvements in the architecture. This also justifies the decision to not use the RCS value in the outputs of *LFANet*. However, the RCS might still be useful for the construction of the artificial radar point r^* since the RCS value might allow to detect outliers in a set of points within the ROI.

Frustum calculation

CenterFusion proposes to filter the point cloud by the depth bounds of the object and thus creating a frustum that is limited in the Z-coordinate. We experimented with changing that definition to using the distance from the origin to the corners of each object instead. Therefore, the frustum is limited by the distance and only points within these distance bounds are associated to the object. The results of training *CenterFusion* with the modified frustum method are given in Table 6.15.

The values for the two association methods only differ slightly. Since only the closest radar point is processed in *CenterFusion*, the selection method only makes

Table 6.15: Comparison of the CenterFusion$_{60}$ model using the regular, depth-wise frustum association and CenterFusion$_{60,\ \mathrm{dist}}$ using the distance frustum association with a frustum expansion ratio of $\delta = 0.5$.

			Error ↓				
Model	mAP↑	NDS↑	mATE	mASE	mAOE	mAVE	mAAE
CenterFusion$_{60}$	**0.3223**	**0.4502**	**0.6588**	0.2623	**0.5008**	**0.5403**	0.1475
CenterFusion$_{60,\ \mathrm{dist}}$	0.3184	0.4364	0.6751	**0.2611**	0.5595	0.5897	**0.1427**

a difference in few cases. Therefore, the association was not changed from depth to distance in this work.

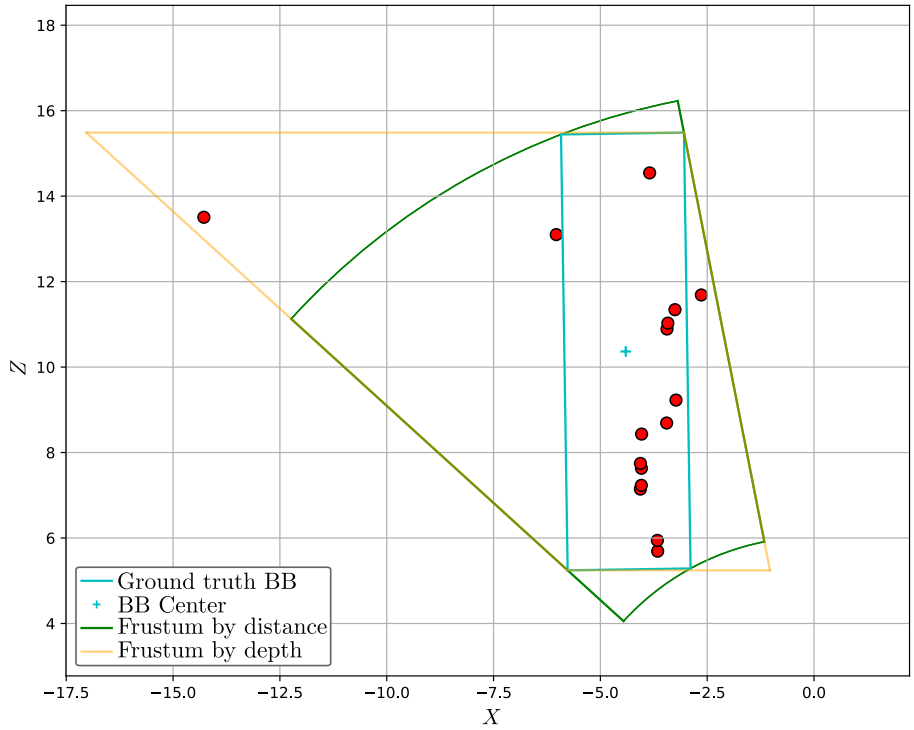


Figure 6.5: Comparison of the frustums created by the depth and distance method.

Radar sweeps

The sparsity of the radar point cloud in the *nuScenes* dataset motivates the use of more radar sweeps and therefore more radar data. As described in Section 3.2, usually 3 radar sweeps are aggregated as a trade-off between point cloud sparsity and suppression of errors introduced by external vehicle's motion.

Table 6.16 compares the results of the regular EarlyFusion$_{140}$ training with 3 radar sweeps to a training run aggregating the data of 6 radar sweeps, termed EarlyFusion$_{140,6S}$. Since this inevitable introduces a bigger error, an additional input channel containing the time difference of the radar sweep to the camera image's capturing is concatenated to the input channels. However, introducing 6 radar time

sweeps instead of 3 does not lead to significant improvements in the performance metrics.

Table 6.16: Results of the Early Fusion training runs, evaluated on the *val* split. The metrics mAOE and mAVE were evaluated without the use of the secondary heads for better comparability.

			Error ↓				
Model	mAP ↑	NDS ↑	mATE	mASE	mAOE	mAVE	mAAE
EarlyFusion_{140}	**0.3074**	**0.4445**	**0.7063**	**0.2617**	**0.4781**	0.5017	**0.1446**
$\text{EarlyFusion}_{140,6S}$	0.3029	0.4425	0.7149	0.2620	0.4856	**0.4774**	0.1494

Adapted head architecture

As described in Section 4.4, the authors of *CenterFusion* define the head architecture of the secondary heads to consist of three consecutive 3×3 convolutional layers followed by a single 1×1 layer. However, the model the authors achieved the benchmark performance with use a single 3×3 kernel and three consecutive 1×1 layers instead. To evaluate the performance differences, the 3×3 layers are introduced to both, the primary and secondary heads. The resulting architecture is trained analogously to CenterFusion_{60} and termed $\text{CenterFusion}_{60,AH}$. Table 6.17 shows the differences in the mentioned training runs.

Table 6.17: Results of the regular CenterFusion_{60} run with three 1×1 in the heads compared to the adapted head architecture $\text{CenterFusion}_{140,AH}$ with three consecutive 3×3 heads followed by a single 1×1 kernel.

			Error ↓				
Model	mAP ↑	NDS ↑	mATE	mASE	mAOE	mAVE	mAAE
CenterFusion_{60}	**0.3223**	**0.4502**	**0.6588**	0.2623	**0.5008**	**0.5403**	**0.1475**
$\text{CenterFusion}_{60,AH}$	0.3130	0.4367	0.6883	**0.2609**	0.5275	0.5696	0.1514

Since their is no improvement of the metrics using the adapted head architecture, it is not used in this work, instead the head architecture used in [40] is implemented.

7
Conclusion

To guarantee the safety and reliability of autonomous vehicles, the perception of the environment is of crucial importance. Our work discusses the current state of deep-learning based sensor fusion regarding camera and radar. Furthermore, we suggest multiple improvements to one state-of-the-art fusion architecture. Our work is a fusion architecture that implements early fusion as well as extracts features from proposals in an subnetwork called "Learned Frustum Association Network".

Section 7.1 discusses the results presented in Chapter 6 and identifies the potential as well as problems of our contributions. Potential solutions to these problems are presented in Section 7.2.

7.1 Discussion

After an extensive literature research, the network *CenterFusion* was chosen to be the base of our work since it presents a novel approach to the fusion of camera and radar in the environment of autonomous driving together with promising results on the *nuScenes* dataset.

After improving the implementation of the rotation loss, see Section 6.2, we were able to reproduce the results presented in [40] to a reasonable extend. Further, two main issues with the network were examined. First, the architecture is not robust w.r.t. camera failure or bad weather conditions since it heavily depends on the camera-based detections in the primary heads. Second, the association of the radar point within the detected ROI results in information loss since only the closest point towards the camera is selected for the fusion to the camera features. We were able to support this claim by analyzing the effectiveness of choosing the closest point when evaluating over the *nuScenes* dataset in Section 6.4.

To tackle the lack of robustness, *Early Fusion* was introduced by projecting the radar data into the image plane and concatenating it to the image input. During training, we used *BlackIn* to enforce the network towards prioritizing radar data in order to make it less dependent of the camera input. The introduced measures result in slightly improved robustness properties, especially in night scenes and when simulating camera sensor failure. However, we also experienced a small reduction in the mAP metric when comparing the training run $\text{EarlyFusionBI}_{140}$ involving *BlackIn* with CenterNet_{140}. Since we were able to reproduce the CenterNet_{140} results using the EarlyFusion_{140} model without *BlackIn*, it is likely that the robustness-measure *BlackIn* also creates the need for a longer training run or different training strategies to compensate for the reduced amount of non-zero input images.

The second improvement was the introduction of *LFANet* which extracts an artificial radar point r^* instead of choosing the closest point as in *CenterFusion*. We were able to reach and even slightly outperform the results obtained by the authors of *CenterFusion* [40]. This result shows that our approach is reasonable and we are able to extract more of the data given in the radar point cloud. However, there is still room for improvement, especially regarding objects that only contain a single radar point. The high sparsity of the radar point cloud in the *nuScenes* dataset increases the difficulty for *LFANet* to learn from the data, since often only a few or even a single radar point can be associated to an object, see Figure 3.4. Further, we have shown the pipeline can be improved a lot through the radar association by evaluating the whole pipeline with ground-truth information into the secondary heads which leads to an increase of 44.52% in mAP. Using different network architectures and training procedures for *LFANet* might make use of more of the potential of the approach.

7.2 Outlook

The outlook is split into two parts: first we investigate potential solutions to the problems described in Section 7.1 before we give an outlook on further research and alternative approaches for fusion of radar and camera data using deep machine learning methods.

Improvements on CenterFusion++

The difference in the mAP metric for the *Early Fusion* approach can be resolved using training runs comprising a bigger number of epochs. To avoid overfitting in this process, learning rates and hyperparameter studies have to be performed.

Additionally, the authors of [42] fuse the radar data at different stages of the image processing architecture. Since the architecture of the backbone DLA used in this work is not restricted to one specific setup, this is definitely feasible in the current architecture. The fusion on multiple network levels has potential to result in slightly better scores while also increasing the networks processing time as a downside. Furthermore, the fusion step is not optimized since the radar data is concatenated directly to the image-based feature map FM and not raw data. This could be improved for example by adding an additional subnetwork that processes the radar data and which output is concatenated to the feature map FM. Currently the backbone architecture used is optimized for image processing but since we use a fused input of images and point cloud data the architecture chosen for the backbone could be revised to handle point cloud data better.

To further improve the results of *LFANet*, the network design can be revised and the training process can be optimized in areas such as the use of other loss functions – L2 or smooth-L1 – and tuning of learning rate and other hyperparameters. It might especially be worth investigating fully connected layers instead of a convolutional layers as in *PointNet++* [61] for extracting features from a point cloud. Additionally, assuming the availability of a dataset including a high-resolution radar and camera

images, the investigation of the current network architecture's performance is of high interest since it is very sensitive to the resolution of the radar.

Further research

In current research regarding camera radar sensor fusion based on deep machine learning one bottleneck is the availability of public datasets. While *nuScenes* is by far the biggest and most diverse dataset in the field, it lacks radar point clouds in high resolution. With the availability of datasets solving this issue in the future, other fusion architectures currently used within the fusion of LiDAR and camera will become increasingly interesting. For example, the *Fusion Painting* approach [43] and others approaches heavily depending on a high density point cloud [61, 87] might be worth investigating.

An alternative to radar data in point clouds is the processing of radar data in its unprocessed form which looks promising in initial research papers [29, 88]. However, there currently is no dataset of sufficient size and scene diversity that includes both, unprocessed radar and camera data.

Recent machine learning networks for 3D object detection based on camera only use temporal information, i.e. multiple camera images from past time stamps. This does not only increase the detection accuracy but also allows the network to predict lateral velocities of objects in the images. Together with the radial velocities provided by the radar, this would allow for a better estimation of velocity in the network.

Bibliography

[1] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 618–11 628.

[2] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner, "Three decades of driver assistance systems: Review and future perspectives," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 4, pp. 6–22, 2014.

[3] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, "Deep learning sensor fusion for autonomous vehicle perception and localization: A review," *Sensors*, vol. 20, no. 15, p. 4220, jul 2020.

[4] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Gläser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2021.

[5] F. de Ponte Müller, "Survey on ranging sensors and cooperative techniques for relative positioning of vehicles," *Sensors*, vol. 17, no. 2, p. 271, 2017.

[6] S. Nyholm and J. Smids, "The ethics of accident-algorithms for self-driving cars: an applied trolley problem?" *Ethical Theory and Moral Practice*, vol. 19, no. 5, pp. 1275–1289, jul 2016.

[7] T. Holstein, G. Dodig-Crnkovic, and P. Pelliccione, "Ethical and social aspects of self-driving cars," 2018.

[8] S. Nyholm, "The ethics of crashes with self-driving cars: A roadmap, i," *Philosophy Compass*, vol. 13, no. 7, p. e12507, may 2018.

[9] J. Borenstein, J. R. Herkert, and K. W. Miller, "Self-driving cars and engineering ethics: The need for a system level analysis," *Science and Engineering Ethics*, vol. 25, no. 2, pp. 383–398, nov 2017.

[10] J. Borenstein, J. Herkert, and K. Miller, "Self-driving cars: Ethical responsibilities of design engineers," *IEEE Technology and Society Magazine*, vol. 36, no. 2, pp. 67–75, jun 2017.

[11] M. Coeckelbergh, "Responsibility and the moral phenomenology of using self-driving cars," *Applied Artificial Intelligence*, vol. 30, no. 8, pp. 748–757, sep 2016.

[12] S. Karnouskos, "Self-driving car acceptance and the role of ethics," *IEEE Transactions on Engineering Management*, vol. 67, no. 2, pp. 252–265, may 2020.

[13] J. McCarthy, "An analysis of carbon emissions between autonomous vehicles and conventional modes of transportation. master's thesis, harvard extension school." 2017.

[14] M. Massar, I. Reza, S. M. Rahman, S. M. H. Abdullah, A. Jamal, and F. S. Al-Ismail, "Impacts of autonomous vehicles on greenhouse gas emissions—positive or negative?" *International Journal of Environmental Research and Public Health*, vol. 18, no. 11, p. 5567, may 2021.

[15] L. Mora, X. Wu, and A. Panori, "Mind the gap: Developments in autonomous driving research and the sustainability challenge," *Journal of Cleaner Production*, vol. 275, p. 124087, dec 2020.

[16] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[17] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[21] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 764–773.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[23] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[24] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[25] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[27] T. Moeller, A. Padhi, D. Pinner, and A. Tschiesner, "Reserve a seat - the future of mobility is arriving early." McKinsey&Company, 2018. [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/reserve-a-seat-the-future-of-mobility-is-arriving-early

[28] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, p. 2140, mar 2021.

[29] A. Ouaknine, A. Newson, J. Rebut, F. Tupin, and P. Pérez, "Carrada dataset: Camera and automotive radar with range- angle- doppler annotations," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 5068–5075.

[30] R. Szeliski, *Computer Vision*. Springer International Publishing, 2022.

[31] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, mar 2004.

[32] D. Y. Kim and M. Jeon, "Data fusion of radar and image measurements for multi-object tracking via kalman filtering," *Information Sciences*, vol. 278, pp. 641–652, sep 2014.

[33] X. Dong, B. Zhuang, Y. Mao, and L. Liu, "Radar camera fusion via representation learning in autonomous driving," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 1672–1681.

[34] X.-p. Guo, J.-s. Du, J. Gao, and W. Wang, "Pedestrian detection based on fusion of millimeter wave radar and vision," in *Proceedings of the 2018 International Conference on Artificial Intelligence and Pattern Recognition*, ser. AIPR 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 38–42.

[35] T.-Y. Lim, A. Ansari, B. Major, D. Fontijne, M. Hamilton, R. Gowaikar, and S. Subramanian, "Radar and camera early fusion for vehicle detection in advanced driver assistance systems," *NeurIPS Machine Learning for Autonomous Driving Workshop*, 2019.

[36] F. Nobis, E. Shafiei, P. Karle, J. Betz, and M. Lienkamp, "Radar voxel fusion for 3d object detection," *Applied Sciences*, vol. 11, no. 12, p. 5598, jun 2021.

[37] X. Wang, L. Xu, H. Sun, J. Xin, and N. Zheng, "On-road vehicle detection and tracking using mmw radar and monovision fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 7, pp. 2075–2084, 2016.

[38] Z. Zhong, S. Liu, M. Mathew, and A. Dubey, "Camera radar fusion for increased reliability in adas applications," *Proc. IS T Int'l. Symp*, pp. 258–1 – 258–4, 2018.

[39] V. John and S. Mita, "Rvnet: Deep sensor fusion of monocular camera and radar for image-based obstacle detection in challenging environments," in *Image and Video Technology*, C. Lee, Z. Su, and A. Sugimoto, Eds. Cham: Springer International Publishing, 2019, pp. 351–364.

[40] R. Nabati and H. Qi, "Centerfusion: Center-based radar and camera fusion for 3d object detection," in *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2021, pp. 1526–1535.

[41] F. C. P. Kevin M. Lynch, *Modern Robotics: Mechanics, Planning, and Control*. CAMBRIDGE, May 2017. [Online]. Available: https://www.ebook.de/de/product/29172931/kevin_m_lynch_frank_c_park_modern_robotics_mechanics_planning_and_control.html

[42] F. Nobis, M. Geisslinger, M. Weber, J. Betz, and M. Lienkamp, "A deep learning-based radar and camera sensor fusion architecture for object detection," in *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, 2019, pp. 1–7.

[43] S. Xu, D. Zhou, J. Fang, J. Yin, Z. Bin, and L. Zhang, "Fusionpainting: Multimodal fusion with adaptive attention for 3d object detection," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021, pp. 3047–3054.

[44] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[45] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov, "Large scale interactive motion forecasting for autonomous driving : The waymo open motion dataset," 2021.

[46] D. Barnes, M. Gadd, P. Murcutt, P. Newman, and I. Posner, "The oxford radar robotcar dataset: A radar extension to the oxford robotcar dataset," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 6433–6438.

[47] Y. Wang, Z. Jiang, X. Gao, J.-N. Hwang, G. Xing, and H. Liu, "Rodnet: Radar object detection using cross-modal supervision," in *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2021, pp. 504–513.

[48] M. Mostajabi, C. M. Wang, D. Ranjan, and G. Hsyu, "High resolution radar dataset for semi-supervised learning of dynamic objects," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, jun 2020.

[49] M. Sheeny, E. De Pellegrin, S. Mukherjee, A. Ahrabian, S. Wang, and A. Wallace, "Radiate: A radar dataset for automotive perception in bad weather," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 1–7.

[50] (2020) nuScenes object detection task. Accessed: 2022/04/24. [Online]. Available: https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Camera%2C%20Radar

[51] (2022) vecteezy.com renault captur blueprint vector. Accessed: 2022/05/07. [Online]. Available: https://www.vecteezy.com/vector-art/640040-renault-captur-blueprint

[52] (2020) nuScenes overview. Accessed: 2022/04/24. [Online]. Available: https://www.nuscenes.org/nuscenes#overview

[53] (2020) ARS 408-21 long range radar sensor 77 GHz - data sheet. Accessed: 2022/05/06. [Online]. Available: https://conti-engineering.com/wp-content/uploads/2020/02/ARS-408-21_EN_HS-1.pdf

[54] (2022) Basler acA1600-60gc specifications. Accessed: 2022/05/06. [Online]. Available: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca1600-60gc/

[55] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, jul 2019.

[56] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," 2019.

[57] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 658–666.

[58] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[59] Z. Wei, F. Zhang, S. Chang, Y. Liu, H. Wu, and Z. Feng, "Mmwave radar and vision fusion for object detection for autonomous driving: A review," *CoRR*, vol. abs/2108.03004, 2021.

[60] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, sep 2009.

[61] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[62] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[63] R. B. Girshick, "Fast r-cnn," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, 2015.

[64] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[65] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[66] A. Rosenfeld and M. Thurston, "Edge and curve detection for visual scene analysis," *IEEE Transactions on Computers*, vol. C-20, no. 5, pp. 562–569, 1971.

[67] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7291–7299.

[68] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *European conference on computer vision*. Springer, 2016, pp. 483–499.

[69] G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, and K. Murphy, "Towards accurate multi-person pose estimation in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4903–4911.

[70] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 734–750.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[72] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 466–481.

[73] F. Yu, D. Wang, E. Shelhamer, and T. Darrell, "Deep layer aggregation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2403–2412.

[74] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Advances in neural information processing systems*, vol. 27, 2014.

[75] Z. Liu, Z. Wu, and R. Tóth, "Smoke: Single-stage monocular 3d object detection via keypoint estimation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 4289–4298.

[76] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 7074–7082.

[77] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456.

[78] M. Geisslinger, "Autonomous driving: Object detection using neural networks for radar and camera sensor fusion," Master's thesis, Technische Universität München, 2019.

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[80] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[81] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.

[82] (2020) NVIDIA DGX Station A100 data sheet. Accessed: 2022/05/20. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-station/nvidia-dgx-station-a100-datasheet.pdf

[83] (2021) NVIDIA A100 Tensor Core GPU data sheet. Accessed: 2022/05/20. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf

[84] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[85] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[86] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[87] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10 526–10 535.

[88] A. Ouaknine, A. Newson, P. Pérez, F. Tupin, and J. Rebut, "Multi-view radar semantic segmentation," Mar. 2021.

A
Appendix

A.1 *nuScenes* information

Table A.1: Overview over categories within the *nuScenes* dataset. The first column displays all categories that are assigned to the annotations without preprocessing. The second column names the 10 categories remaining after filtering, these categories are used within the *nuScenes* detection task, see section 3.1. Some of the categories in the first column are neglected in this step, they are displayed with a − in the second column. Finally, the third column contains the category ID for the detection classes in the second column. The table is partly replicated from [1]

nuScenes object class	Detection Class	Category ID
car	car	1
truck	truck	2
bus.bendy	bus	3
bus.rigid		
trailer	trailer	4
construction	construction_vehicle	5
adult	pedestrian	6
child		
construction_worker		
police_officer		
motorcycle	motorcycle	7
bicycle	bicycle	8
trafficcone	traffic_cone	9
barrier	barrier	10
personal_mobility	−	−
stroller	−	−
wheelchair	−	−
animal	−	−
debris	−	−
pushable_pullable	−	−
bicycle_rack	−	−
ambulance	−	−
police	−	−

Table A.2: Attributes within the *nuScenes* dataset. The first two columns contain the Categories and their corresponding IDs to which the attributes in column three and four are applicable. Column three contains the attribute name and the last column its corresponding ID.

Category IDs	Categories	Attribute	Attribute ID
9 10	traffic_cone barrier	–	0
7	motorcycle	cycle.with_rider	1
8	bicycle	cycle.without_rider	2
6	pedestrian	pedestrian.moving	3
		pedestrian.standing	4
		pedestrian.sitting_laying_down	5
1 2 3 4 5	car truck bus trailer construction_vehicle	vehicle.moving	6
		vehicle.parked	7
		vehicle.stopped	8