Semantic Environment Perception for Automated Driving with Deep Neural Networks

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Maschinenbau
des Karlsruher Instituts für Technologie (KIT)

angenommene

Dissertation

von

M.Sc. Juncong Fei

Tag der mündlichen Prüfung: 09.04.2025

Hauptreferent: Prof. Dr.-Ing. Christoph Stiller
Korreferent: Prof. Dr.-Ing. Klaus Dietmayer

Preface

This thesis was written during my time as a PhD student at Opel Automobile GmbH, supervised by Prof. Dr. Christoph Stiller from the Institute of Measurement and Control Systems (MRT) at the Karlsruhe Institute of Technology (KIT).

First of all, I would like to extend my deepest gratitude to Prof. Dr. Christoph Stiller for his invaluable guidance and motivational support throughout my PhD journey. His expertise and insightful feedback have been fundamental in advancing my research. Moreover, I would like to thank Prof. Dr. Klaus Dietmayer from Ulm University for his interest in my work and role as co-examiner.

The research work detailed in this dissertation was carried out within the scope of the research project KI-Absicherung at Opel Automobile GmbH. Special thanks to my industrial supervisor, Dr. Philipp Heidenreich, for many inspiring discussions about my research, his supportive mentorship, and co-authoring publications. I also want to thank my manager, Dr. Nikolas Wagner, and the project leader, Frank Bonarens, for their continuous encouragement and support during my PhD studies, and for creating a great research environment that enabled this thesis. Extra thanks to my Opel colleagues, Lukas Stäcker, Ahmed Hammam, Patrick Feifel, Dr. Seyed Ghobadi, Dr. Ulrich Eberle, and Dr. Christoph Thiem for their great collaboration and many inspiring discussions.

Furthermore, I want to thank all the PhD students at MRT for many great conversations. Special thanks go to Felix Hauser for his tremendous support with the organizational matters I encountered during my PhD time.

I also wish to express my deep gratitude to my parents for their unconditional love and unwavering support over the years. Finally, I am immensely grateful to my wife Lei for her encouragement and for always being there for me during this challenging journey.

Stuttgart, June 2024 *Juncong Fei*

Kurzfassung

Um eine zuverlässige Umweltwahrnehmung zu erlangen, sind automatisierte Fahrzeuge normalerweise mit einer Reihe von komplementären Sensoren, wie Kameras und LiDAR, ausgestattet. Nur durch eine akkurate Kombination der Sensordaten kann ein umfassendes Modell der Umgebung erstellt werden, das dann als einheitliche Abstraktionsschicht für nachfolgende Module im automatisierten Fahrsystem dient. Ein solches Umgebungsmodell muss sowohl Informationen über die statische Umgebung als auch über die dynamischen Verkehrsteilnehmer enthalten.

Diese Arbeit stellt ein hybrides Umgebungsmodell vor, das sich aus zwei Aufgaben ableitet: der semantischen Rasterkartengenerierung für die statische Umgebung und der 3D-Objekterkennung für die dynamischen Verkehrsteilnehmer. Diese Aufgaben werden erfolgreich mittels maschineller Lernverfahren bewältigt, die tiefe neuronale Netze einsetzen und umfangreiche, annotierte Datensätze nutzen.

Die vorliegende Dissertation beschäftigt sich zum einen mit der Bestimmung einer dichten semantischen Rasterkarte und stellt mit PillarSegNet dafür ein neuartiges End-to-End-Verfahren vor. Im Gegensatz zu bestehenden Methoden, die eine dünn besetzte LiDAR-Punktwolke zunächst in eine mehrschichtige, zweidimensionale Merkmalsdarstellung transformieren, bevor die semantische Segmentierung angewendet wird, lernt PillarSegNet die Merkmale direkt aus der LiDAR-Punktwolke. Das vorgeschlagene Verfahren wird mit dem SemanticKITTI-Datensatz trainiert und evaluiert. Dabei wird die Effektivität und die Überlegenheit der gelernten Darstellung gegenüber handgefertigten mehrschichtigen Merkmalsdarstellungen demonstriert. Darüber hinaus werden durch Ray-Casting gewonnene Belegungsinformationen integriert, wodurch die Leistung zusätzlich verbessert wird.

Weiterhin wird in der Arbeit mit SemanticVoxels ein effektiver Fusionsansatz für die 3D-Objekterkennung präsentiert. Dabei werden räumliche Infor-

mationen aus der LiDAR-Punktwolke mit semantischen Informationen aus Kamerabildern sequenziell fusioniert. Die vorgeschlagene Methode wird sowohl mit dem KITTI- als auch mit dem nuScenes-Datensatz trainiert und evaluiert. Deutliche Leistungsverbesserungen an beiden Datensätzen im Vergleich zu Verfahren, die ausschließlich LiDAR-Daten verwenden, bestätigen die Wirksamkeit des vorgeschlagenen Fusionsansatzes. Insbesondere können so weit entfernte oder teilweise verdeckte Verkehrsteilnehmer in herausfordernden Szenarien zuverlässig erkannt und die falsch-positiven Detektionen insgesamt reduziert werden. Außerdem wird SemanticVoxels erfolgreich auf einem eingebetteten Edge-AI-Gerät eingesetzt, wobei mit der durchgeführten Laufzeitoptimierung eine Prozessierung in Echtzeit erreicht werden kann. Weiterhin kann eine zuverlässige Erkennungsleistung in unbekannten realen Daten mit einem vergleichbaren LiDAR-Sensor beobachtet werden, was die herausragende Generalisierungsfähigkeit demonstriert.

Abstract

To achieve reliable scene understanding, automated vehicles are usually equipped with a series of complementary sensors, such as cameras and LiDARs. Their data needs to be accurately combined to create a comprehensive environment model that serves as a unified abstraction layer for subsequent modules in the automated driving system. Such an environment model must contain information about both the static environment and the dynamic traffic participants.

This thesis presents a hybrid environment model derived from two tasks: semantic grid map estimation for the static environment and 3D object detection for the dynamic traffic participants. These tasks are effectively addressed using machine learning methods that employ deep neural networks and leverage large-scale annotated datasets.

To approach dense semantic grid map estimation, the thesis proposes a novel end-to-end method named PillarSegNet. While existing methods first transform a sparse LiDAR point cloud into a 2D multi-layer grid map representation before applying semantic segmentation, PillarSegNet directly learns features from the input point cloud and forms a top-view representation. The proposed method is trained and evaluated on the SemanticKITTI dataset, demonstrating its effectiveness and the superiority of the learned representation over hand-crafted grid map representations. The thesis also incorporates occupancy information obtained via ray-casting into PillarSegNet for further performance enhancement.

For 3D object detection, the thesis presents an effective fusion approach termed SemanticVoxels, which fuses spatial information from LiDAR point clouds and semantic information from camera images in a sequential fashion. The proposed method is trained and evaluated on the KITTI and nuScenes datasets. Significant performance improvements on both datasets verify its effectiveness. Especially, when compared to 3D object detection using only LiDAR data, it is shown that incorporating semantic information from camera images helps

to detect challenging distant or occluded traffic participants and reduce false positive detections. Additionally, SemanticVoxels is successfully deployed on an embedded edge AI device with runtime optimization, achieving real-time performance. It shows reliable detection performance on unseen real-world data, despite only being trained on a dataset with a similar LiDAR, thereby demonstrating the outstanding generalization ability of SemanticVoxels.

Table of Contents

Abbreviations and Notations

Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
AdaGrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
API	Application Programming Interface
ASPP	Atrous Spatial Pyramid Pooling
BEV	Bird's Eye View
BatchNorm	Batch Normalization
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRF	Conditional Random Field
CUDA	Compute Unified Device Architecture
DNN	Deep Neural Network

FCN	Fully Convolutional Network
FN	False Negative
FNN	Feedforward Neural Network
FP	False Positive
FPS	Farthest Point Sampling
fps	frames per second
GPU	Graphics Processing Unit
IoU	Intersection over Union
kNN	k Nearest Neighbor
LiDAR	Light Detection And Ranging
LSS	Lift Splat Shoot
mAP	mean Average Precision
ML	Machine Learning
MLP	Multilayer Perceptron
mIoU	mean Intersection over Union
NDS	nuScenes Detection Score
NMS	Non-Maximum Suppression
ONNX	Open Neural Network Exchange
PCL	Point Cloud Library
PFN	Pillar Feature Network
RADAR	Radio Detection And Ranging
ReLU	Rectifield Linear Unit

RMSProp	Root Mean Squared Propagation
ROS	Robot Operating System
RPN	Region Proposal Network
SDK	Software Development Kit
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TP	True Positive
VFE	Voxel Feature Encoding
VRU	Vulnerable Road User

Notations

$\mathbf{x}$	Input sample vector
$\mathbf{y}$	Label vector
$\hat{y}$	Predicted output scalar
$\hat{\mathbf{y}}$	Predicted output vector
$\mathbf{w}$	Trainable weight vector in a neural network
$\boldsymbol{\theta}$	Trainable parameters in a neural network
η	Learning rate
$\mathcal{L}$	Loss
$\mathcal{N}$	Normal distribution

1 Introduction

In recent years, both academia and industry have shown great interest in automated driving, and tremendous advancements have been achieved in this field. Automated driving offers the potential for numerous advantages. It can improve road safety by reducing human errors and enhance transportation accessibility for people who cannot drive, such as the elderly and disabled people [Gol18]. Additionally, automated driving can reduce traffic congestion and improve traffic efficiency while lowering fuel consumption and harmful emissions [Gol18].

A typical software architecture for an automated driving system consists of three interconnected core modules: perception, planning, and control [PAD+17]. Using onboard sensor data, the perception module constructs a comprehensive environment model to understand the automated vehicle's surroundings. The subsequent planning module relies on the environment model to determine the vehicle's motions, which are executed by the control module. Therefore, as the foundation for other modules, the perception and adequate representation of the driving environment are critical for the automated driving system.

1.1 Motivation

To achieve reliable scene understanding, automated vehicles are usually equipped with a number of complementary sensors, among which Light Detection And Ranging (LiDAR) sensors and cameras are commonly used. LiDAR point clouds provide accurate spatial information, while camera images are able to provide rich contextual and semantic information. Data from various sensors needs to be accurately combined to create a comprehensive environment model that serves as a unified abstraction layer for subsequent modules in the auto-

mated driving system. Such an environment model must contain information about

- the static environment, such as occupied and drivable free-space areas, and
- dynamic traffic participants, such as vehicles, pedestrians, and bicyclists [Sch18].

In automated driving, 2D occupancy grid maps are commonly used to represent the static environment, where the mapping area is discretized into a grid of cells in the top view. Depending on the sensor measurements, each grid cell is assigned a probability of being occupied. 2D grid maps provide a concise and accurate representation of the static environment and the presence of obstacles. As an advancement of conventional occupancy grid maps, a semantic grid map assigns a semantic class to each grid cell. By doing so, it can be advantageous to further distinguish free-space areas such as roads, sidewalks, or terrain, and occupied areas such as parked vehicles, buildings, or vegetation.

The semantic grid map is effective at representing the static surroundings. However, it is challenging for it to reliably perceive dynamic objects, especially moving objects. In addition, some automated driving modules, such as behaviour and motion planning, may require discrete object-level representations of dynamic objects. To address these challenges, 3D object detection has proven to be an appropriate solution. It determines the pose, dimensions, and semantic class of traffic participants and represents them by oriented cuboids in 3D space. This discrete representation can be easily associated with additional attributes of traffic participants extracted by other algorithms, such as motion.

Therefore, it is a suitable choice to combine semantic grid map estimation and 3D object detection to understand the surrounding environment, with the former focusing on static environment modeling and the latter targeting dynamic object perception.

In order to derive precise spatial information of the static environment, LiDAR data is primarily used to approach semantic grid map estimation. A state-of-the-art method [BWJ+20] transforms a sparse LiDAR point cloud into a multi-layer grid map representation and applies semantic segmentation to estimate a dense semantic grid map. However, the hand-crafted grid map feature extraction can

result in a potential loss of information. To overcome this challenge, a novel end-to-end approach named PillarSegNet is proposed, which uses a simplified PointNet [QSMG17] to learn features directly from the input point cloud. In addition to the learned features, the benefit of aggregating different occupancy information obtained via model-based ray-casting is investigated.

In 3D object detection, LiDAR spatial information is crucial for localizing objects and estimating their shapes, while image semantic information is needed for reliably classifying objects and interpreting scenes. Thus, it is appropriate to leverage these complementary knowledge by fusing LiDAR and camera data, aiming to achieve better performance. To this end, a fusion-based approach termed SemanticVoxels is developed and validated in this work. Furthermore, SemanticVoxels is successfully deployed on an embedded edge AI device, achieving real-time performance.

1.2 Contributions

The objective of this thesis is to develop deep learning approaches for semantic environment perception for automated driving using onboard sensor data. This is achieved with a hybrid environment model obtained from two tasks: semantic grid map estimation for static environment modeling and 3D object detection for perceiving dynamic traffic participants. The proposed approach for the former is based on LiDAR data, while the approach for the latter additionally fuses camera image data.

The main contributions of this thesis are the following:

- A novel end-to-end approach named PillarSegNet is proposed for semantic grid map estimation based on sparse LiDAR data, aiming to avoid the information bottleneck caused by hand-crafted feature engineering. Occupancy information obtained via ray-casting is further incorporated into PillarSegNet to enhance the overall performance.
- An effective sequential fusion approach termed SemanticVoxels is proposed for 3D object detection by exploiting spatial information from the LiDAR point cloud and semantic information from the camera image.

Extensive experiments are conducted on various datasets to validate the proposed method.

- SemanticVoxels is successfully deployed on an embedded edge AI device with runtime optimization, achieving real-time performance. It shows reliable detection performance on unseen real-world data, despite only being trained on a dataset with a similar LiDAR sensor, thereby demonstrating the outstanding generalization ability of SemanticVoxels.

1.3 Thesis Outline

The remainder of this thesis is structured as follows:

Chapter 2 provides the relevant background and an overview of related work. It first introduces the fundamentals of deep learning and its applications on LiDAR point clouds. Afterwards, research related to semantic grid map estimation and 3D object detection is presented. Finally, it introduces the datasets used in this thesis.

Chapter 3 introduces the proposed end-to-end approach for semantic grid map estimation using sparse LiDAR data. Experimental evaluations on the SemanticKITTI [BGM+19] dataset demonstrate the effectiveness of the proposed method. Further experiments investigate the advantages of incorporating occupancy information as well as various data augmentation techniques.

Chapter 4 focuses on 3D object detection for perceiving dynamic traffic participants. A fusion approach that leverages both LiDAR and camera data is introduced. To prove its effectiveness, extensive experimental evaluations are conducted on both the KITTI [GLU12] and nuScenes [CBL+20] datasets. The proposed method is successfully deployed on an embedded platform with runtime optimization and real-time inference speed is achieved.

Finally, Chapter 5 concludes the thesis and offers an outlook for potential future research directions.

2 Technical Background

2.1 Deep Learning Fundamentals

As a part of artificial intelligence, Machine Learning (ML) describes methods that leverage data to make predictions or decisions on certain tasks, without being explicitly programmed [Mit97, Sam59]. In supervised learning, consider a labelled dataset denoted by $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, which contains N pairs of data examples, with each pair consisting of input data $\mathbf{x}_i$ and its corresponding target label $\mathbf{y}_i$. A machine learning system learns a function $f_{\boldsymbol{\theta}} : \mathbf{x} \rightarrow \hat{\mathbf{y}}$ with parameters $\boldsymbol{\theta}$ that maps input data $\mathbf{x}$ into the predicted output $\hat{\mathbf{y}}$.

Deep learning is a specific kind of machine learning. It uses Artificial Neural Networks (ANNs) with multiple layers, also known as Deep Neural Networks (DNNs), to progressively extract higher-level features from the raw input. Over the past few decades, deep learning approaches have proven successful in computer vision tasks such as image classification and object detection.

In this section, the basics of deep learning will be covered. First, Section 2.1.1 introduces artificial neural networks. Then, two common networks: feedforward neural networks (FNNs) and convolutional neural networks (CNNs) are described in Section 2.1.2 and Section 2.1.3, respectively. In Section 2.1.4, the training of neural networks is presented. Finally, regularization techniques are introduced in Section 2.1.5.

2.1.1 Artificial Neural Networks

ANNs are computing systems inspired by the biological brain which is constructed from biological neurons [GBC16]. They are created to mimic the functioning of biological brains. An ANN is composed of a collection of connected nodes which are called artificial neurons or perceptrons. A node

receives one or more inputs and then produces an output, while a connection transmits a signal to other nodes.

As the elementary computation unit in ANNs, each neuron performs calculation independently. Mathematically, its calculation can be described as follows:

$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b), \tag{2.1}$$

where $\hat{y}$ is the scalar output of the neuron, obtained by applying the activation function σ to the weighted sum of inputs plus a bias b. The notations $\mathbf{w}$ and $\mathbf{x}$ represent the weight vector and input vector, respectively, while the symbol $\cdot$ indicates the dot product between them. During training, both the weights and the bias are learnable parameters, while the activation function introduces non-linearity into the neuron, enabling it to learn and model more complex patterns. Commonly used activation functions include sigmoid, Hyperbolic Tangent (tanh), and Rectifield Linear Unit (ReLU).

2.1.2 Feedforward Neural Networks

FNNs, also known as Multilayer Perceptrons (MLPs), are a common type of ANNs in which the nodes are structured in a directed acyclic graph without feedback loops. They consist of multiple layers of neurons, including an input layer, an output layer, and one or more hidden layers in between. Besides, a layer in which each neuron is connected to every neuron in the preceding layer is called a Fully Connected (FC) layer.

Considering a FNN with L layers, the operation of the l-th layer can be represented by a mapping $\hat{\mathbf{y}} = f^{(l)}(\mathbf{x})$, which maps the input $\mathbf{x}$ to the output $\hat{\mathbf{y}}$ through the function $f^{(l)}$. Thus, the FNN can be formulated as a composition of the mappings represented by each layer, as expressed by the equation:

$$\begin{aligned} f(\mathbf{x}) &= f^{(L)} \circ f^{(L-1)} \circ \ldots \circ f^{(2)} \circ f^{(1)}(\mathbf{x}) \\ &= f^{(L)}(f^{(L-1)}(\ldots(f^{(2)}(f^{(1)}(\mathbf{x}))))), \end{aligned} \tag{2.2}$$

in which the operation $\circ$ represents function composition.

2.1.3 Convolutional Neural Networks

CNNs are a specialized type of ANNs for processing data that has a grid-like topology, such as images [GBC16]. A CNN performs convolution operations by sliding convolution kernels across the input data, thereby progressively learning and extracting hierarchical features. This section first introduces the convolution operation, and then describes two typical layers in CNNs, namely the conventional layer and the pooling layer.

The Convolution Operation The convolution operation is a fundamental building block of CNNs, where 2D convolution is commonly employed to extract features from input data, such as images. Given a 2D input I and a 2D kernel K, the 2D convolution operation is defined as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{2.3}$$

where S denotes the output and is often called the feature map. This can be intuitively understood as sliding a flipped kernel over the input and applying an element-wise multiplication and a summation for every position to construct the output feature map. Note that in popular machine learning frameworks such as TensorFlow [AAB+16] and PyTorch [PGM+19], the convolution operation is typically implemented without the step of flipping the kernel, making it equivalent to the operation of cross-correlation.

Convolutional Layer The convolutional layer is a key component of CNNs, responsible for learning and extracting relevant features from the input data. It consists of a set of learnable kernels, each of which typically has a small height and width, but with the same depth as the input. By sliding these convolutional kernels across the input data with a specified stride s (i.e., moving the kernel s pixels at a time), and performing the convolution operation at each position, multiple feature maps are generated, each corresponding to a distinct kernel. The number of feature maps is determined by the number of applied kernels. These feature maps are further stacked along the depth dimension and are used as input to the subsequent layers in CNNs. To define a conventional layer,

several key parameters need to be specified, including the number of kernels, their sizes (height and width), the stride, and the amount of padding.

In contrast to the FC layer, the convolutional layer has two main characteristics: local connection and parameter sharing. Local connection means that each convolutional kernel processes data exclusively within its receptive filed, while parameter sharing refers to the concept of using the same set of kernels across various spatial positions of the input. These characteristics make convolutional layers more adept at learning spatially localized features and achieve translation invariance while being more computationally efficient. Therefore, convolutional layers are well-suited for spatially structured data, such as images.

Pooling Layer The pooling layer is widely used in CNNs and is usually added after the convolutional layer. It performs downsampling by summarizing features present in each region of the feature maps in order to reduce the spatial dimensions of the feature maps while keeping important information. Two common types of pooling layers are max pooling and average pooling.

In CNNs, the pooling layer offers several advantages. By reducing the dimensionality of feature maps, it improves the computational efficiency and also contributes to avoiding overfitting through a reduction in the number of parameters in the model. Moreover, in combination with convolutional layers, pooling layers help to achieve translation invariance, making CNNs more robust in feature extraction even when the input data is translated.

2.1.4 Training

The training of a neural network can be viewed as finding a set of parameters $\boldsymbol{\theta}$ that minimizes the difference between the predicted output $\hat{\mathbf{y}}$ and the ground truth label $\mathbf{y}$, as quantified by a cost function: $J(\boldsymbol{\theta})$. Typically, the cost function is derived by evaluating the loss function $\mathcal{L}$ for each training example $(\mathbf{x}_i, \mathbf{y}_i)$ and summing the results across the entire training set, giving:

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i). \tag{2.4}$$

Optimizers

Stochastic Gradient Descent In order to minimize the cost function, gradient descent is commonly used during the training. It is an iterative optimization algorithm. In each iteration, it calculates the gradient of the cost function with respect to the parameters $\boldsymbol{\theta}$, and updates the model's parameters in the direction of steepest descent, as defined by the negative gradient. The update step is formulated as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \cdot \nabla J(\boldsymbol{\theta}_t). \tag{2.5}$$

Here, $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t+1}$ represent the model's parameters at iteration t and after the update, respectively. η is the learning rate which controls the step size of the update. $\nabla J(\boldsymbol{\theta}_t)$ denotes the gradients of the cost function J with respect to the parameters $\boldsymbol{\theta}_t$. To effectively compute the gradients, the backpropogation [RHW86] algorithm is used. It applies the chain rule to propagate error derivatives backwards from the output layer to the input layer and compute all intermediate gradients.

Modern neural networks usually require large training datasets to achieve good performance and generalization. Therefore, the vanilla gradient descent can pose a significant computational overhead and slow down convergence speed because it processes the entire training dataset in every iteration. To address these challenges, Stochastic Gradient Descent (SGD) is commonly employed in practice. In each iteration, rather than using all training examples, SGD randomly selects a single example (or a mini-batch) to calculate the gradient and update the model's parameters.

SGD with Momentum As SGD solely considers the current gradient, it may get stuck at local minima or saddle points when the gradient is zero. In addition, SGD can also lead to oscillations (or zigzag pattern) when the loss surface has elongated valleys or the gradients are noisy due to the stochastic nature of its mini-batch updates. These issues can be addressed by SGD with momentum [Qia99], which introduces a momentum term that accumulates a fraction of past gradients. The momentum term allows the algorithm to

escape local minima and saddle points more effectively and counteract against oscillations. The update equation for SGD with momentum is given by:

$$\begin{aligned} \boldsymbol{m}_{t+1} &= \beta \cdot \boldsymbol{m}_t + (1-\beta) \cdot \nabla J(\boldsymbol{\theta}_t), \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta \cdot \boldsymbol{m}_{t+1}. \end{aligned} \tag{2.6}$$

Here, $\boldsymbol{m}$ denotes the momentum, and the parameter β controls the influence of historical gradients. A typical value for β is 0.9. In the first iteration, the momentum term $\boldsymbol{m}_0$ is often initialized to zero.

Adam When training neural networks, it is usually helpful to anneal the learning rate gradually over time. Besides, setting different learning rates for different parameters is advantageous. For instance, parameters with higher gradients or frequent updates benefit from smaller learning rates to avoid overshooting minima, while those with lower gradients or infrequent updates benefit from larger learning rates for faster convergence. To this end, AdaGrad [DHS11] and RMSProp [Hin12] utilize the squares of past gradients to adjust the learning rate of each parameter adaptively.

Adaptive Moment Estimation (Adam) [KB15] is a popular and powerful optimizer that combines the advantages of both RMSProp and SGD with momentum. It maintains moving averages of both past gradients and squares of past gradients to adaptively adjust the learning rate for each parameter. In addition, Adam includes bias correction terms to counteract the biases towards zero in its moment estimates during the early training iterations. The update step of Adam is formulated as follows:

$$\begin{aligned} \boldsymbol{m}_{t+1} &= \beta_1 \cdot \boldsymbol{m}_t + (1-\beta_1) \cdot \nabla J(\boldsymbol{\theta}_t), \\ \boldsymbol{v}_{t+1} &= \beta_2 \cdot \boldsymbol{v}_t + (1-\beta_2) \cdot (\nabla J(\boldsymbol{\theta}_t))^2, \\ \hat{\boldsymbol{m}}_{t+1} &= \frac{\boldsymbol{m}_{t+1}}{1-\beta_1^{t+1}}, \\ \hat{\boldsymbol{v}}_{t+1} &= \frac{\boldsymbol{v}_{t+1}}{1-\beta_2^{t+1}}, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\hat{\boldsymbol{v}}_{t+1}} + \epsilon} \cdot \hat{\boldsymbol{m}}_{t+1}. \end{aligned} \tag{2.7}$$

In the above equations, $\boldsymbol{m}$ and $\boldsymbol{v}$ are vectors representing the exponentially weighted averages of the historical gradients and the element-wise squared past gradients, respectively. β_1 and β_2 are the corresponding exponential decay rates. $\nabla J(\boldsymbol{\theta}_t)$ is the gradient at iteration t. Note that the squared gradient $(\nabla J(\boldsymbol{\theta}_t))^2$ is computed element-wise. Besides, $\hat{\boldsymbol{m}}$ and $\hat{\boldsymbol{v}}$ represent moment estimates after applying the corresponding bias correction terms $1 - \beta_1^{t+1}$ and $1 - \beta_2^{t+1}$. η is the learning rate and ϵ is a small value to avoid dividing by zero.

Loshchilov and Hutter [LH19] propose an improved version of Adam, named AdamW [LH19], which decouples weight decay from the gradient update step. Unlike the standard Adam, which applies weight decay by adding an L_2 penalty term to the loss function, AdamW directly subtracts a weight decay term from the weights during each parameter update step. This decoupling leads to more consistent regularization and better generalization.

2.1.5 Regularization

Regularization techniques are important for training neural networks to prevent overfitting and improve their generalization capabilities. This section introduces L_1 and L_2 regularization as well as Batch Normalization.

L_1 and L_2 Regularization L_1 and L_2 regularization are both types of weight decay, which is a regularization technique that adds a penalty term to the cost function of a model in order to prevent its parameters from being too large.

The penalty term in L_1 regularization is proportional to the L_1 norm of the model's parameters, while the one in L_2 regularization is proportional to the square of the L_2 norm of the parameters, giving:

$$\begin{aligned} J_{L_1}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1, \\ J_{L_2}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2. \end{aligned} \tag{2.8}$$

Here, $\boldsymbol{\theta}$ represents the parameter vector of the model, $J(\boldsymbol{\theta})$ denotes the original cost function, and λ is the weight decay coefficient. $\|\boldsymbol{\theta}\|_1$ and $\|\boldsymbol{\theta}\|_2$ denote the L_1 and L_2 norms of the parameter vector $\boldsymbol{\theta}$, respectively. Both penalty terms

encourage small parameter values and reduce the complexity of models in order to prevent overfitting.

Batch Normalization Batch Normalization (BatchNorm) [IS15] is a widely used technique to stabilize and accelerate the training of neural networks. During the training phase, each new input example results in a different distribution of inputs for each layer of the model. Thus, lower learning rates and more careful parameter initialization are required, which slows down training. In addition, models may suffer from non-linearity saturation, thus become difficult to train. BatchNorm tackles the above challenges through normalizing the inputs of each layer to have approximately zero mean and unit variance [IS15]. This is achieved by computing the mean and variance for each layer across each mini-batch of training data.

During testing or inference, BatchNorm behaves differently than during training. It uses the population means and variances computed on the entire training dataset to normalize each layer's inputs. By doing so, consistent behavior is ensured during model testing. In practice, it is common to fuse BatchNorm with convolutional layers for inference efficiency.

2.2 Deep Learning on Point Clouds

A point cloud is a set of points in 3D space that represents geometric information of a physical object or a scene. Each data point in the point cloud usually contains spatial information such as its 3D coordinates (x, y, z), and may also contain extra features like reflectance and color.

In the fields of automated driving, point clouds are typically collected through LiDAR sensors. 3D LiDAR point clouds provide a highly accurate representation of the surrounding environment, making them essential for perception tasks such as 3D object detection and scene understanding. However, LiDAR point clouds are inherently unordered, irregular and sparse, which makes it challenging to process them. In particular, typical deep learning approaches for images cannot be directly adopted to point clouds. As a result, additional pre-processing or new deep learning architectures are needed to address these challenges and enable the effective processing of 3D point clouds.

This section covers the basics of deep learning for LiDAR point clouds, especially in the domain of automated driving. First, Section 2.2.1 introduces the main properties of point clouds. Then, different representation forms are described in Section 2.2.2. In Section 2.2.3, a unified deep learning architecture for processing point cloud data, named PointNet [QSMG17], is presented. Finally, Section 2.2.4 introduces a powerful and efficient feature extractor, which learns features from raw point cloud data and forms a structured representation.

2.2.1 Properties of Point Clouds

As an important type of geometric data, point clouds have several unique properties:

- Unordered. Unlike 2D images, a point cloud is a set of points without a specific order. Therefore, typical deep learning methods that rely on structured data inputs cannot be directly applied on point clouds.
- Irregular. The density and distribution of points in the 3D space can vary greatly depending on the surface geometry of objects in the scene, as well as the location of the acquisition sensor. This is challenging for many applications such as automated driving, which requires robust environment perception.
- Sparse. Due to the principle of point cloud collection, the captured data is usually sparse and only represents parts of physical objects. Despite being sparse, points in a point cloud are not isolated but interact with local neighbors. To effectively process point clouds, it is crucial to handle their sparsity while also capturing local structure from nearby points.

Overall, the aforementioned properties of point clouds present many challenges for processing them. In order to overcome these challenges, it is common to pre-process point clouds and transform them into structured representations. In addition, by taking the above properties into account, new deep learning architectures have been proposed for direct processing of point clouds.

2.2.2 Structured Representations for LiDAR Point Clouds

Deep learning methods are widely used for processing structured data such as images and significant progress has been achieved in this field. Thus, one popular way to tackle the aforementioned challenges is to convert 3D point clouds into structured representations. In this way, existing deep learning methods can be easily applied to further process them.

There are mainly two ways to process them and get structured representations. One way is partitioning the 3D space occupied by the point cloud into regular voxel grids, and assigning LiDAR points to each voxel [MS15, Li17, WP15, WSK+15]. Then, each generated voxel can be described by different features, such as the presence and density of points within it. This voxel-based representation can be further processed by 3D convolutional neural networks. While it is possible to preserve 3D spatial structures well, this representation is sensitive to voxel resolution, resulting in a trade-off between computation and memory footprint on one side, and representation accuracy on the other. Moreover, the use of 3D convolutions on sparse volumes also introduces a high computational overhead. Therefore, many works tackle the above challenges, for instance, Vote3Deep [ERW+17] and SECOND [YML18] adopt sparse convolutions to improve the inference speed.

Another way of representing point clouds is to project them into a perspective view and obtain an image-like representation, which can be processed by common 2D convolutional layers. When focusing on LiDAR point clouds, there are three typically used views:

- Camera image view. Given the calibration parameters, 3D LiDAR points can be projected onto the camera image plane. The resulting map provides a similar front view as the camera and can be directly fused with camera images. However, this representation may contain some empty pixels. Additional up-sampling operations can be applied to address these gaps in the subsequent feature maps [PGA+16, PD18].
- Spherical view. To construct a dense front-view representation, 3D points can be mapped onto a sphere which is defined by the azimuth and elevation angles [LZX16]. Similar to an ordinary image, the spherical map represents the point cloud in a compact and regular manner, making it appropriate for point cloud segmentation [WWYK18, MVBS19].

- Bird's eye view (BEV). Another common choice is projecting 3D point cloud to the bird's eye view. In automated driving, the BEV representation has several advantages. Compared to the aforementioned front-view representations, the BEV map preserves each object's size, orientation, and position on the ground plane. In addition, such representation minimizes occlusions since objects usually do not overlap in the top view. Therefore, the BEV representation is widely used for various perception tasks in automated driving, such as object detection and environment modeling.

In summary, the above-mentioned structured representations offer an effective way to process LiDAR point cloud data using standard CNNs, leading to promising results for a variety of environment perception tasks in automated driving.

2.2.3 PointNet

PointNet [QSMG17] is a unified deep learning architecture designed for processing 3D point cloud data. In contrast to many existing methods that require structured representations of 3D data (such voxel grids or collections of images), PointNet directly takes unstructured point sets as input.

As shown in Figure 2.1, a lightweight PointNet is composed of two main components: a shared MLP and a max pooling layer. Assume an input point cloud containing n points $\{P_i \mid i = 1, ..., n\}$, where each point P_i is fully described by its (x, y, z) coordinates for simplicity. The shared MLP in PointNet takes a single point P_i as input and learns high-dimensional features for that point. Then, the generated m-dimensional point-wise feature vectors for all points are aggregated by a max pooling layer to obtain a global feature vector for the entire point cloud. Mathematically, the processing of PointNet can be represented as:

$$F_{\text{global}} = g(h(P_1), ..., h(P_n)), \tag{2.9}$$

where h denotes the shared MLP network, g stands for the max pooling function, and F_{global} is the global feature vector.

As mentioned above, point clouds are inherently unstructured data and are represented as 3D point sets. As an example, given n data points, there are in

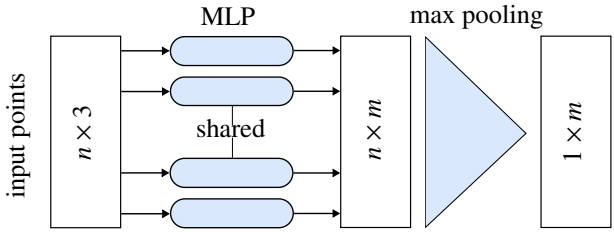


Figure 2.1: A lightweight PointNet.

total $n!$ possible permutations. In order to be invariant to input permutations, PointNet adopts a symmetric function which returns the same output value, regardless of the order of its input arguments. Specifically, the max pooling layer is used as a symmetric function to aggregate information from all unordered points into global features.

PointNet learns features independently for each point. As a result, it could not capture local structural information between points by design. To resolve this limitation, Qi *et al.* [QYSG17] propose a hierarchical network PointNet++ that employs PointNet recursively on nested partitions of the input point cloud. The hierarchy of PointNet++ is composed of multiple set abstraction levels, each containing three key layers: the sampling layer, the grouping layer, and the PointNet layer. First, the sampling layer applies iterative farthest point sampling (FPS) to choose a set of points from the input points, which defines the centroids of local regions. Afterwards, the grouping layer selects neighboring points for each centroid using ball query or k nearest neighbor (kNN) search. Finally, the PointNet layer utilizes a mini-PointNet to extract local features for each group of points. By stacking a number of set abstraction levels, PointNet++ progressively learns local features with increasing contextual scales and abstracts larger local regions.

2.2.4 Pillar Feature Network

As introduced in Section 2.2.2, it is common to represent 3D point clouds in the top view in automated driving, due to its advantages including scale invariance and minimal occlusions. While many approaches encode point clouds in the top view using hand-crafted features such as density and height information

[CMW+17, KML+18], the Pillar Feature Network (PFN) employs a simplified PointNet to directly learn features from raw point cloud data and form a BEV representation [LVC+19].

An overview of the PFN is shown in Figure 2.2. It consists of two main modules, namely pre-processing and feature extraction. In the pre-processing stage, the input point cloud is first discretized into a set of pillars in the ground plane, where pillar is a special voxel without an extension bound in the Z axis. Each LiDAR point with (x, y, z, r) encoding in a pillar is further augmented with its offsets from the arithmetic mean $(\Delta x_c, \Delta y_c, \Delta z_c)$ of all points in the pillar and its offsets from the pillar center $(\Delta x_p, \Delta y_p, \Delta z_p)$. The dimension of the resulting point encoding thus becomes $D = 10$. The number of pillars per point cloud (P) and the number of points per pillar (N) are pre-defined to create an input tensor of fixed size (P, N, D). To this end, the points are randomly sampled if a pillar has more than N points, or zero padding is applied to populate the tensor when a point cloud has less than P non-empty pillars or a pillar has too few points.

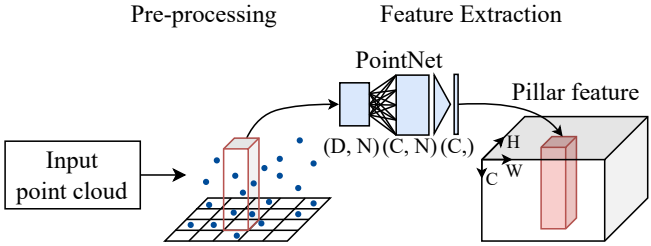


Figure 2.2: The architecture of the Pillar Feature Network (PFN). Modified from Figure 2 in [SFH+21] ©IEEE.

After pillarization and point feature decoration, in the feature extraction step, the feature of each point in pillars is then processed by a simplified PointNet [QSMG17] that consists of a linear layer, BatchNorm, and ReLU, outputing a tensor of size (P, N, C). Then, a max operation along the N axis is applied to create an output tensor of size (P, C), where each pillar is represented by a feature vector of size (C). Finally, all pillar-wise features are scattered back to the pillar locations to create a top-view representation of size (W, H, C), where W and H denote the width and height of the grid, corresponding to the X and Y axes, respectively.

2.3 Semantic Segmentation

Semantic segmentation is a key perception task for scene understanding in automated driving. It aims to partition an image or a point cloud into multiple segments or regions that belong to the same object class. Semantic segmentation can be considered as a classification task in computer vision that assigns a semantic class to each pixel of an image or point in a LiDAR scan. Figure 2.3 illustrates an example image and its semantic segmentation annotation, where unique colors in the segmentation map represent different object categories. An excerpt of the used color scheme is shown in Table 2.1.

This section provides an introduction of semantic segmentation methods in deep learning. First, Section 2.3.1 introduces several popular methods for image semantic segmentation. Afterwards, semantic segmentation for LiDAR point cloud is described in Section 2.3.2 and Section 2.3.3, including point-wise segmentation and semantic grid map estimation. Lastly, Section 2.3.4 presents evaluation metrics for semantic segmentation.

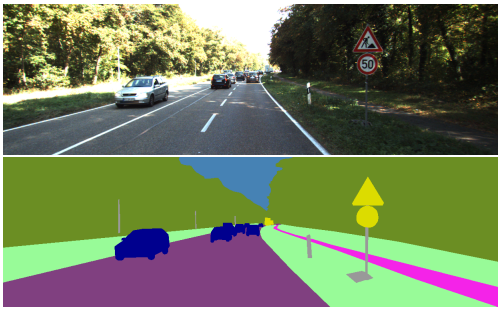


Figure 2.3: An example image (top) from the KITTI dataset [AMM+18] and its semantic segmentation annotation (bottom). The color scheme representing different object classes is detailed in Table 2.1.

2.3.1 Image Semantic Segmentation

Image-based semantic segmentation networks are widely used in autonomous vehicles and significant progress is being made in that field. As a pioneering

road	sidewalk	vegetation	terrain
sky	building	wall	fence
traffic light	traffic sign	car	person
rider	bicycle	motorcycle	truck
trailer	train	bus	caravan

Table 2.1: Selected Cityscapes [COR+] color scheme used in Figure 2.3 and Chapter 4.

work, Long *et al.* [LSD15] propose a Fully Convolutional Network (FCN) for semantic segmentation, replacing FC layers in a CNN classifier with convolutional layers to process images of arbitrary size and generate segmentation maps. FCN also integrates skip connections to combine deep semantic and shallow appearance information. Another popular paradigm for semantic segmentation is based on the encoder-decoder architecture [NHH15, BKC17], which contains an encoder that successively downsamples the input image and extracts semantic details, and a decoder that gradually restores resolution of the encoder's feature maps for pixel-wise classification.

U-Net [RFB15] builds upon the concept of FCN. It consists of a contracting path to capture context information and a symmetric expanding path that enables precise localization, making it an encoder-decoder network as well. In U-Net, the encoder uses max-pooling layers for downscaling, while the decoder applies transposed convolutions for upsampling. U-Net also uses skip connections to concatenate feature maps from corresponding layers of the encoder and decoder, effectively retaining spatial information that might be lost during downsampling, thereby enhancing localization accuracy.

DeepLab [CPK+15] is a family of evolving CNNs designed for semantic segmentation, with each version introducing new mechanisms. DeepLabv1 [CPK+15] combines deep CNNs and Conditional Random Fields (CRFs) to improve segmentation accuracy. Besides, it employs atrous (dilated) convolutions to enlarge the receptive field without reducing the spatial resolution of the feature maps. DeepLabv2 [CPK+18] inherits these features and introduces the Atrous Spatial Pyramid Pooling (ASPP) module in the convolutional layers to capture objects and contexts at multiple scales. DeepLabv3 [CPSA17] further refines the ASPP module and incorporates cascade atrous convolutions for

more effective feature extraction, while removing the CRF post-processing to enable end-to-end network training.

DeepLabv3+ [CZP+18] extends its predecessor by combining the previously used ASPP module with an encoder-decoder architecture inspired by SegNet [BKC17] and U-Net [RFB15]. More specifically, it uses the DeepLabv3 network as encoder and appends a simple decoder module. A skip connection between the encoder and decoder is used to better recover object segmentation details. In DeepLabv3+, the authors further evaluate the Xception [Cho17] backbone and apply the atrous separable convolution in the network, achieving better segmentation performance and a faster runtime, respectively.

In many practical applications such as automated driving, it is crucial to perform semantic segmentation in real-time with limited computing resources. To tackle this challenge, Hong *et al.* [HPSJ21] propose a Deep Dual-Resolution Network (DDRNet). DDRNet features a dual-pathway architecture: one low-resolution pathway learns rich contextual information, and the other one maintains higher resolution for accurate spatial information, with several bilateral connections in between for effective information fusion. In the low-resolution branch, the authors design a Deep Aggregation Pyramid Pooling Module (DAPPM) to increases the receptive fields and extract context information more effectively. Finally, the DDRNet-23-slim variant achieves an inference speed of 109 frames per second (fps) on the Cityscapes [COR+] test set (measured on an NVIDIA 2080Ti GPU), while also delivering outstanding semantic segmentation results.

2.3.2 Point Cloud Semantic Segmentation

The immense progress made in image semantic segmentation has greatly contributed to the development of point cloud semantic segmentation. This section focuses on point-wise point cloud semantic segmentation. An example point cloud that is colorized based on point-wise segmentation labels is shown in Figure 3.1.

Point-wise LiDAR semantic segmentation aims to classify each 3D point in a LiDAR scan. Wu and Xu *et al.* propose a series of SqueezeSeg [WWYK18, WZZ+19, XWW+20] networks for efficient LiDAR semantic segmentation. SqueezeSegv1 [WWYK18] applies CNN and CRF on the spherical represen-

tation of LiDAR data to approach 2D semantic segmentation. Afterwards, the 2D output is restored to 3D to obtain point-wise segmentation results. Continuing with the spherical view, SqueezeSegv2 [WZZ+19] introduces a context aggregation module to aggregate contextual information from a larger receptive filed. SqueezeSegv3 [XWW+20] proposes spatially-adaptive convolution to handle spatially varying feature distributions in LiDAR spherical images.

Similarly, RangeNet++ [MVBS19] performs 2D semantic segmentation on a LiDAR spherical image, followed by a post-processing algorithm that addresses issues caused by blurry 2D segmentation outputs and discretization errors during the spherical projection. Triess *et al.* [TPRZ20] propose a scan unfolding method and a cyclic padding mechanism to mitigate systematic point occlusions encountered during the transformation of ego-motion corrected point clouds from the Cartesian to the spherical coordinate system.

RandLA-Net [HYX+20] approaches point-wise semantic segmentation directly on raw point clouds. For memory and computational efficiency, it uses random point sampling instead of complicated point selection methods, so that it can operate on large-scale point clouds with low latency. To counter the loss of key information during sampling, RandLA-Net proposes a local feature aggregation module to preserve useful features by progressively increasing the receptive field for each point.

Zhu *et al.* [ZZW+21] discover that the 2D spherical view representation abandons many valuable 3D structures during the 3D-to-2D projection. Thus, they propose Cylinder3D where a 3D point cloud is discretized into a set of 3D cylindrical partitions. The obtained 3D representation is further processed by a modified 3D U-Net [cAL+16] with sparse convolutions, followed by a lightweight segmentation head that predicts results for each partition.

2.3.3 Semantic Grid Map Estimation

In automated driving and robotics applications, 2D occupancy grid maps, first introduced in [Elf89], are widely used to represent the environment, where the environment is discretized into a grid of cells on the ground plane. Depending on sensor measurements, each grid cell is encoded with a probability of being occupied. As an advancement of conventional occupancy grid maps, a semantic

grid map assigns a semantic class to each grid cell. Figure 3.1 displays an example dense semantic grid map.

Before accurate semantic grid map labels are available, Erkent *et al.* [EWL+18] and Lu *et al.* [LvdMD18] generate pseudo ground truths by mapping the output of image semantic segmentation to the 3D space using the depth information or with the help of LiDAR points. Taking a monocular image as input, Lu *et al.* [LvdMD18] propose a variational encoder-decoder network to encode front-view image features and subsequently decode them into a top-view semantic grid map. Erkent *et al.* [EWL+18] first perform image semantic segmentation and project the output to a BEV representation via inverse perspective mapping. The obtained BEV maps are further fused with occupancy maps derived from LiDAR measurements and then processed by an encoder-decoder network to output a refined semantic grid map. The performance of these methods is greatly limited due to imperfect sensor calibration, parallax, and violation of the plane assumption at large distances.

The release of the SemanticKITTI [BGM+19] dataset, where every LiDAR point is labeled with a semantic category, facilitates the development of semantic grid map estimation, particularly LiDAR-based methods. Bieder *et al.* [BWJ+20] approach dense top-view semantic segmentation using sparse LiDAR data. Taking a single point cloud as input, they first generate a multi-layer grid map representation, which includes multiple maps representing intensity, as well as minimum and maximum detected heights. They also calculate an observability feature and the minimum observed height using ray-casting, and use them as additional layers. Then, the grid map representation is further processed by a 2D semantic segmentation network to predict a dense semantic grid map. To train the network, point-wise semantic labels in each LiDAR scan are transformed into a grid map representation. Considering the sparsity of the single-scan ground-truth map, multiple consecutive LiDAR scans are aggregated to obtain dense ground-truth maps. The proposed method achieves promising segmentation performance, especially on static objects.

In a follow-up work, Bieder *et al.* [BWRS21] fuse sequential information from multiple consecutive LiDAR measurements to improve the grid map-based semantic grid map estimation method [BWJ+20]. Although both works demonstrate good overall performance, the hand-crafted grid map feature extraction can result in a potential loss of information.

2.3.4 Evaluation Metrics

The Intersection over Union (IoU) [EVGW+10], also known as the Jaccard index, is the most commonly used metric for evaluating semantic segmentation tasks. It can be applied across tasks involving either image pixels or LiDAR points. For each class c, it is calculated by:

$$\mathrm{IoU}_c = \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c + \mathrm{FN}_c}, \tag{2.10}$$

where $\mathrm{TP}_c, \mathrm{FP}_c$, and FN_c correspond to the numbers of true positive, false positive, and false negative predicted pixels of class c, respectively.

The mean IoU (mIoU) is the average of the IoUs across all classes. It is often used to assess the overall semantic segmentation performance and is computed as follows:

$$\mathrm{mIoU} = \frac{1}{|C|} \sum_{c \in C} \mathrm{IoU}_c, \tag{2.11}$$

in which C denotes the set of all classes.

2.4 3D Object Detection

As one of the most important perception components in automated driving, 3D object detection aims to determine the pose, dimensions, and semantic class of relevant traffic participants in the surrounding environment. Typically, each detected object is represented by an oriented cuboid in 3D space, along with a classification score. To approach 3D object detection, LiDAR and camera data are two commonly used modalities. Figure 2.4 displays multiple 3D bounding box labels on an example image and the corresponding LiDAR point cloud from the KITTI dataset [GLU12].

This section offers an introduction of deep learning-based 3D object detection approaches. First, Section 2.4.1 focuses on LiDAR-based 3D object detection and presents several popular methods. Then, a number of camera-based 3D object detection networks are introduced in Section 2.4.2. Furthermore, Section 2.4.3 describes multi-modal sensor fusion approaches for 3D object detection. Finally, Section 2.4.4 presents evaluation metrics for 3D object detection.

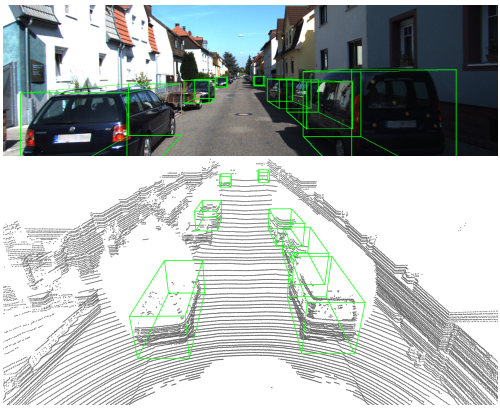


Figure 2.4: 3D bounding box labels displayed on an example image (top) and the corresponding LiDAR point cloud (bottom) from the KITTI dataset [GLU12].

2.4.1 LiDAR-based 3D Object Detection

LiDAR point clouds provide accurate spatial information, which is crucial for estimating the position and size of 3D objects. Depending on the used representation of point clouds, introduced in Section 2.2.2 and 2.2.3, LiDAR-based 3D object detection networks can be divided into three main groups: voxel-based, projection-based, and point-based.

Voxel-based Methods

Voxel-based 3D object detection methods first discretize unstructured and sparse 3D point clouds into a regular 3D voxel grid, and then perform object detection based on it. Early works like Vote3D [WP15] applies a 3D sliding window approach with Support Vector Machine (SVM) classification on the generated voxel grid, while Vote3Deep [ERW+17] employs 3D sparse convolutional layers within a binary classifier to predict object detection scores.

Li [Li17] proposes a 3D FCN for end-to-end 3D object detection by extending a 2D FCN to operate on 3D voxel grids, with each voxel represented by bi-

nary occupancy. Instead of the occupancy feature, VoxelNet [ZT18] proposes a voxel feature encoding (VFE) layer to learn a unified feature representation from points within each 3D voxel. The VFE layer uses a simplified PointNet [QSMG17] to extract features for each point in a voxel, then aggregates these point-wise features through an element-wise max operation before concatenating them back to the original point-wise features. VoxelNet adopts 3D convolutions to aggregate local voxel features and a Region Proposal Network (RPN) for final detection. However, 3D convolutions lead to a significant computational overhead and large inference times.

Following VoxelNet, SECOND [YML18] employs stacked VFE layers to extract voxel-wise features for each voxel in the 3D grid. For faster inference, it leverages 3D spatially sparse convolutions to process the obtained voxel features, and converts 3D data into image-like 2D feature maps. Then, an RPN which is similar to the single shot multibox detector (SSD) architecture [LAE+16] processes the 2D feature maps and outputs predictions of classification, regression offsets, and orientation direction. Compared with VoxelNet, SECOND achieves better detection performance at a significantly reduced runtime.

PointPillars In order to remove 3D convolutions completely, Lang *et al.* propose PointPillars [LVC+19] to discretize point clouds into a number of vertical columns called pillars that are arrayed in a BEV perspective. A pillar can be considered as a special type of voxel that has no bounds along the Z direction. By eliminating 3D convolutions and using novel feature extraction, PointPillars achieves a reduced inference time and outperforms previous detectors such as SECOND [YML18].

The network architecture of PointPillars is depicted in Figure 2.5. It mainly consists of the following four modules: pre-processing, feature extraction, backbone and detection head, and post-processing.

The pre-processing and feature extraction blocks constitute the pillar feature network, which is described in Section 2.2.4. The PFN learns high-level features from the input point cloud and further forms them in a 2D top-view representation.

The backbone network is a common SSD-like 2D CNN [LAE+16], consisting of a top-down module that gradually learns higher semantic information and

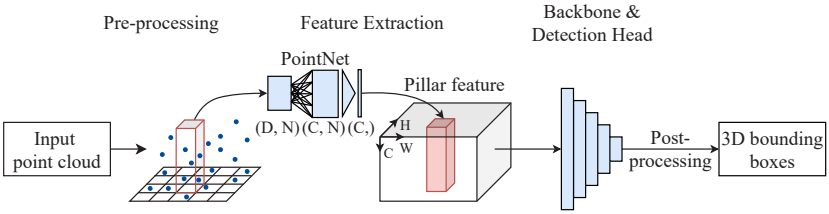


Figure 2.5: Network architecture of PointPillars. Adapted from Figure 2 in [SFH+21] ©IEEE.

a second module that upsamples and concatenates the features. The detection head processes the concatenated feature maps for anchor classification, box offsets regression and direction regression. To get the final 3D bounding boxes, the predicted box offsets and directions are decoded along with the anchor information. Then, Non-Maximum Suppression (NMS) is applied to select the best predictions from overlapping boxes.

While many aforementioned methods, such as PointPillars, rely on pre-defined anchor boxes, AFDet [GDH+20] and CenterPoint [YZK21] propose anchor-free methods for 3D object detection. Both methods design a center-based head to find centers of objects in the top-view feature maps obtained using the feature encoder in PointPilalrs [LVC+19] or VoxelNet [ZT18]. Moreover, additional heads are used to regress other object attributes such as 3D size and orientation for each detected center. Both AFDet and CenterPoint demonstrate strong performance on public benchmarks [CBL+20, SKD+20].

Projection-based Methods

Projection-based methods project point clouds into a perspective view and obtain an 2D image-like representation, which is further processed using standard 2D CNNs to detect objects. As introduced in Section 2.2.2, commonly used views for 3D object detection include the spherical view and the BEV.

As a pioneering work that performs 3D object detection in the range view, VeloFCN [LZX16] maps point clouds to a spherical representation that encodes distance and height information of each point, and applies an FCN to predict 3D bounding boxes. To address the scale variation issue, RangeDet [FXW+21]

proposes a range conditioned pyramid technique and achieves performance improvements, especially for distant or small objects. Nevertheless, the overall performance of range view-based methods still lags behind BEV-based methods in the object detection task.

As described in Section 2.2.2, the BEV representation is advantageous for 3D object detection, because it avoids occlusion and preserves each object's size, orientation, and position. Thus, it simplifies localization and regression tasks, and has been widely used in LiDAR-based object detection. For instance, PIXOR [YLU18] discretizes point clouds into a 2D grid and encodes occupancy and reflectance per cell. To preserve more detailed height information, the point cloud is divided equally into several horizontal slices when encoding the occupancy information. Thus, the final BEV representation contains multiple occupancy maps and a reflectance map, upon which a proposal-free, single-stage CNN detector is applied to estimate 3D objects.

Complex-YOLO [SMAG18] follows MV3D [CMW+17] and adopts BEV maps encoded with height, density, and intensity information. To predict 3D bounding boxes, it extends YOLOv2 [RF17] with an Euler regression module that estimates orientations with the aid of complex numbers. The model achieves good detection performance with an impressively high inference speed.

Point-based Methods

Point-based methods directly learn features from raw point clouds without any quantization to retain as much information as possible. To this end, Shi *et al.* propose a two stage Point-RCNN model [SWL19]. The first stage generates 3D object proposals by utilizing PointNet++ [QSMG17, QYSG17] to learn semantic cues and segment the input point cloud into foreground and background points. The second stage refines each proposal in the canonical coordinate by combining global semantic local spatial features.

To balance accuracy and computational efficiency, 3DSSD [YSLJ20] presents a single-stage 3D object detector. It removes the feature propagation layers and the refinement stage, commonly used in point-based methods, to reduce runtime. 3DSSD proposes a novel sampling strategy that considers spatial and feature distances during set abstraction, enhancing representative points sampling and reducing the impact of removing the feature propagation layer.

2.4.2 Camera-based 3D Object Detection

Camera images provide rich contextual and semantic information, which is crucial for the detection and classification of objects. Nevertheless, it is challenging to perform monocular 3D object detection because estimating the depth of objects from a monocular image is an inherently ill-posed problem. Camera-based 3D object detection methods can be categorized in three main groups.

One group of works builds upon the advances in 2D object detection and utilizes 2D image features to directly predict 3D bounding boxes. Early works like Mono3D [CKZ+16] first scores 3D object proposals and projects top candidates onto the image plane. These candidates are further refined by a modified Fast R-CNN [Gir15] to obtain the final 3D object detection results. Deep3DBox [MAFK16] extends a standard 2D object detector with 3D object orientation and size regression. It also employs a MultiBin method to discretize and regress the yaw angle. More recently, CenterNet [ZWK19] and FCOS3D [WZPL21] extend mature one-stage anchor-free 2D object detectors to detect 3D objects. Specifically, they modify the detection head to predict attributes of 3D bounding boxes, such as 3D center projected onto the image, depth, 3D dimension, and orientation.

The second group of research uses a pseudo-LiDAR representation for 3D object detection [WCG+19, XC18, WK19]. To this end, visual depth estimation is first performed, followed by back-projection of each 2D pixel to 3D space using the predicted depth to generate a pseudo 3D point cloud. Such a representation can be further processed by any LiDAR-based 3D object detector. Wang *et al.* propose one of the first pseudo-LiDAR methods and achieve promising overall performance [WCG+19] . However, its detection performance for distant object is unsatisfactory, because the error of depth estimation grows significantly at long distances. Observing this issue, Pseudo-LiDAR++ [YWC+20] proposes a depth-propagation algorithm that uses sparse but accurate real LiDAR measurements as guidance to correct and debias depth estimates.

The third group of monocular 3D object detectors transforms image-view features into the BEV and subsequently performs object detection in 3D space. OFTNet [RKC19] introduces orthographic feature transform (OFT) to map perspective image-based features into the orthographic BEV. To this end, OFTNet projects predefined 3D voxels onto the image and obtains voxel-wise features

by accumulating image features within their 2D projection area. To reduce memory and computational overheads, it performs weighted summation of voxel-wise features vertically, producing 2D BEV feature maps processed by a detection head for final prediction. OFTNet encounters a bottleneck as an image pixel contributes the same features to all voxels along a ray, regardless of the object's actual depth at that pixel. To address this, Lift-Splat-Shoot (LSS) [PF20] predicts depth distribution per image pixel and lifts 2D image-view latent features to 3D space accordingly.

Very recently, BEVDet [HHZD21] employs LSS [PF20] as a view transformer to transform image-view features to the BEV, which are then processed by a ResNet [HZRS16]-based encoder. Finally, the same center-based detection head in CenterPoint [YZK21] is applied to detect 3D objects. BEVDet shows impressive detection performance, but with a low inference speed. BEVDepth [LGY+23] improves depth estimation, thereby achieving a remarkable performance improvement in 3D object detection.

Instead of performing LSS-based view transformation, BEVFormer [LWL+22] presents a transformer [VSP+17]-based encoder to generate BEV representations. It uses learnable grid-shaped BEV queries along with a spatial cross-attention module to query and aggregate spatial features from multiple camera images. Besides, a temporal self-attention layer is designed to extract temporal information from history BEV features. The resulting BEV features are then input into a 3D detection head based on Deformable DETR [ZSL+21], which predicts 3D bounding boxes and their corresponding velocities.

2.4.3 LiDAR-camera Fusion-based 3D Object Detection

LiDAR-camera fusion approaches aim to fuse LiDAR and camera modalities using deep learning in order to learn and exploit their complementary properties, mentioned in Section 2.4.1 and 2.4.2.

As one of the first works, MV3D [CMW+17] proposes a two-stage multi-modal 3D object detector based on object-centric fusion. It contains a 3D proposal network which generates 3D candidate boxes using a BEV representation of point clouds, and a region-based fusion network that projects 3D proposals to multiple views (image view, LiDAR BEV, and LiDAR front view) and fuses view-specific Region of Interest (RoI) features to produce refined detection

results. In addition to LiDAR BEV features, AVOD [KML+18] incorporates image features in the 3D proposal network to improve the recall of proposal generation. It removes the LiDAR front view input and only considers the other two views. By design, object-centric fusion requires a two-stage network architecture, which is challenging to achieve real-time inference.

Frustum-PointNet [QLW+18] first performs image 2D object detection to generate 2D region proposals, which are then back-projected to 3D space to form 3D frustums. Then, Frustum-PointNet applies PointNet [QSMG17] to perform 3D object detection on the point cloud in each frustum. IPOD [YSL+18] uses the output of image semantic segmentation, rather than 2D object detection, to generate 3D proposals, achieving a higher recall. The overall performance of this fusion paradigm depends heavily on 2D image perception tasks, because unrecognized objects in images cannot be recovered in later stages.

Based on VoxelNet [ZT18], Multimodal VoxelNet (MVX-Net) [SZT19] is proposed. In its PointFusion variant, MVX-Net projects LiDAR points onto the image plane and extracts image features from a pre-trained 2D detector for each point. The extracted image features are concatenated with corresponding point features and processed jointly by the VoxelNet architecture. Similarly, PointPainting [VLHB20] uses a stand-alone image semantic segmentation network and augments LiDAR points with its compact segmentation output. The augmented points can be further fed into any LiDAR-based 3D object detector such as PointPillars [LVC+19]. This fusion strategy is simple but effective. However, the fusion is limited to the early stage, which may not be optimal.

Another family of methods performs view transformation of features, so that multimodal features are represented in the same view. Considering the advantages mentioned in Section 2.2.2, the BEV is commonly chosen for 3D object detection. Yang [Yan20] applies OFT [LVC+19] on the output of an image semantic segmentation network to transform image semantic features to the BEV, which are fused with LiDAR BEV features learned by a PFN, introduced in Section 2.2.4. Benefiting from the recent advancement of 3D object detectors based on image view transformation, BEVFusion-PKU [LXY+22] and BEVFusion-MIT [LTA+23] adopt LSS [PF20] to transform multi-view image features to 3D and obtain camera BEV features. Due to the high computational overhead of the image view transformation module, real-time deployment of the above methods on embedded automated driving system is still very challenging.

2.4.4 Evaluation Metrics

This section introduces the evaluation metrics for object detection and related basics.

Intersection over Union (IoU)

In the task of object detection, IoU is a common metric to quantify the localization accuracy by measuring the overlap between the predicted and ground-truth boxes of objects. Given two bounding boxes B_1 and B_2, their IoU score is defined as the area (2D) or volume (3D) of their intersection divided by that of their union [EVGW+10]:

$$\mathrm{IoU}(B_1, B_2) = \frac{\mathrm{area}(B_1 \cap B_2)}{\mathrm{area}(B_1 \cup B_2)}. \tag{2.12}$$

An IoU score ranges between 0 and 1, with a higher score suggesting a more precise bounding box prediction.

Precision and Recall

Precision and recall are commonly used evaluation metrics in object detection. When evaluating the performance of an object detector, only detections with a classification score above a certain threshold are considered. A detection is considered a True Positive (TP) if its IoU score relative to a ground truth bounding box is greater than a IoU threshold. Otherwise, it is considered a False Positive (FP). In addition, False Negatives (FN) refers to ground truth bounding boxes that are either missed or not associated with any prediction due to low IoU scores.

Precision is the ratio between the number of TP and the total number of positive predictions, and recall is the ratio between the number of TP and the number of ground truth objects:

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \quad \mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{2.13}$$

In other words, precision measures the proportion of positive predictions which are actually correct, reflecting how precise the detector is when it claims a prediction is positive. Recall indicates the proportion of ground truth objects that are correctly detected, showing how well the detector predicts the actual objects.

Precision-recall Curve

In object detection, the numbers of TP, FP, and FN, as well as the resulting precision and recall, are largely dependent on the confidence threshold. When decreasing the threshold, the model retrieves more ground truth objects, thereby increasing recall, but may also include more FP, which decreases precision. To measure the performance independent of the confidence threshold, precision and recall values for different thresholds are calculated and plotted. This plot is called the precision-recall curve, showing the trade-off between precision and recall at different thresholds.

Average Precision

In object detection, Average Precision (AP) is a popular metric. It summarizes the aforementioned precision-recall curve into a scalar value, representing the performance of an object detector independent of the confidence threshold. AP can be obtained by computing the area under the precision-recall curve. In practice, it is often approximated by averaging precision values across multiple recall levels.

In the KITTI object detection benchmark [GLU12], an interpolated AP variant is used, which was originally proposed in [SM86] and used in the PASCAL VOC challenges [EVGW+10] between 2007 and 2010. For a single class, it is calculated as follows:

$$\mathrm{AP} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} p_{\mathrm{interp}}(r), \tag{2.14}$$

where r is a certain recall value belonging to a set of predefined recall levels $\mathcal{R}$. $p_{\text{interp}}(r)$ denotes the interpolated precision at recall r and is determined by taking the highest precision when recall is greater than or equal to r:

$$p_{\text{interp}}(r) = \max_{r':r' \geq r} p(r'). \tag{2.15}$$

From 2019, KITTI follows [SBP+19] and uses 40 recall levels $\mathcal{R} = \{1/40, 2/40, 3/40, \ldots, 1\}$ instead of the 11 uniformly sampled recall levels $\mathcal{R} = \{0, 0.1, 0.2, \ldots, 1\}$ proposed in the Pascal VOC benchmark [EVGW+10]. In the case of 11 recall points, where recall begins at 0, even a single true positive prediction yields 100% precision in the lowest recall bin, thereby contributing a score of $1/11$ to the final AP score. The move to 40 recall points primarily eliminates the potential AP bias caused by this.

Object detection datasets often contain multiple classes of objects. To assess the overall performance of an object detector, mean average precision (mAP) is commonly used. It is defined as the average of AP values across all classes:

$$\text{mAP} = \frac{1}{|C|} \sum_{c \in C} \text{AP}_c, \tag{2.16}$$

where AP_c denotes the AP for class c and C represents the set of all classes.

2.5 Datasets

In recent years, the release of numerous public datasets featuring real-world driving scenarios has substantially accelerated the development of automated driving algorithms. This section introduces the datasets used in this thesis, namely KITTI [GLU12] and nuScenes [CBL+20] for 3D object detection and SemanticKITTI [BGM+19] for semantic grid map estimation.

2.5.1 KITTI

The KITTI dataset [GLU12], which was proposed in 2012, is a pioneering dataset in the field of automated driving. The data was captured in Karlsruhe,

Germany using an autonomous driving platform, which was equipped with various sensors [GLSU13], including:

- 1 × Velodyne HDL-64E LiDAR, 10 Hz, 64 beams
- 2 × PointGray Flea2 color cameras, 1.4 Megapixels
- 2 × PointGray Flea2 grayscale cameras, 1.4 Megapixels
- 1 × OXTS RT3003 inertial and GPS navigation system, 6 axis, 100 Hz.

KITTI provides benchmarks for tasks such as object detection, image semantic segmentation, odometry, and object tracking. The KITTI object detection dataset [GLU12] provides 7481 annotated frames for training and 7518 frames for testing, with each frame containing cropped camera images of 1242 × 375 pixels and a corresponding LiDAR point cloud. Within the camera field of view, objects belonging to 8 classes such as Car, Pedestrian, and Cyclist are annotated with 2D and 3D bounding boxes. In addition, each object's occlusion and truncation levels are specified. In experimental evaluations, the results are reported in three difficulty levels, namely Easy, Moderate, and Hard, according to each object's occlusion level, maximal truncation, and height of the 2D box in the image. Note that the online KITTI object detection benchmark only supports evaluation of the Car, Pedestrian, and Cyclist classes.

2.5.2 nuScenes

The nuScenes dataset [CBL+20] is a large-scale, comprehensive autonomous driving dataset. It contains 1000 driving sequences, collected in Boston and Singapore, which are subsequently divided into 700 sequences for training, 150 for validation, and 150 for testing. Each sequence is approximately 20 s long. The vehicle platform for data collection contains the following sensors:

- 1 × Velodyne HDL-32E LiDAR, 20 Hz, 32 beams
- 6 × Basler color cameras, 12 Hz, 1600 × 1200 pixels
- 5 × Continental long range RADAR, 13 Hz
- 1 × IMU and GPS system.

With the above sensor setup, nuScenes offers 360° sensor coverage of the surrounding environment. Note that the size of the provided camera images in nuScenes is 1600×900 pixels after cropping the ROI from the original image.

In the selected 1000 scenes, nuScenes samples keyframes at 2Hz and provides 3D bounding box annotations for 23 object classes in every keyframe. This yields a total of 40000 annotated keyframes available for training, validation, and testing. In the nuScenes object detection benchmark, these initial classes are further grouped into 10 classes: Car, Truck, Bus, Trailer, Construction Vehicle, Pedestrian, Motorcycle, Bicycle, Traffic Cone, and Barrier. The main metrics for performance evaluation are mAP and the nuScenes detection score (NDS). When calculating the AP, instead of using the IoU, the BEV Euclidean distance is considered for matching bounding boxes, where four different distance thresholds of $\mathcal{D} = \{0.5, 1, 2, 4\}$ meters are used. Finally, the mAP is computed by averaging the APs over all classes and matching thresholds, giving:

$$\text{mAP} = \frac{1}{|C|\,|\mathcal{D}|} \sum_{c \in C} \sum_{\text{d} \in \mathcal{D}} \text{AP}_{c,d}, \tag{2.17}$$

where C denotes the set of classes and $\mathcal{D}$ is the set of distance thresholds. NDS is a weighted average of mAP and five other metrics $\mathcal{E} = \{\text{E}_\text{l}, \text{E}_\text{s}, \text{E}_\text{o}, \text{E}_\text{v}, \text{E}_\text{a}\}$ calculated for TP detections that indicate the prediction errors for box location, scale, orientation, velocity, and attributes, respectively. It can be calculated by:

$$\text{NDS} = \frac{1}{10}(5\,\text{mAP} + \sum_{\text{E} \in \mathcal{E}} (1 - \min(1, \text{E}))). \tag{2.18}$$

2.5.3 SemanticKITTI

The SemanticKITTI dataset [BGM+19] is a large dataset for LiDAR-based semantic segmentation. It contains all 22 sequences of the KITTI odometry dataset [GLU12], where sequences 00 – 10 are used for training and evaluation, and sequences 11 – 21 for testing. In each sequence, SemanticKITTI provides 360° LiDAR scans and annotates each point with a semantic label out of 28 classes. Additionally, it contains accurate pose information for each LiDAR point cloud.

Since the ground truth labels for the testing sequences are not publicly available, it is common practice to use sequences 00 – 07 and 09 – 10 for training, and sequence 08 for validation, as suggested in [BGM+19]. With this split, the training and validation set consist of 19130 and 4071 LiDAR scans, respectively. In addition, it is recommended to use 19 of the initial 28 classes for training [BGM+19]. The reduction of classes is achieved by removing 3 classes that have only a few points and combining 3 moving classes with the corresponding non-moving class.

3 Semantic Grid Map Estimation

In this chapter, a novel semantic grid map estimation approach called PillarSegNet is presented. PillarSegNet takes a sparse single-sweep LiDAR point cloud as input and predicts a dense top-view semantic grid map. Firstly, Section 3.1 introduces the motivation for proposing PillarSegNet and its general concept. Afterwards, the details of PillarSegNet are introduced in Section 3.2. Finally, Section 3.3 presents experimental evaluations conducted on the SemanticKITTI dataset, demonstrating the effectiveness of the proposed approach.

This chapter is based on own previous publication [FPH+21].

3.1 Introduction

Understanding the surroundings perceived by multiple onboard sensors is crucial in many autonomous systems such as automated vehicles. As one of the key tasks in scene understanding, semantic segmentation associates each pixel of an image or point in a LiDAR scan with a semantic class. In contrast to conventional camera images, LiDAR point clouds provide more precise distance measurements of the 3D world and preserve the geometric information of objects, empowering a better understanding of the 3D surroundings.

In this chapter, the surroundings, particularly the static environment, is modeled by means of a semantic grid map generated from LiDAR data. As an advancement of conventional occupancy grid maps, a semantic grid map can be advantageous in automated driving functions for further distinguishing free-space areas such as road, sidewalk, or terrain, and occupied areas such as parked vehicles, buildings, or vegetation. Besides, when compared to sparse point-wise semantic segmentation of the 3D LiDAR point cloud, a dense semantic grid map representation can be valuable for subsequent processing in automated vehicles, given its ability to interpret areas without LiDAR mea-

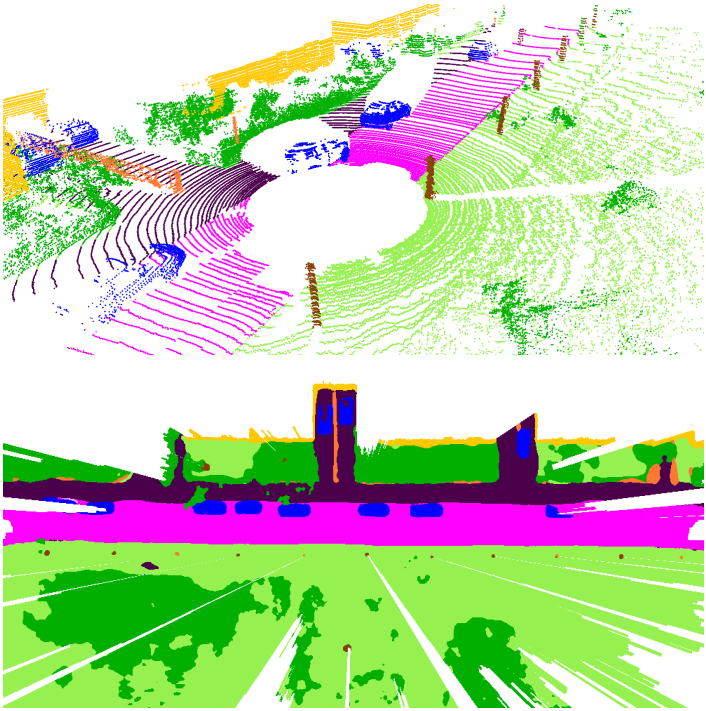


Figure 3.1: A semantically annotated 3D point cloud (top) and its corresponding top-view semantic grid map (bottom). The pixel colors represent various object categories. The unobservable areas in the semantic grid map are filtered out according to observability of LiDAR measurements. In contrast to the sparse point-wise segmentation, a dense semantic grid map representation is advantageous in interpreting areas without LiDAR measurements. The color scheme representing different object classes can be found in Table 3.2. Modified from Figure 1 in [FPH+21] ©IEEE.

surements. Figure 3.1 depicts a semantically annotated 3D point cloud and a corresponding top-view semantic grid map, in which the pixel colors represent various object categories.

To approach dense semantic grid map estimation from sparse LiDAR data, a state-of-the-art grid map method [BWJ+20] transforms the sparse LiDAR point cloud into a multi-layer grid map representation and then applies semantic segmentation to it. However, the hand-crafted grid map feature extraction can result in a potential loss of information. To overcome this challenge, a novel end-to-end approach named PillarSegNet is proposed, which uses a simplified PointNet [QSMG17] to learn features directly from the input point cloud. PillarSegNet takes a sparse single sweep LiDAR point cloud as input and subdivides it into a set of pillars on the XY plane. The raw point data in the generated pillars is then fed into a Pillar Feature Network (PFN) to extract pillar-wise features, which are scattered back onto the top view to form 2D feature maps. The feature maps are further processed by a modified U-Net [RFB15] to predict a dense semantic grid map.

To demonstrate the effectiveness of PillarSegNet, experimental evaluations are conducted on the SemanticKITTI [BGM+19] dataset. To train and evaluate PillarSegNet, point-wise semantic labels in a single LiDAR sweep are transformed into sparse ground truth represented by a 2D grid map. Furthermore, multiple neighboring labelled scans are accumulated to obtain dense ground truth. The dense label enables a proper evaluation of the dense prediction produced by PillarSegNet. In addition to the learned pillar features, the benefit of aggregating different occupancy information obtained via ray-casting is investigated. Moreover, a comprehensive analysis of the impact of various augmentation techniques on the semantic segmentation performance is performed. Finally, both quantitative and qualitative experimental results are presented and discussed.

3.2 PillarSegNet

The architecture of the proposed semantic grid map estimation approach PillarSegNet is depicted in Figure 3.2. PillarSegNet takes a sparse single-sweep 3D point cloud as input and predicts a dense semantic grid map in the top view. It consists of two main blocks, a point cloud feature encoding module

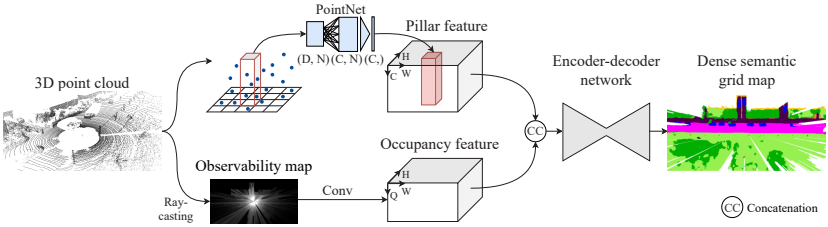


Figure 3.2: Architecture of the proposed PillarSegNet. Given a sparse single-sweep 3D point cloud, PillarSegNet first encodes pillar features and optional occupancy features in two parallel streams. The pillar features are encoded using a pillar feature network, whereas the occupancy features are encoded from an observability map as a result of model-based ray-casting. Then, an encoder-decoder network is used to predict a dense semantic grid map from the aggregated features. Note that the depicted prediction is obtained by additionally filtering the final output grid map using the observability map to exclude occluded areas. The color scheme used in the semantic grid map is detailed in Table 3.2. Adapted from Figure 2 in [FPH+21] ©IEEE.

that leverages parallel pillar and occupancy networks, and a dense semantic segmentation module built upon an encoder-decoder network. The details of each block will be presented in the following sections.

3.2.1 Point Cloud Feature Encoding

In the SemanticKITTI dataset, each LiDAR point in the raw point cloud is represented by (x, y, z, r), where x, y, and z are the 3D coordinates and r indicates the reflectance. Two kinds of feature encodings obtained from the point cloud are considered in this work: pillar features and occupancy features.

Pillar Feature Encoding The input LiDAR point cloud is processed through a pillar feature network, as detailed in Section 2.2.4, to learn pillar features in a top-view representation of dimensions (W, H, C). Here, W and H represent the width and height of the BEV grid along the X and Y axes, respectively, while C denotes the dimension of features for each pillar.

Occupancy Feature Encoding During the measurement process, the detected LiDAR points are direct consequences of physical ray-casting. When representing the LiDAR points using pillar features, one fundamentally neglects the hidden model information of observability, including information on free space and occupied areas [HZHR20]. However, the observability information may be beneficial for the dense top-view segmentation. Hence, occupancy features are further extracted and considered as an additional input stream.

In this work, the grid map framework in [BWJ+20] is adopted to generate a single-channel observability map, which indicates the number of transmissions in a grid cell. The observability map is able to represent occupancy information in the 2D grid. Then, a 3×3 convolutional layer is applied on it to learn 2D occupancy features of size (W, H, Q). Considering the potential information loss in the height direction during 2D grid mapping, the incorporation of 3D occupancy information is also investigated. For that, the point cloud is discretized into a 3D voxel grid, where each voxel is assigned a singular value that represents the occupancy probability obtained through ray-casting and voxel traversal [AW87]. As a result, a multi-channel map representing 3D occupancy information is obtained and used to replace the observability map. Similarly, the voxel map is convolved by 3×3 filters to create 3D occupancy features of size (W, H, Q).

Feature Aggregation After obtaining pillar features and occupancy features from the above two input streams, the concatenation operation is adopted to aggregate them, which is simple yet effective for fusing different features [FCH+20]. After concatenation, aggregated point cloud features of size $(W, H, C + Q)$ are obtained.

3.2.2 Dense Semantic Segmentation

To obtain the dense semantic segmentation of the input point cloud features, an encoder-decoder network is applied, followed by a semantic segmentation head.

Encoder–Decoder Network In this work, a modified U-Net [RFB15], which has an encoder-decoder architecture, is used for feature extraction. In the U-Net architecture, the encoder module progressively downsamples the spatial resolution of feature maps and captures higher-level semantic information, while the decoder module gradually upsamples feature maps and recovers the spatial information. U-Net also introduces skip connections between the aforementioned modules to combine semantic and spatial information. As a modification to the vanilla U-Net, its input convolution block is removed, since the aggregated input point cloud features have already been transformed into a high-dimensional feature space. This reduces computational and memory overhead with a negligible loss of performance.

Segmentation Head After the aggregated features pass through the encoder-decoder network, a set of 1×1 convolutions is then performed in the segmentation head. Finally, for each grid cell, the segmentation head predicts logits, which are the raw unnormalized scores indicating the likelihood of each class in the training data. During inference, the softmax function is applied to the unbounded logits to provide normalized softmax probabilities.

Loss Function The SemanticKITTI [BGM+19] dataset has a severe class imbalance issue. To tackle it, the weighted cross-entropy loss is adopted to optimize the model. The weights for the classes are determined by considering the class distribution of the ground-truth labels. The weighted cross-entropy loss $\mathcal{L}_{\text{seg}}$ for a single image can be calculated as:

$$\mathcal{L}_{\text{seg}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} \lambda_c \cdot y_{i,c} \cdot \log(\hat{y}_{i,c}), \tag{3.1}$$

where N is the total number of labeled pixels in the ground-truth map and C is the number of classes. $y_{i,c}$ is a binary indicator (0 or 1) if class label c is the correct classification for the i-th pixel, while $\hat{y}_{i,c}$ denotes the predicted probability that the i-th pixel is of class c. Besides, λ_c is the weighting coefficient for class c, used to handle class imbalance.

3.3 Experiments

In this section, experimental evaluations of PillarSegNet are provided. First, Section 3.3.1 describes the dataset used for training and evaluation as well as the preparation of the ground truth. Then, the implementation details of PillarSegNet are introduced in Section 3.3.2. Lastly, Section 3.3.3 presents the results of quantitative evaluations, followed by Section 3.3.4 which shows exemplary qualitative results.

3.3.1 Dataset Preparation

SemanticKITTI Dataset The proposed PillarSegNet is trained and evaluated on the SemanticKITTI dataset [BGM+19], which is detailed in Section 2.5.3. Following [BGM+19, BWJ+20], sequences 00 – 07 and 09 – 10 are used for training and sequence 08 for validation. To tackle the under-representation of rare classes, the 19 classes originally defined in the SemanticKITTI dataset are merged into 12 classes as in [BWJ+20]. This also ensures a fair comparison between PillarSegNet and the grid map based method [BWJ+20]. Table 3.1 shows the specific mapping. Note that the remaining SemanticKITTI classes, i.e., road, sidewalk, building, vegetation, trunk, terrain, and person, are not changed.

SemanticKITTI class	Merged class
car, truck, and other-vehicle	vehicle
motorcyclist and bicyclist	rider
bicycle and motorcycle	two-wheel
traffic-sign, pole, and fence	object
other-ground and parking	other-ground

Table 3.1: Mapping of SemanticKITTI classes to the classes defined in [BWJ+20]. The remaining SemanticKITTI classes, i.e., road, sidewalk, building, vegetation, trunk, terrain, and person, are not changed.

SemanticKITTI originally provides point-wise semantic labels, while the training of PillarSegNet requires the ground truth represented in a grid map. Thus,

it is necessary to transform the initial labels into the desired representation. In this work, two kinds of ground-truth grid map, namely sparse ground truth and dense ground truth, are considered. Next, the details of generating them will be introduced.

vehicle	person	two-wheel
rider	road	sidewalk
other-ground	building	object
vegetation	trunk	terrain

Table 3.2: Selected SemanticKITTI [BGM+19] color scheme for the remapped classes used in Figure 3.1 to Figure 3.6.

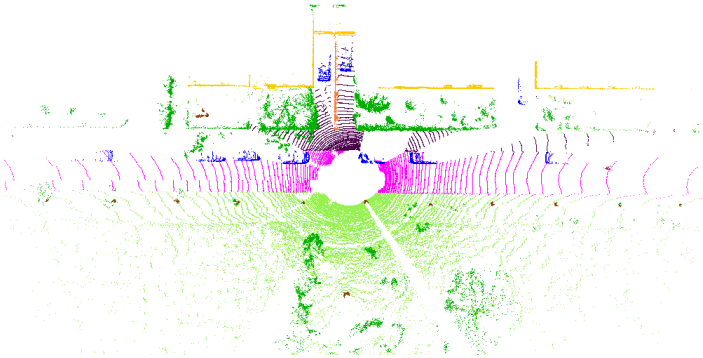
Sparse Ground Truth Generation In order to obtain the top-view sparse ground truth, a point-wise labelled point cloud is first rasterized into grid cells. For each grid cell, the number of points for each of the 12 classes is counted. The semantic class c_i for the i-th grid cell is then determined through a weighted argmax operation:

$$c_i = \underset{k \in \{0,1,\ldots,K-1\}}{\operatorname{argmax}} \left(w_k \cdot n_{i,k} \right), \tag{3.2}$$

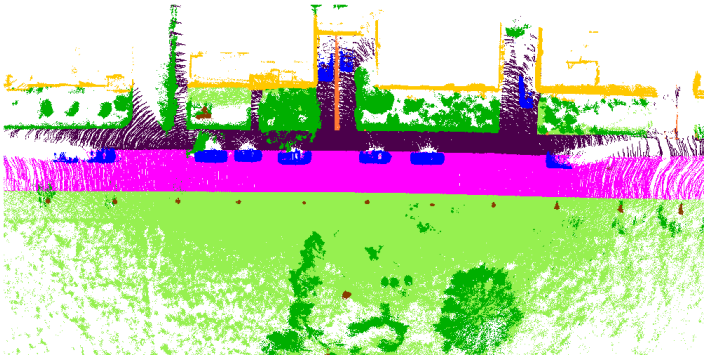
where K is the number of classes, $n_{i,k}$ denotes the number of points belonging to the class with index k inside the i-th grid cell, and w_k is the weighting factor for the k-th class. The class-specific weighting factors are chosen according to:

$$w_k = \begin{cases} 0 & \text{for unlabeled} \\ 5 & \text{for vehicle, person, rider, and two-wheel} \\ 1 & \text{for others} \end{cases}. \tag{3.3}$$

In addition, grid cells without any assigned points are marked as unlabeled, which are not considered for optimizing the model during training. An example of a sparse ground-truth map is depicted in Figure 3.3a, with the color scheme representing different object classes detailed in Table 3.2. Note that this example corresponds to the frame depicted in Figure 3.1.



(a) An exemplary sparse ground-truth map.



(b) An exemplary dense ground-truth map.

Figure 3.3: Sparse and dense ground-truth maps. To enable the training of PillarSegNet, the sparse ground truth is generated from a semantically annotated 3D point cloud, while the dense ground truth is obtained by superimposing multiple nearby labeled scans. The corresponding color scheme is defined in Table 3.2. Both ground-truth maps correspond to the frame shown in Figure 3.1. Modified from Figure 1 in [FPH+21] ©IEEE.

Dense Ground Truth Generation A semantic grid map is a dense representation of the surrounding environment. To better indicate the performance of semantic grid estimation approaches in real-world applications, it is more appropriate to evaluate them with dense ground truth. It may also be beneficial to train the proposed PillarSegNet with dense ground truth.

SemanticKITTI contains consecutive LiDAR scans with accurate pose information, which allows the generation of dense ground truth by superimposing multiple labeled scans. For each scan in a sequence, a series of adjacent point clouds are collected based on the following criteria:

$$\Delta d_{\text{LiDAR}} \leqslant 2 \cdot d_{\text{farthest point}}, \tag{3.4}$$

where Δd_{LiDAR} denotes the LiDAR sensor distance between the positions of the current and the neighboring scan, and $d_{\text{farthest point}}$ is the distance of the farthest point in the current scan. In addition, the maximum number of considered neighboring scans is set to 40. With the provided poses, the selected point clouds are then transformed to the LiDAR coordinate system of the current scan. The aggregation of multiple adjacent scans assumes that the scene is non-moving and there is an aggregation error for moving objects, resulting in a smearing of objects. Therefore, only points belonging to static objects are aggregated. For moving objects, only measurements from the current scan are used. Note that the distinction between static objects and moving objects is achieved with the help of a "moving" label in the SemanticKITTI dataset. Finally, the process used for sparse ground truth generation is adopted to create a dense ground-truth grid map. An example of a dense ground-truth map is shown in Figure 3.3b, where the color scheme representing different object classes can be found in Table 3.2.

3.3.2 Implementation Details

PillarSegNet takes a sparse single LiDAR sweep as input and outputs semantic probabilities for the 12 classes described in Section 3.3.1. The implementation of PillarSegNet is based on the OpenPCDet [Tea] codebase. For all experiments, the input point cloud is cropped at $[-50, 50] \times [-25, 25] \times [-2.5, 1.5]$ meters along the X, Y, and Z axes, respectively. When generating pillars, the pillar grid size is set to $0.1\,\text{m} \times 0.1\,\text{m}$, the number of pillars P is 30000, and

the number of points per pillar N is 20. The channels of the pillar-wise features learned by PointNet is defined as $C = 64$. The grid cells for generating the observability map and the voxels for encoding the 3D occupancy map have the same size in the X and Y axes as the pillars, and in addition the voxels have a height of 0.2 meters along the Z axis. The occupancy features after convolutions have $Q = 16$ channels.

During training, multiple data augmentation techniques, namely global flipping, global rotation, global scaling, and global translation, are applied to prevent the network from over-fitting on the training data. The details of data augmentation are described below:

- Global flipping: all LiDAR points are flipped along the X or Y axis by a 50% chance.
- Global rotation: all LiDAR points are rotated around the upright Z axis by a uniformly distributed angle between $[-\pi/4, +\pi/4]$ radians.
- Global scaling: the coordinates of all LiDAR points are multiplied by a factor randomly sampled from a uniform distribution between $[0.95, 1.05]$.
- Global translation: all LiDAR points shifted by $(\Delta x, \Delta y, \Delta z)$, where $\Delta x, \Delta y$, and Δz are independently sampled from zero-mean normal distributions:

$$\begin{aligned} \Delta x, \Delta y &\sim \mathcal{N}(0,\, \sigma_{\mathrm{x,y}}^2) \\ \Delta z &\sim \mathcal{N}(0,\, \sigma_{\mathrm{z}}^2), \end{aligned} \tag{3.5}$$

  in which the standard deviations $\sigma_{\mathrm{x,y}}$ and σ_{z} are set to 5 and 0.5 meters, respectively.

Note that the corresponding operation is applied to the ground-truth grid map for each data augmentation technique. When taking occupancy features as input, the input occupancy map is consistently augmented as the LiDAR points so that pillar and occupancy features are always aligned.

When training PillarSegNet using the sparse ground-truth grid map, the weighting coefficient λ_c in the loss function (Equation 3.1) is set to 2 for the class vehicle and 8 for the classes pedestrian, two-wheel, and rider. For training with the dense ground truth, the coefficient for the class vehicle is modified to 5.

The default coefficients for the remaining classes are set to 1. These values are determined based on class frequency statistics computed on ground truth labels, with empirical refinement. The network is trained from scratch using the Adam optimizer [KB15] with an initial learning rate of 0.001, and a weight decay of 0.01. The training lasts 30 epochs with a mini-batch size of 2.

3.3.3 Quantitative Evaluation

This section describes the quantitative evaluation of PillarSegNet on the SemanticKITTI dataset. Various experiments are designed and conducted to assess the effectiveness of the proposed PillarSegNet, explore the benefits of leveraging additional occupancy information and dense ground truth, and study the effect of different data augmentation techniques. All experiments are evaluated using the IoU and mIoU metrics, detailed in Section 2.3.4. In addition, the inference time of PillarSegNet is measured and reported.

The training of PillarSegNet is done in two modes, namely *Sparse Train* and *Dense Train*, according to which ground truth is used. The former mode considers the sparse ground truth, which is obtained from single sweep, whereas the latter one considers the dense ground truth, which is obtained by aggregating multiple adjacent sweeps. For each experiment using the *Sparse Train* mode, the evaluation is performed in two ways, which are *Sparse Eval* and *Dense Eval*, in order to achieve fair comparisons with the method in [BWJ+20]. The former uses the sparse ground truth for metrics calculation, while the latter considers the dense ground truth and additionally filters the predictions with the observability map to exclude occluded areas. The *Dense Eval* mode better indicates the performance of semantic grid map estimation in deployment. Note that the experiments in the *Dense Train* mode are only evaluated by the *Dense Eval* approach.

First, to prove the effectiveness of PillarSegNet, the variant that only considers pillar features is trained and evaluated in the (*Sparse Train*, *Sparse Eval*) and (*Sparse Train*, *Dense Eval*) modes. Afterwards, two PillarSegNet variants that take occupancy features as input are evaluated, in order to study the benefit of leveraging occupancy information for semantic grid map estimation. Moreover, further experiments are conducted to demonstrate the advantage of training PillarSegNet using dense ground truth. The corresponding evaluation results

on the SemanticKITTI validation set are summarized in Table 3.3. Finally, a quantitative comparison of PillarSegNet with state-of-the-art methods is performed, with metrics reported in Table 3.4.

In addition to the above evaluations, a thorough ablation study is conducted to investigate the effect of different data augmentation techniques in PillarSegNet. The experimental results are presented in Table 3.5. Lastly, the inference time of PillarSegNet is measured and reported.

In the following, all the aforementioned evaluation results are discussed in detail.

Effectiveness In the *Sparse Train* mode shown in Table 3.3, the proposed pillar-based method significantly outperforms the state-of-the-art grid map method [BWJ+20] by 15.3% and 4.7% mIoU in the *Sparse Eval* and *Dense Eval* modes, respectively. When observing the IoU performance on each category, PillarSegNet shows notable IoU gains for almost all classes. In particular, PillarSegNet achieves outstanding IoU improvements on vulnerable road users (VRUs), such as person, two-wheel, and rider, in the *Sparse Eval* mode. These remarkable performance boosts indicate the effectiveness of PillarSegNet and the superiority of the learned pillar features over the hand-crafted grid map representation. In spite of the significant improvements on VRUs, the results of PillarSegNet are still not satisfactory for automated driving applications. This can be complemented by 3D object detection introduced in Chapter 4, which provides more reliable perception of dynamic objects.

Occupancy Features Representation In addition to the pillar features, the role of 2D and 3D occupancy features in semantic grid map estimation is studied. As shown in Table 3.3, PillarSegNet achieves additional performance improvements in terms of mIoU when aggregating occupancy features, regardless of 2D or 3D. Besides, the Pillar + 3D Occupancy variant performs slightly better than the Pillar + 2D Occupancy variant. The improvements demonstrate that the observability information, generated via model-based ray-casting, can be successfully used to further improve the performance of semantic grid map estimation. Additionally, measurements are conducted on an AMD EPYC 7R32 CPU to determine the computation time required for generating 2D

Mode	Method	mIoU [%]	vehicle	person	two-wheel	rider	road	sidewalk	other-ground	building	object	vegetation	trunk	terrain
Sparse Train Sparse Eval	Bieder *et al.*	39.8	69.7	0.0	0.0	0.0	85.8	60.3	25.9	72.8	15.1	68.9	9.9	69.3
	Pillar	55.1	79.5	15.8	**25.8**	**51.8**	89.5	70.0	**38.9**	80.6	25.5	**72.8**	38.1	**72.7**
	Pillar + 2D Occupancy	55.3	82.7	**20.3**	24.5	51.3	90.0	71.2	36.5	81.3	**28.3**	70.4	38.5	69.0
	Pillar + 3D Occupancy	**56.2**	**83.8**	19.5	24.8	**51.8**	**90.1**	**72.3**	36.9	**81.5**	28.2	72.1	**41.7**	71.5
Sparse Train Dense Eval	Bieder *et al.*	32.8	43.3	0.0	0.0	0.0	84.3	51.4	22.9	54.7	10.8	51.0	6.3	68.6
	Pillar	37.5	45.1	0.0	0.1	3.3	82.7	57.5	29.7	64.6	14.0	**58.5**	25.5	**68.9**
	Pillar + 2D Occupancy	38.4	52.5	0.0	**0.2**	3.0	85.6	60.1	**29.8**	**65.7**	16.1	56.7	26.2	64.5
	Pillar + 3D Occupancy	**38.9**	**53.3**	0.0	0.1	**5.0**	**86.1**	**60.5**	**29.8**	65.1	**16.4**	56.7	**28.1**	66.2
Dense Train Dense Eval	Pillar	42.8	70.3	5.4	**6.0**	8.0	89.8	65.7	**34.0**	65.9	16.3	61.2	23.5	67.9
	Pillar + 2D Occupancy	44.1	72.8	7.4	4.7	**10.2**	**90.1**	**66.2**	32.4	67.8	17.4	63.1	27.6	69.2
	Pillar + 3D Occupancy	**44.6**	**73.1**	**7.8**	**6.0**	10.0	89.7	65.7	30.3	**68.3**	**18.4**	**65.0**	**30.4**	**70.8**

Table 3.3: Quantitative results on the SemanticKITTI validation set, using IoU and mIoU metrics. The best results in each mode are highlighted in bold.

and 3D occupancy features. Utilizing multithreading with 16 threads, the 2D occupancy feature generation takes 30 ms, while the 3D variant takes 80 ms.

Ground Truth Representation To investigate the effect of training PillarSegNet with different representations of ground truth, the results obtained in the (*Dense Train*, *Dense Eval*) mode are now discussed. When compared with the (*Sparse Train*, *Dense Eval*) mode, training the network with the dense ground truth leads to notable IoU improvements for all classes. For the deployment of semantic grid map estimation, it is thus advisable to leverage dense labels during training for better performance. Consistent with the previous observation, 2D and 3D occupancy information yield similar overall performance gains. Considering the increased computational overhead when generating the 3D occupancy map, it is more practical to aggregate 2D occupancy features and pillar features as a compromise between performance and computational efficiency.

Comparison with State-of-the-Art Methods PillarSegNet is further compared with recent methods that are trained and evaluated in the (*Dense Train*, *Dense Eval*) mode. GM_Temp [BWRS21] extends the baseline grid map method [BWJ+20] by aggregating temporal information from past lidar scans, while GM_RV [BLR+21] fuses features learned in complementary top-view and range-view representations. MASS [PFY+22] builds upon PillarSegNet by further incorporating multi-attention mechanisms. In addition, CVA [GLY+24] adopts a cross-view association module to combine top-view, range-view, and voxel-wise features for semantic grid map estimation. The corresponding metrics are reported in Table 3.4. Note that GM_Temp and GM_RV merge rider and two-wheel classes in their ground truth during both training and evaluation. Consequently, the results for these categories are not directly comparable and are excluded from the table to ensure a fair comparison.

When compared with recent grid map approaches employing temporal fusion (GM_Temp) or multi-view fusion (GM_RV), PillarSegNet (the Pillar + 2D Occupancy variant) achieves consistent IoU gains across nearly all classes, with substantial mIoU improvements of 13.2% and 12.6%, respectively. This demonstrates the advantages of the proposed learning-based feature extraction of PillarSegNet. In contrast to more advanced learning-based approaches such

as MASS and CVA, PillarSegNet delivers a comparable overall performance, with mIoU differences within 1.0%. Moreover, it surpasses both methods on several important classes, including vehicle, road, and sidewalk, underscoring PillarSegNet's strong performance while maintaining architectural simplicity.

Method	mIoU [%]	vehicle	person	road	sidewalk	other-ground	building	object	vegetation	trunk	terrain
PillarSegNet	51.4	**72.8**	7.4	**90.1**	**66.2**	32.4	**67.8**	17.4	63.1	27.6	69.2
GM_Temp	38.2	39.2	0.0	82.0	48.7	8.9	58.0	13.8	61.1	6.4	64.7
GM_RV	38.8	40.2	4.2	80.5	46.3	20.5	48.1	**23.1**	52.6	17.5	55.2
MASS	**52.3**	72.1	6.8	**90.1**	65.8	**37.8**	67.1	18.8	**68.1**	24.7	**71.4**
CVA	**52.3**	68.5	**22.0**	89.9	61.0	34.4	65.1	20.4	63.3	**30.4**	67.6

Table 3.4: Quantitative comparison of PillarSegNet with state-of-the-art methods on the SemanticKITTI validation set. IoU and mIoU metrics are reported, with the best results highlighted in bold.

Data Augmentation Thorough ablation experiments are performed to investigate the effect of different data augmentation techniques in PillarSegNet. The results are reported in Table 3.5. Baseline denotes the PillarSegNet variant that takes pillar features as input and it is trained on the sparse ground truth without any data augmentations. All experiments are evaluated in the *Sparse Eval* mode and the achieved mIoU performances are presented. It can be observed that global flipping and global rotation significantly boost the performance by 2.6% and 2.0% in terms of mIoU, respectively. In addition, global scaling contributes to another 0.6% improvement. In comparison with the baseline, applying the above three data augmentation techniques leads to a total performance gain of 5.2%. Unexpectedly, global translation has a negative effect on the overall performance, presumably, because it violates the condition that the ego car should always be in the grid center.

Baseline	Flipping	Rotation	Scaling	Translation	mIoU [%]
✓					50.4
✓	✓				53.0
✓	✓	✓			55.0
✓	✓	✓	✓		55.6
✓	✓	✓	✓	✓	55.1

Table 3.5: Ablation study for data augmentation techniques on the SemanticKITTI validation set. Baseline denotes the PillarSegNet variant that takes pillar features as input and it is trained on the sparse ground truth without any data augmentations.

Runtime Performance The inference time of PillarSegNet is measured on a desktop computer with an Intel i9-7920X CPU and an NVIDIA 2080 Ti GPU. The pillar-based PillarSegNet variant achieves a total runtime of 58 ms. When further incorporating 2D occupancy features, additional 10 ms and 6 ms are required for occupancy grid mapping and network inference, respectively. Note that the computation time for 2D occupancy grid mapping here is measured using GPU with parallel processing [HKOB10], which differs from the previous measurement conducted on CPU.

3.3.4 Qualitative Results

To qualitatively demonstrate the performance of PillarSegNet and its improvement over the grid map method from [BWJ+20], the prediction results in three exemplary scenes from the SemanticKITTI validation set are shown in Figures 3.4 – 3.6. Note that the studied PillarSegNet variant here only uses the pillar features and is trained using the dense labels. From top to bottom, each figure includes the input point cloud, the observability map, the prediction of the grid map method [BWJ+20], the prediction of PillarSegNet, as well as the corresponding dense ground truth map. The unobservable areas in each prediction map are filtered out according to the observability map. Additionally, ceartain key regions are highlighted on the right-hand side to facilitate easier comparisons.

As depicted in all three examples, PillarSegNet accurately segments general road scenes, including the static environment and varying traffic participants, when compared to the ground truth. This demonstrates the ability of PillarSegNet for semantic scene understanding from a single sparse LiDAR scan. In Figures 3.4 and 3.5, PillarSegNet shows more accurate prediction for vehicles when compared to the grid map method from [BWJ+20]. In particular, the pillar-based approach reconstructs their shapes very well, even with few LiDAR measurements. As shown in Figures 3.5 and 3.6, PillarSegNet provides better predictions for road and sidewalk than the grid map method. Besides, PillarSegNet successfully segments challenging riders in these two scenes. The above comparisons indicate the superiority of the learned pillar features over the hand-crafted grid map representation. Overall, the proposed PillarSegNet exhibits excellent performance in modeling the environment, especially the static environment, as shown in the quantitative results in Table 3.3 and in the comparison with state-of-the-art methods in Table 3.4. Despite notable improvements on VRUs, PillarSegNet alone is not sufficient to reliably perceive them. Thus, other methods, such as 3D object detection approaches, are needed to complement the environment model.

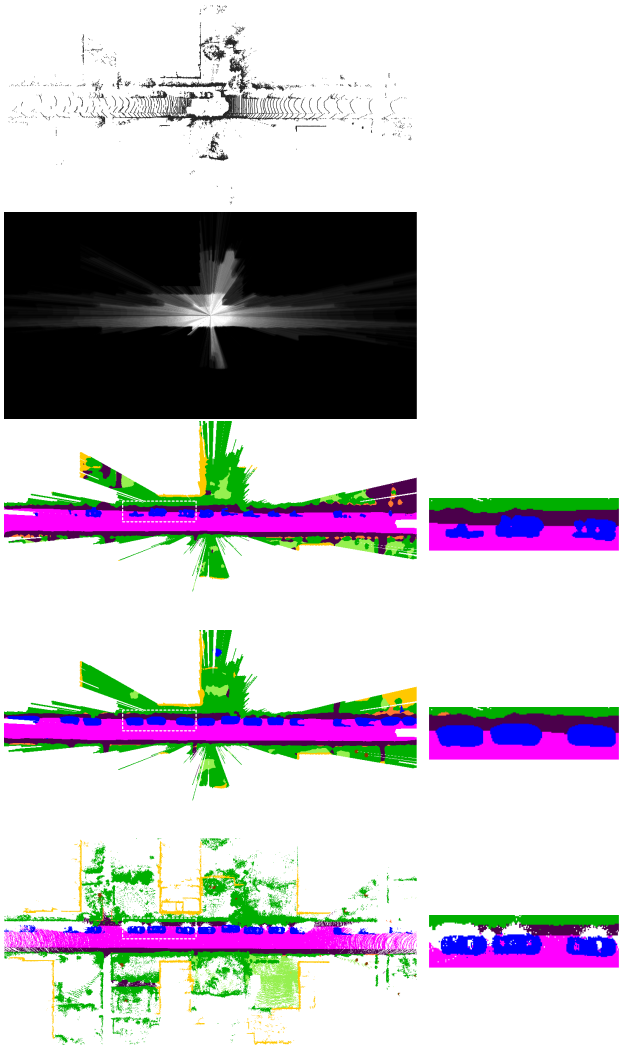


Figure 3.4: Qualitative results produced by PillarSegNet (utilizing only pillar features, trained using dense labels) and the grid map method [BWJ+20] on the SemanticKITTI validation set. From top to bottom, the input point cloud, the observability map, the prediction of [BWJ+20], the prediction of PillarSegNet, and the corresponding ground truth are depicted. The unobservable areas in each prediction map are filtered out according to the observability map. Besides, some relevant areas are highlighted on the right for easier comparison. The corresponding color scheme is defined in Table 3.2. Modified from Figure 3 in [FPH+21] ©IEEE.

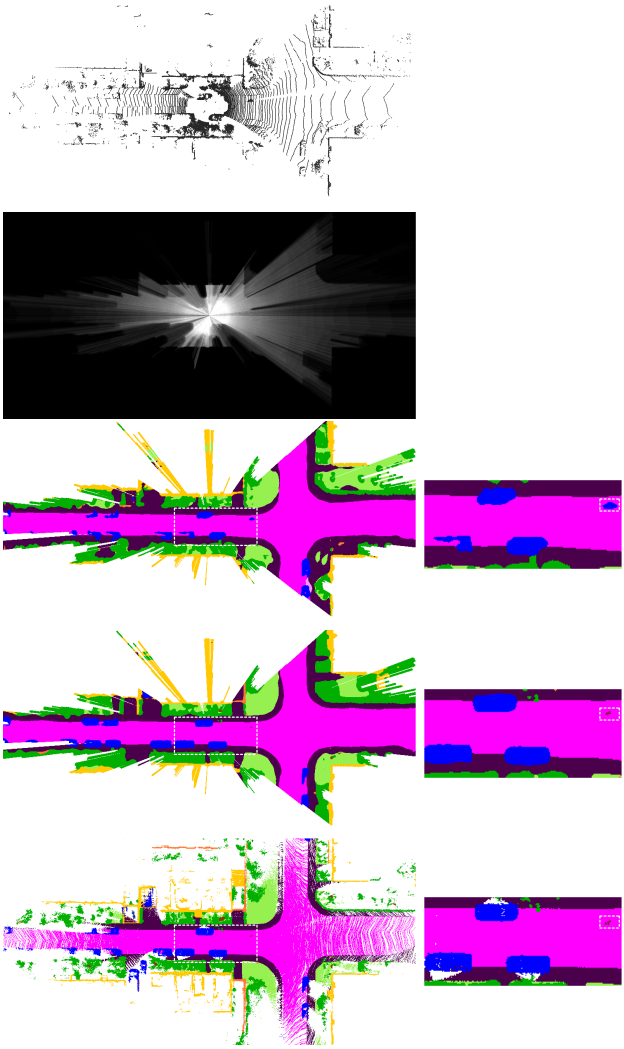


Figure 3.5: Qualitative results produced by PillarSegNet (utilizing only pillar features, trained using dense labels) and the grid map method [BWJ+20] on the SemanticKITTI validation set. From top to bottom, the input point cloud, the observability map, the prediction of [BWJ+20], the prediction of PillarSegNet, and the corresponding ground truth are depicted. The unobservable areas in each prediction map are filtered out according to the observability map. Besides, some relevant areas are highlighted on the right for easier comparison. The corresponding color scheme is defined in Table 3.2. Modified from Figure 3 in [FPH+21] ©IEEE.

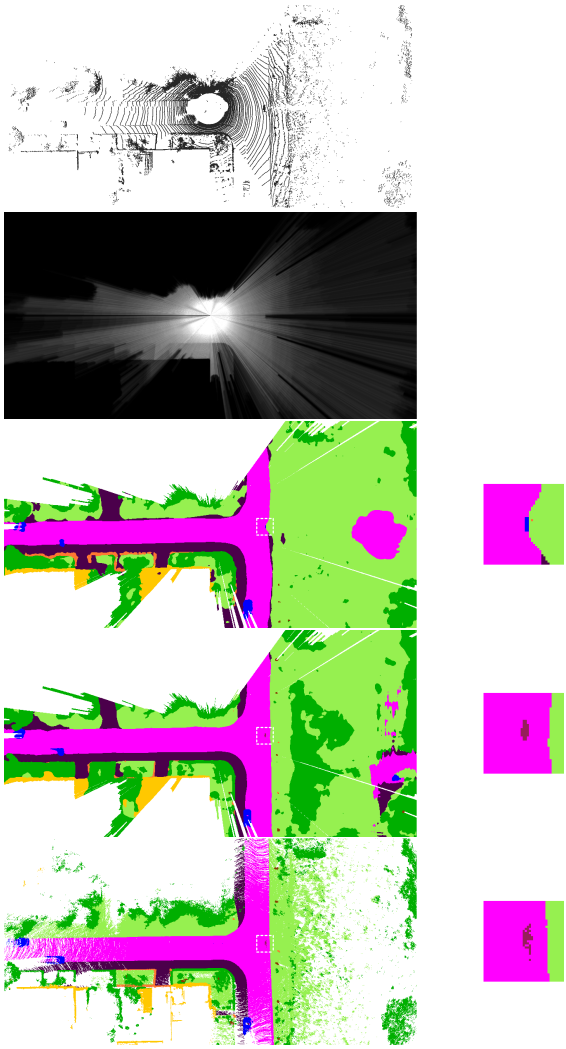


Figure 3.6: Qualitative results produced by PillarSegNet (utilizing only pillar features, trained using dense labels) and the grid map method [BWJ+20] on the SemanticKITTI validation set. From top to bottom, the input point cloud, the observability map, the prediction of [BWJ+20], the prediction of PillarSegNet, and the corresponding ground truth are depicted. The unobservable areas in each prediction map are filtered out according to the observability map. Besides, some relevant areas are highlighted on the right for easier comparison. The corresponding color scheme is defined in Table 3.2. Modified from Figure 3 in [FPH+21] ©IEEE.

4 3D Object Detection with LiDAR-camera Fusion

This chapter introduces SemanticVoxels, an approach for detecting traffic participants in the 3D environment through fusion of LiDAR and camera input data. In Section 4.1, the motivation for proposing SemanticVoxels and an overview of it are described. Then, the details of the proposed approach are presented in Section 4.2, followed by quantitative and qualitative experimental evaluations in Section 4.3. Finally, Section 4.4 demonstrates the deployment of the proposed approach on an edge AI device.

Parts of this chapter are based on own previous publications [FCH+20, SFH+21].

4.1 Introduction

In the context of automated driving, the goal of 3D object detection is to determine the pose (including position and orientation), shape as well as semantic class of relevant traffic participants. To simplify this task, the pose and shape information of an object are usually represented by an oriented 3D bounding box with known position [GLU12, CBL+20]. For automated vehicles, LiDAR and camera are two commonly used sensor modalities, which deliver complementary information: LiDAR point clouds provide accurate spatial information, while camera images are able to provide rich contextual and semantic information. In 3D object detection, LiDAR spatial information is crucial for localizing objects and estimating their shapes, while image semantic information is needed for reliably classifying objects and interpreting scenes. Thus, it is a favorable choice to leverage these complementary knowledge by fusing LiDAR and camera data, aiming to achieve better performance.

Recently, deep learning-based approaches have advanced significantly, achieving remarkable results in 3D object detection [QLW+18, CMW+17, YML18, LVC+19], which is enabled by the availability of annotated datasets with multiple calibrated and synchronized sensors [GLU12, CBL+20]. In 3D object detection, early-level data fusion using DNNs offers advantages over conventional object-level fusion, given the fact that DNNs utilize more information available in early processing stages. This is demonstrated by trends in public benchmarks [GLSU13, CBL+20]. When doing early data fusion, it is necessary that all sensor measurements are represented in a common coordinate system. In order to fuse LiDAR and camera modalities efficiently, representing both of them in the Bird's Eye View (BEV) is especially advantageous. The reasons for this are manifold. Firstly, all traffic participants move on a common ground surface, and they do not overlap in the BEV. This property avoids the occlusion issue when dealing with the range view. Besides, the scale of an object keeps consistent in the BEV regardless of varying positions, getting rid of the issue of diverse scales that often encountered in image-based object detection. Lastly, it allows a unified and modular approach for sensor fusion by representing and then fusing different sensor modalities in the BEV. In this way, it is straightforward to extend this approach when considering multiple range or image sensors as well as additional sensor modalities.

Creating a BEV representation for LiDAR data is fairly straightforward because the point cloud data is initially represented in 3D and can be mapped to the top view directly. However, this is not trivial for camera images, since retrieving 3D information from a single RGB image is an ill-posed inverse problem. In the proposed LiDAR-camera fusion approach SemanticVoxels, to encode image features in the BEV, LiDAR points in 3D are first projected onto the image plane, as in [Fei19, VLHB20]. Following that, raw point data are appended with semantic segmentation scores that have been obtained from an image-only semantic segmentation network. In this way, image semantic features originally in the image view are successfully lifted into 3D space. Then, the augmented point cloud is processed in two pathways. During spatial feature encoding, it is discretized into pillars and the Pillar Feature Network (PFN), introduced in Section 2.2.4, is adopted to learn pillar-wise spatial features from the raw point data. In semantic feature encoding, each generated pillar is further divided into voxels along the height direction, in which the appended image semantic information is encoded, yielding voxel-wise semantic features. This preserves the spatial distribution of the image information in the vertical

direction, being valuable for accurate object detection. So far, LiDAR spatial and image semantic features in the BEV have been obtained. In the end, these two kinds of complementary features are fused in a CNN backbone, followed by a detection head to predict oriented 3D bounding boxes. Note that the proposed approach differs from PointPainting [VLHB20], in which the augmented point cloud is directly fed into a LiDAR-based object detection network.

In this work, the output of a semantic segmentation network is considered as the input image feature. It provides a dense, fine-grained, yet memory-efficient representation of object classes and scene layout. Alternative image features, such as those from a 2D object detection model's backbone or its classification head, can also be considered. Backbone features offer rich semantic information but are typically high-dimensional and computationally costly, while features from the classification head provide compact, object-level semantics constrained by bounding-box supervision, which may be less representative for areas outside object contours. Semantic segmentation outputs combine the advantages of both. Moreover, semantic segmentation networks are widely used in automated driving, and significant progress is being made in that field, allowing the proposed LiDAR-camera fusion approach to benefit with minimal overhead.

In order to verify the effectiveness of SemanticVoxels, extensive evaluations are conducted using both the KITTI [GLU12] and nuScenes [CBL+20] 3D object detection datasets. Finally, SemanticVoxels is successfully deployed on an embedded platform with runtime optimization techniques and real-time inference speed is achieved. It shows reliable detection performance on unseen real-world data, despite only being trained on a dataset with a similar LiDAR, thereby demonstrating its outstanding generalization ability.

4.2 SemanticVoxels

An overview of the proposed approach is depicted in Figure 4.1. It takes an image and a 3D point cloud as inputs and estimates oriented 3D bounding boxes of relevant objects in the traffic scene. SemanticVoxels consists of five main blocks: image semantic segmentation, point cloud augmentation, feature encoding, feature fusion and backbone, and detection head. Details of each block are presented in the following subsections.

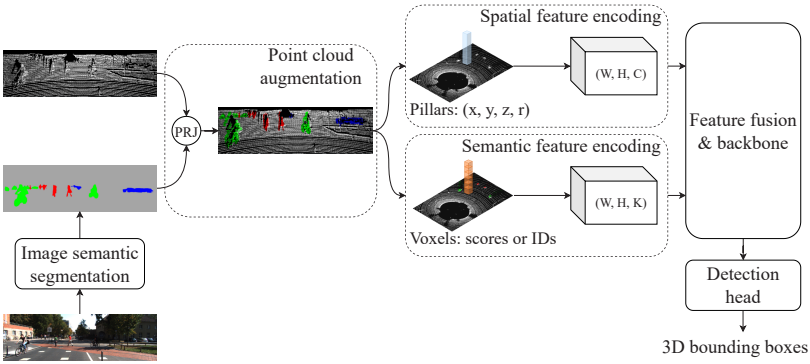


Figure 4.1: Architecture of the proposed LiDAR-camera fusion method SemanticVoxels. It can be divided into five parts: image semantic segmentation, point cloud augmentation, multi-modal feature encoding, feature fusion and backbone, as well as detection head. In the first part, semantic segmentation is conducted on the input camera image. Then, the input point cloud is augmented with image semantic information. In the third part, spatial features in pillars and semantic features in voxels are both encoded in a Bird's Eye View (BEV) representation. Afterwards, spatial and semantic features are fused and further processed in the backbone. Finally, the output of the backbone is fed into the detection head, which predicts oriented 3D bounding boxes. Modified from Figure in [FCH+20] ©IEEE.

4.2.1 Image Semantic Segmentation

In order to obtain a compact representation of the semantic information in the image, semantic segmentation is performed. To this end, a semantic segmentation network predicts segmentation scores for each pixel, representing the probabilities of belonging to a set of classes. In practice, it is often sufficient to use the categorical label IDs (Identifiers) to represent the semantic segmentation output. Therefore, in this work, two kinds of output representations are studied: the softmax class probabilities and the class label IDs.

In SemanticVoxels, the image semantic segmentation network is independently trained. This allows to use any network for this task in the fusion-based 3D object detection pipeline. In this work, two networks are chosen: DeepLabv3+ [CZP+18] and DDRNet [HPSJ21], where the former achieves state-of-the-art

segmentation performance, while the latter is real-time capable with a slightly reduced segmentation performance.

4.2.2 Point Cloud Augmentation

After obtaining image semantic segmentation results, LiDAR points are projected onto the image plane and augmented with the compact image semantic features. In this way, the image information is lifted to the 3D space and can be formed in the BEV.

Typically, a LiDAR point in the point cloud is represented by (x, y, z, r), where x, y, and z are the 3D coordinates and r is the reflectance. The raw point cloud is augmented using the obtained image semantic segmentation output. To this end, a 3D point $(x,\ y,\ z)$ in the LiDAR coordinate system is first transformed to the camera coordinate system $(x',\ y',\ z')$ via a rigid transformation, which can be mathematically formulated as:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \tag{4.1}$$

where $R_{3\times3}$ and $t_{3\times1}$ are the rotation matrix and translation vector between the LiDAR coordinate system and the camera coordinate system, respectively. Afterwards, the transformed point is projected onto the image plane using a perspective projection, according to:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \propto K_{3\times3} \cdot \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}, \tag{4.2}$$

where $K_{3\times3}$ denotes the camera intrinsic matrix and $(u,\ v)$ are the image coordinates. For each point, which falls in the image after projection, the semantic segmentation result of the corresponding nearest pixel is then appended to the original 3D point.

4.2.3 Multi-modal Feature Encoding

After point cloud augmentation, the resulting augmented point cloud contains both spatial information from the LiDAR data and semantic information from the camera image. To encode multi-modal features in the BEV, it is further processed in two separate branches, namely spatial feature encoding and semantic feature encoding.

Spatial Feature Encoding The augmented point cloud is processed through a pillar feature network, outlined in Section 2.2.4, to extract LiDAR spatial features. During this process, the augmented point cloud is first discretized into a set of pillars on the XY plane. In the subsequent processing of the pillar feature network, only the raw data provided by the LiDAR, i.e., the (x, y, z, r) information of the points, is utilized for learning spatial features. Finally, a top-view representation of size (W, H, C) is generated, where W and H denote the width and height of the grid, corresponding to the X and Y axes, respectively, while C denotes the feature dimension for each pillar. In this way, LiDAR spatial features are successfully encoded in a BEV representation.

Semantic Feature Encoding As described in Section 4.2.2, the semantic image information is lifted to 3D space through point cloud augmentation. The main goal of semantic feature encoding is to represent this compact image information in the BEV. During the spatial feature encoding, the augmented point cloud has already been pillarized. Thus, a simple but effective way would be to directly encode the semantic information within the pillars generated above [VLHB20]. Considering that preserving the spatial distribution of semantic information can be advantageous for object detection and classification, the generated pillars are further divided into voxels along the height direction, and voxel-wise semantic features are encoded.

The encoding of semantic features for each voxel depends on the representation of the semantic segmentation output. When using the segmentation scores, the semantic features of each voxel are summarized by calculating the mean probabilities over the classes of all points in the voxel. In the case of categorical label IDs, it is determined as the label ID of the class with the most occurrences. These two encoding methods guarantee efficient utilization of image semantic

information, which is important for real-time applications. Next, voxel-wise semantic features are stacked along the Z axis and a 3×3 convolutional layer is applied to aggregate voxel-wise features. Finally, semantic features in a BEV representation of size (W, H, K) are obtained.

4.2.4 Backbone and Feature Fusion

In order to extract relevant high-dimensional features for object detection, the obtained LiDAR features and image features are further processed by a backbone network. Meanwhile, different feature fusion strategies are considered to achieve optimal fusion of LiDAR spatial and image semantic features.

Backbone The backbone in PointPillars [LVC+19] is used as the base backbone. Its architecture is depicted in Figure 4.2. The backbone contains three blocks of convolutional layers, where each block downsamples the spatial size of the feature maps by half. Besides, the first block keeps the channel dimension of the input, while the second and third blocks double it. The output of every block is then upsampled and the resulting features are concatenated to create final feature maps.

Feature Fusion To effectively fuse multi-modal features and determine the optimal way, three kinds of feature fusion schemes, namely early, late and deep fusion, are discussed. Early fusion combines LiDAR spatial and image semantic features before inputting into the first block of the backbone (Figure 4.3a). In contrast, late fusion uses separate networks to learn domain specific features from individual inputs, and combines them before the final detection head stage. In this work, the base backbone extracts features from the input LiDAR features, which are then fused with image semantic features (Figure 4.3b). The fused feature maps are finally fed into the detection head. Deep fusion performs feature fusion at various resolution levels in a top-down manner. As depicted in Figure 4.3c, input LiDAR and image features are first combined before forwarding them into the backbone. Then, input image features are gradually downsampled in three stages. The resulting features maps are fused with the feature maps outputted by the three blocks in the backbone, respectively. In all fusion schemes, the concatenation operation is adopted to combine features.

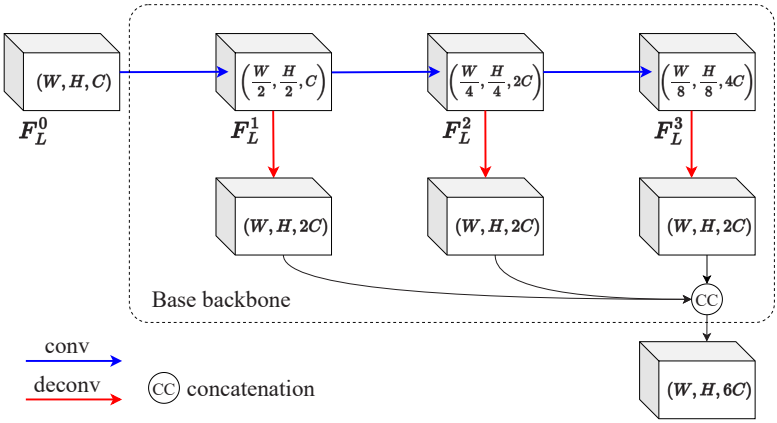


Figure 4.2: The base backbone. The blue edges represent convolution operations, while the red ones correspond deconvolutions. LiDAR spatial feature maps are denoted with F_L where the superscript denotes the resolution level of the respective feature maps.

4.2.5 Detection Head

SemanticVoxels employs two types of detection heads: an anchor-based head from PointPillars [LVC+19] or a center-based from CenterPoint [YZK21]. The corresponding SemanticVoxels models are termed SemanticVoxels-Anchor (SV-A) and SemanticVoxels-Center (SV-C), respectively. Next, the details of both detection heads are introduced.

Anchor-based Head

To predict oriented 3D bounding boxes, the same anchor-based detection head as in [LVC+19, YML18] is applied. It processes the fused feature maps from the backbone network and consists of three independent heads, which are specialized for the desired tasks, namely bounding box regression, bounding box classification, and direction classification. Each head is composed of a simple 1×1 convolution layer.

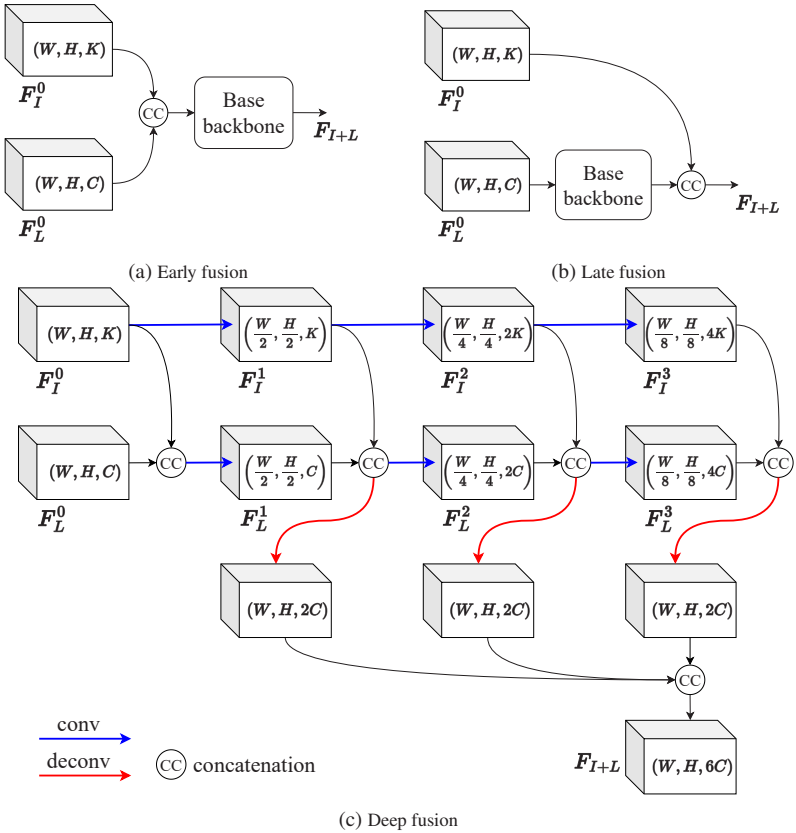


Figure 4.3: Various feature fusion schemes. The blue edges represent convolution operations, while the red ones correspond deconvolutions. LiDAR spatial features F_L and image semantic features F_I are fused using three different strategies, namely early, late, and deep fusion.

Bounding Box Regression Head The regression head aims to estimate the object's bounding box parameters. Rather than regressing the absolute parameters directly, it predicts the offsets to prior 3D anchor boxes. As suggested in [LVC+19, YML18], a 3D bounding box is parameterized by the 3D centroid position x, y, z, its length l, width w, and height h, as well as the orientation θ.

In particular, following [YML18], the *sine* function is applied when regressing the orientation. Thus, the regression variables are given as:

$$\mathbf{y}_{\text{bbox}} = (\Delta x, \Delta y, \Delta z, \log(l), \log(w), \log(h), \sin(\Delta\theta)). \tag{4.3}$$

During training, the regression targets, i.e., the offsets between ground-truth boxes (denoted with the superscript gt) and their matched anchors (denoted with the superscript a), are computed by:

$$\begin{gathered} \Delta x = \frac{x^{\text{gt}} - x^{\text{a}}}{d^{\text{a}}},\ \Delta y = \frac{y^{\text{gt}} - y^{\text{a}}}{d^{\text{a}}},\ \Delta z = \frac{z^{\text{gt}} - z^{\text{a}}}{h^{\text{a}}}, \\ \log(l) = \log\left(\frac{l^{\text{gt}}}{l^{\text{a}}}\right),\ \log(w) = \log\left(\frac{w^{\text{gt}}}{w^{\text{a}}}\right),\ \log(h) = \log\left(\frac{h^{\text{gt}}}{h^{\text{a}}}\right), \\ \sin(\Delta\theta) = \sin(\theta^{\text{gt}} - \theta^{\text{a}}), \end{gathered} \tag{4.4}$$

where the anchor's diagonal length $d^{\text{a}} = \sqrt{(l^{\text{a}})^2 + (w^{\text{a}})^2}$. SmoothL1 [Gir15] is used to calculate the 3D bounding box regression loss, giving the overall loss $\mathcal{L}_{\text{reg}}$ as follows:

$$\mathcal{L}_{\text{reg}} = \sum_{b \in \{\Delta x, \Delta y, \Delta z, \log(l), \log(w), \log(h), \sin(\Delta\theta)\}} \text{SmoothL1}\left(\hat{b} - b\right), \tag{4.5}$$

in which $\hat{b}$ denotes the predicted box offsets. SmoothL1 corresponds to a quadratic loss function if the error falls below a certain threshold or an L1 cost function otherwise. It can be defined by:

$$\text{SmoothL1}(x) = \begin{cases} 0.5x^2/\beta & \text{if } |x| < \beta \\ |x| - 0.5\beta & \text{otherwise,} \end{cases} \tag{4.6}$$

where $\beta = 1.0/9.0$ is applied in this work.

Note that the learned orientation offset could not distinguish boxes when they are shifted by $\pm\pi$ radians. To address this issue, each object's heading direction, either positive or negative, is categorized by an additional direction classification head, which is described in more detail later.

Direction Classification Head To solve the aforementioned ambiguous heading prediction problem, the direction classification head classifies the heading direction as positive or negative by predicting a softmax probability $\hat{p}_{\text{dir}}$. The cross-entropy loss is used to compute the classification loss $\mathcal{L}_{\text{dir}}$.

Bounding Box Classification Head This head outputs a softmax confidence $\hat{p}_{\text{cls}}$ for each anchor to determine whether it is a positive object or background. Considering that the numbers of positive and negative samples are imbalanced, focal loss [LGG+17] is used as the object classification loss:

$$\begin{aligned} \mathcal{L}_{\text{cls}} &= -\alpha_{\text{t}} \left(1 - p_{\text{t}}\right)^{\gamma} \log p_{\text{t}}, \\ p_{\text{t}} &= \begin{cases} \hat{p}_{\text{cls}} & \text{if } y = 1 \\ 1 - \hat{p}_{\text{cls}} & \text{otherwise,} \end{cases} \end{aligned} \tag{4.7}$$

where $y \in \{0, 1\}$ specifies the ground-truth label. During training, the recommended settings in [LVC+19, LGG+17] with $\alpha = 0.25$ and $\gamma = 2$ are used.

Considering all three heads, the total loss of the anchor-based detection head is therefore defined as:

$$\mathcal{L}_{\text{total}} = \frac{1}{N_{\text{pos}}} \left(\beta_1 \mathcal{L}_{\text{reg}} + \beta_2 \mathcal{L}_{\text{dir}} + \beta_3 \mathcal{L}_{\text{cls}}\right), \tag{4.8}$$

where N_{pos} denotes the number of positive anchors. β_i are constant weights for balancing different losses. The settings $\beta_1 = 2.0$, $\beta_2 = 0.2$, and $\beta_3 = 1.0$ are adopted, as suggested in [LVC+19].

Center-based Head

Given the output feature maps of the backbone, the center-based head directly predicts object center positions in the top view and then regresses to other object attributes. These are achieved by using a center heatmap head and several regression heads.

Center Heatmap Head For all classes, this head produces a dense N_{c}-channel heatmap $\hat{Y} \in [0, 1]^{W_{\text{o}} \times H_{\text{o}} \times N_{\text{c}}}$, where W_{o} and H_{o} denote the width and height

of the heatmap, and N_c is the number of classes. By extracting peaks of each single heatmap, each object's center position (x_c, y_c) on the horizontal plane can be easily determined. In addition, the heatmap values are also used as detection scores.

During training, for each 2D ground truth center obtained by projecting the 3D box center onto the BEV, a 2D Gaussian distribution is computed on the target heatmap Y. Then, a modified focal loss [LD20, ZWK19] is applied to train this head, giving:

$$\mathcal{L}_{\text{hm}} = \frac{-1}{N_{\text{gt}}} \sum_{x,y,c} \begin{cases} (1-\hat{Y}_{xyc})^{\gamma} \log(\hat{Y}_{xyc}) & \text{if } Y_{xyc} = 1 \\ (1-Y_{xyc})^{\beta} (\hat{Y}_{xyc})^{\gamma} \log(1-\hat{Y}_{xyc}) & \text{otherwise,} \end{cases} \tag{4.9}$$

where β and γ are parameters of the focal loss, and N_{gt} is the number of ground truth centers. Following [YZK21], β is set to 4 and γ to 2.

Regression Heads To predict an object in the form of a 3D oriented bounding box, multiple heads are further applied to regress to a set of object properties $\mathbf{y}_{\text{bbox}}$: the positional offsets between its center and the local BEV grid cell center (o_x, o_y), the height of the center z_c, the 3D size (l, w, h) in the log scale, and a yaw angle represented in *sine* and *cosine*, thereby yielding:

$$\mathbf{y}_{\text{bbox}} = (o_x, o_y, z_c, \log(l), \log(w), \log(h), \sin(\theta), \cos(\theta)). \tag{4.10}$$

The local offsets (o_x, o_y) aims to mitigate the quantization error introduced by rasterization and striding in the backbone. The height of the center z_c estimates the object's elevation above ground. The *sine* and *cosine* of the yaw angle are used to represent the orientation, in order to be continuous and thus avoid a singularity at $\theta = \pm\pi$ radians. Together with the predicted box extents, all state information of the 3D box is obtained.

During training, L1 loss is used for all regression heads but only at the ground truth center locations:

$$\mathcal{L}^{b}_{\text{reg}} = \frac{1}{N_{\text{gt}}} \sum_{k=1}^{N_{\text{gt}}} |\hat{b}_k - b_k|, \; b \in \{o_x, o_y, z_c, l, w, h, \sin(\theta), \cos(\theta)\}, \tag{4.11}$$

where $\hat{b}$ and b denotes the predicted and ground truth values, respectively, and N_{gt} is the number of ground truth centers. Note that for experiments on the nuScenes dataset, an additional head is used to estimate the velocity $(v_{\text{x}}, v_{\text{y}})$ of each object. A loss weight of 0.2 is applied when calculating loss for the velocity targets.

Finally, after obtaining the bounding box information, either from the anchor-based or the center-based detection head, Non-Maximum Suppression (NMS) is applied to select the best detections from the redundant bounding boxes.

4.3 Experiments

In this section, experimental evaluations of SemanticVoxels are presented. First, Section 4.3.1 introduces the datasets used for training and evaluation. Afterwards, the details of networks for image semantic segmentation and object detection are covered in Section 4.3.2 and 4.3.3, respectively. Finally, Section 4.3.4 provides the results of quantitative evaluations, followed by exemplary qualitative results in Section 4.3.5.

4.3.1 Datasets

The experiments are conducted on the KITTI object detection [GLU12] dataset and the nuScenes [CBL+20] dataset, which are introduced in Section 2.5.1 and 2.5.2, respectively.

KITTI Dataset For the purposes of training and evaluation, the 7481 annotated samples in the KITTI object detection [GLU12] dataset are divided into 3712 training and 3769 validation samples, as in [CKZ+15, CMW+17, KML+18]. The experimental results are evaluated using AP for the 3D object detection task with IoU thresholds of 0.7 for Car and 0.5 for both Pedestrian and Cyclist. In addition, evaluations are conducted in three difficulty levels: Easy, Moderate, and Hard. A more detailed description of the KITTI dataset is provided in Section 2.5.1.

nuScenes Dataset For training and evaluation, the official nuScenes split [CBL+20], with 700 sequences for training and 150 for validation, is used. The models are trained on 10 nuScenes object classes: Car, Truck, Bus, Trailer, Construction Vehicle, Pedestrian, Motorcycle, Bicycle, Traffic Cone, and Barrier. The mAP and the nuScenes detection score (NDS) metrics are adopted for performance evaluation. Note that the mAP is computed by averaging the APs over all classes and 4 matching thresholds. A comprehensive overview of the nuScenes dataset and the evaluation metrics is available in Section 2.5.2.

4.3.2 Semantic Segmentation Network Details

This section describes the experimental details of the image semantic segmentation network on both KITTI and nuScenes datasets. Due to its fast inference speed and competitive segmentation performance, DDRNet-23-Slim is considered for obtaining segmentation results.

KITTI The considered DDRNet-23-slim network is first pre-trained on ImageNet [DDS+09], and then trained on Cityscapes [COR+]. Finally, it is fine-tuned on KITTI-domain datasets, first on the KITTI-STEP (Segmenting and Tracking Every Pixel) dataset [WXC+21] and then on the KITTI pixel semantic segmentation dataset [AMM+18]. When inferring on the KITTI object detection dataset, the segmentation scores for Pedestrian, Rider, Bicycle, and Car are kept and the class label ID for each pixel is further determined from the maximum segmentation score.

In the object detection dataset, a cyclist is defined as a combination of rider and bicycle. However, in the semantic segmentation dataset, a cyclist is only labeled as rider with bicycle being a separate class. To tackle this inconsistency, it is necessary to map bicycles with riders to the Cyclist class, while classifying the rest to background. As in [VLHB20], after point cloud augmentation, points labeled as bicycles and within a 1 m radius of rider points are mapped to cyclists. This remapping strategy is however not applicable when augmenting the point cloud using segmentation scores. In such a case, all bicycles are suppressed to background and only riders are mapped to cyclists.

To study the impact of semantic segmentation quality on the final object detection performance, the DeepLabv3+ [CZP+18] network with state-of-the-art

performance is considered as well. The trained model with the WideResNet38 backbone [WSVDH19] provided in [ZSR+19] is employed to predict categorical labels. Initially, the network is pre-trained on the Mapillary Vistas dataset [NOBK17], and then fine-tuned on Cityscapes [COR+] and the KITTI semantic segmentation dataset [AMM+18].

nuScenes nuScenes does not contain image segmentation annotations. However, the nuImages dataset [CBL+20] complements nuScenes by providing 93k images annotated with semantic segmentation labels for all classes in the nuScenes dataset. Therefore, nuImages is used to fine-tune the DDRNet-23-slim network after pre-training on ImageNet [DDS+09] and Cityscapes [COR+]. Next, the segmentation labels for nuScenes images are predicted using the trained model.

4.3.3 Object Detection Network Details

In this section, the experimental details of the 3D object detection networks are provided. SemanticVoxels is implemented using two baseline methods with different detection heads: anchor-based PointPillars [LVC+19] and center-based CenterPoint [YZK21], termed SemanticVoxels-Anchor and SemanticVoxels-Center, respectively. The implementation is based on the OpenPCDet [Tea] codebase.

In order to prove the effectiveness of the proposed fusion-based approach, SemanticVoxels models are compared with their corresponding LiDAR-only baselines. Note that the baseline models have identical settings as the corresponding SemanticVoxels models, except for the differences due to the use of semantic image information. Hence, this section only covers the details of SemanticVoxels.

The anchor-based SemanticVoxels is evaluated on both the KITTI and nuScenes datasets, while the center-based variant is only evaluated on nuScenes. All SemanticVoxels models are optimized using the AdamW [LH19] optimizer with an initial learning rate of 0.003, a weight decay rate of 0.01, and exponential decay rates of $\beta_1 = 0.9$ and $\beta_2 = 0.99$ (see Section 2.1.4). The one-cycle learning rate policy [ST19] is applied with the parameters summarized in Ta-

ble 4.1. The learning rate and the exponential decay rate β_1 in AdamW are dynamically adjusted during training according to this policy.

Parameter	Value
Max. learning rate	3e-3 (KITTI), 1e-3 (nuScenes)
Division factor	10
Base momentum	0.85
Max. momentum	0.95
Annealing	cosine
pct_start	0.4

Table 4.1: Parameters of the one-cycle policy used during training models. pct_start indicates the percentage of the cycle when the learning rate increases.

During training, multiple data augmentation techniques [ZT18, YML18] are employed to prevent the network from over-fitting. Data augmentation is applied after the input point cloud is augmented with image semantic information but before it is discretized into pillars or voxels. Specially, on KITTI, only the LiDAR points in the camera field of view are considered in this process. Below are descriptions of all data augmentation techniques:

- Global flipping: LiDAR points and all 3D ground-truth boxes are flipped along the X axis by a 50% chance. On nuScenes, random flipping is additionally performed along the Y axis.
- Global rotation: LiDAR points and ground-truth boxes are rotated around the upright Z axis by a uniformly distributed angle between $[-\pi/4, +\pi/4]$ radians for KITTI and $[-\pi/8, +\pi/8]$ for nuScenes.
- Global scaling: the 3D location and size of each ground-truth box as well as the coordinates of all LiDAR points are multiplied by a factor randomly sampled from a uniform distribution between $[0.95, 1.05]$.
- Ground-truth sampling: annotated 3D bounding boxes in the training set and LiDAR points inside them are extracted to create a database. During training, a predefined number of ground-truth boxes and corresponding LiDAR points in the database are randomly sampled and pasted to the

input frame. Newly added bounding boxes that intersect with existing bounding boxes will be discarded. Note that "shadows" behind objects in LiDAR scans caused by occlusions are not simulated when adding objects to the scene.

Next, the specific implementation details of SemanticVoxels on KITTI and nuScenes are introduced separately.

KITTI For all experiments on KITTI, the input range is defined as $[0, 69.12] \times [-39.68, 39.68] \times [-3, 1]$ meters along the X, Y, and Z axes, respectively. The input LiDAR point cloud is cropped accordingly. When generating pillars, the pillar grid size is set to $0.16\,\text{m} \times 0.16\,\text{m}$, the number of pillars P is 16000, and the number of points per pillar N is 32. The voxels for encoding image semantic features share the same grid size as pillars and have a resolution of 0.25 meters along the Z axis. During the voxelization, in the height direction a fixed elevation range of $[0.1, 2.1]$ meters above the ground is considered. Finally, all models are trained with a batch size of 4 for 120 epochs.

nuScenes For experiments of SemanticVoxels-Anchor and SemanticVoxels-Center on nuScenes, the input range is set to $[-51.2, 51.2] \times [-51.2, 51.2] \times [-3, 5]$ meters along the X, Y, and Z axes, respectively. The number of pillars P allowed in each point cloud is 30000, and number of points per pillar N is set to 20. The generated pillars and voxels have a grid size of $0.2\,\text{m} \times 0.2\,\text{m}$, and each voxel has a resolution of 0.25 meters on the Z axis. All LiDAR points within the entire input range are considered during voxelization.

Besides the data augmentation strategies mentioned above, the training set is expanded by duplicating training samples with less represented objects, in order to alleviate severe class imbalance in nuScenes, as suggested in CBGS [ZJZ+19]. Moreover, it is common to accumulate the point cloud of an annotated keyframe with those of 9 preceding non-keyframes after transformation to compensate the ego motion [CBL+20, ZJZ+19, VLHB20, YZK21]. Following this practice, experiments are conducted using 10 LiDAR sweeps in addition to using single sweep as input. All models are trained with a batch size of 4 for 20 epochs.

4.3.4 Quantitative Evaluation

This section presents the quantitative results of SemanticVoxels, first on the KITTI dataset and then on the nuScenes dataset.

KITTI Evaluation

On the KITTI dataset, experiments with SemanticVoxels-Anchor are performed, and the results are then analyzed.

First, to prove the effectiveness of the proposed approach and determine a proper fusion strategy, SemanticVoxels is implemented and trained using three different fusion schemes as described in Section 4.2.4, whereas PointPillars is trained as a baseline model. All models are evaluated on three classes and across easy, moderate, and hard levels using the 3D AP metric. In addition, the mAP over nine APs (three classes $\times$ three difficulty levels) for each model as well as the AP differences between SemanticVoxels and PointPillars are computed and reported. The corresponding evaluation results are summarized in Table 4.2.

Afterwards, thorough ablation experiments are conducted to investigate the effect of different design choices in SemanticVoxels, including an alternative network with state-of-the-art semantic segmentation performance and an alternative representation of the segmentation information. In addition, the effects when not performing elevation correction or when dealing with an degraded calibration are investigated. The corresponding evaluation results are summarized in Table 4.3. Note that the rows marked with Δ show the AP and mAP differences from the baseline method for each experiment (highlighted in red for decreases or green for increases).

In the following, all results in Tables 4.2 and 4.3 are discussed in detail.

Effectiveness As shown in Table 4.2, when compared with the baseline, the proposed SemanticVoxels shows broad improvements across all comparisons, regardless of the fusion strategy. Particularly, significant improvements are further observed on more challenging pedestrian and cyclist categories. These demonstrate the effectiveness of the proposed LiDAR-camera fusion approach

	PP	SV-A early	SV-A deep	SV-A late
mAP	66.68	69.50	**69.63**	68.35
Δ	-	+2.82	**+2.95**	+1.67
$AP_{car,easy}$	87.17	**88.59**	88.02	88.03
Δ	-	**+1.42**	+0.85	+0.86
$AP_{car,moderate}$	77.75	78.34	**78.67**	78.13
Δ	-	+0.59	**+0.92**	+0.38
$AP_{car,hard}$	74.79	75.44	**75.85**	75.44
Δ	-	+0.65	**+1.06**	+0.65
$AP_{ped,easy}$	57.78	60.99	**61.25**	58.32
Δ	-	+3.21	**+3.47**	+0.54
$AP_{ped,moderate}$	51.95	**54.88**	54.73	52.74
Δ	-	**+2.93**	+2.78	+0.79
$AP_{ped,hard}$	47.47	**49.86**	49.74	48.01
Δ	-	**+2.39**	+2.27	+0.54
$AP_{cyc,easy}$	81.76	**88.26**	88.07	86.81
Δ	-	**+6.50**	+6.31	+5.05
$AP_{cyc,moderate}$	62.86	66.91	**67.53**	65.97
Δ	-	+4.05	**+4.67**	+3.11
$AP_{cyc,hard}$	58.59	62.21	**62.85**	61.70
Δ	-	+3.62	**+4.26**	+3.11

Table 4.2: Evaluation results of SemanticVoxels measured by Average Precision (AP) and mean AP (mAP) on the KITTI validation set. The best results are highlighted in bold. Δ indicates the performance difference between SemanticVoxels and the baseline PointPillars. Abbreviations: PointPillars (PP), and SemanticVoxels-Anchor (SV-A).

and indicate the advantages of exploiting image information in 3D object detection.

	Ref	Exp1	Exp2	Exp3	Exp4
Sem. Seg. Network	DDRNet	DeepLab V3+	DDRNet	DDRNet	DDRNet
Sem. Seg. Representation	ID	ID	score	ID	ID
Elevation Correction	✓	✓	✓		✓
Calibration Errors					✓
mAP	69.63	69.27	68.43	68.97	69.23
Δ	-	−0.36	−1.20	−0.66	−0.40
AP_{car}	78.67	78.88	78.07	78.23	78.27
Δ	-	+0.21	−0.60	−0.44	−0.40
AP_{ped}	54.73	55.81	55.05	54.55	54.38
Δ	-	+1.08	+0.32	−0.18	−0.35
AP_{cyc}	67.53	65.47	63.47	66.19	66.54
Δ	-	−2.06	−4.06	−1.34	−0.99

Table 4.3: Evaluation results of SemanticVoxels with various network configurations on the KITTI validation set. Δ indicates the performance difference between each experiment and the reference experiment.

Fusion Scheme Among three fusion schemes, both deep and early fusion achieve remarkable improvements in performance and notably outperform late fusion (see Table 4.2). Deep fusion shows a marginal improvement against early fusion, but at a cost of slightly higher computations. Therefore, the fusion scheme should be decided according to the accuracy and run-time requirements of specific applications. Unless otherwise noted, deep fusion is employed in SemanticVoxels in the following experiments.

	IoU (%)	
	DDRNet	DeepLabv3+
Person	54.52	61.65
Rider	52.92	58.78
Car	86.44	91.02
mean	64.63	70.48

Table 4.4: Image semantic segmentation performance of DDRNet-23-slim and DeepLabv3+ on the KITTI semantic segmentation dataset.

Segmentation Quality In SemanticVoxels, the compact image semantic segmentation output is used to obtain semantic features in the top view. Thus, it is of great significance to study the impact of semantic segmentation quality on the subsequent object detection results. For this reason, two networks with varying accuracy performance, i.e. DDRNet and DeepLabv3+, are trained. Table 4.4 summarizes the evaluation results in terms of IoU on the KITTI semantic segmentation dataset. DeepLabv3+ shows significant performance advantages, outperforming DDRNet in mean IoU as well as IoUs for all three classes of interest.

Using the two semantic segmentation networks mentioned above, two SemanticVoxels networks with identical settings are trained and evaluated. Settings Ref and Exp1 in Table 4.3 summarize the results. By using the improved segmentation output, SemanticVoxels shows performance boost for pedestrians and cars. Unexpectedly, the performance notably decreases on the cyclist class, causing a slight drop in mAP.

Segmentation Representation To investigate the effect of using different segmentation output representations in SemanticVoxels, further experiments are conducted. The results are summarized in experiments Ref and Exp2 in Table 4.3. Using more informative scores surprisingly decreases mAP by −1.20, mainly due to a remarkable performance drop on cyclists. This suggests that the simple semantic feature encoder and backbone cannot fully utilize the additional information in scores [VLHB20]. Unlike cyclists, the APs on pedestrians

and cars of two representations are comparable. This may be explained that there are much fewer cyclist samples in KITTI, which may increase training noise and lead to large AP variations on cyclists.

Elevation Correction In common traffic scenarios, all traffic participants exist on the ground. SemanticVoxels encodes image semantic information in voxels, considering that preserving its spatial distribution along the height direction is valuable for accurate object detection. Aiming to obtain the actual distribution of the semantic information of each object above the ground, it is essential to derive the height relative to the ground for each LiDAR point. Thus, the ground surface is estimated first, and then height estimates of all points are computed.

The effect of elevation correction on the final detection performance is evaluated. Experiments Ref and Exp3 in Table 4.3 report the evaluation results for with and without elevation correction, respectively. Without elevation correction, it is assumed that the ground surface is planar and that the ego vehicle has a pitch angle of 0°. With elevation correction, marginal AP increases on cars and pedestrians are observed. Together with a notable AP gain of 1.34% on cyclists, estimating the ground surface and then correcting the elevation slightly improves the mAP by 0.66%.

Calibration Accuracy A key step in SemanticVoxels is the point cloud augmentation, in which the image semantic information is lifted back to 3D space. This is achieved by projecting LiDAR points onto the camera image by using the LiDAR-camera extrinsics as well as the camera intrinsics. Consequently, the calibration accuracy of the aforementioned parameters affects the performance of point cloud augmentation, which in turn affects the final object detection performance. The calibration of multi-modal sensors is a non-trivial task, and it is hard to achieve exact calibration results. Moreover, the extrinsic parameters may alter due to mechanical vibration and thermal strain.

To study the performance and robustness of SemanticVoxels under calibration errors, further experiments are conducted. The calibration parameters provided by KITTI [GLU12] are regarded as reference calibration. To simulate realistic calibration errors, zero-mean Gaussian noises are applied on the relative rotation (yaw ϕ, pitch θ, and roll ψ) and translation (t_x, t_y, and t_z) between the

LiDAR and the camera, as well as the focal length f and the principal point (u_0, v_0) of camera, giving:

$$\begin{aligned} \Delta\phi, \Delta\theta, \Delta\psi &\sim \mathcal{N}(0,\, \sigma_{\text{rot}}^2) \\ \Delta t_{\text{x}}, \Delta t_{\text{y}}, \Delta t_{\text{z}} &\sim \mathcal{N}(0,\, \sigma_{\text{trans}}^2) \\ \Delta f &\sim \mathcal{N}(0,\, \sigma_{\text{f}}^2) \\ \Delta u_0, \Delta v_0 &\sim \mathcal{N}(0,\, \sigma_{\text{p}}^2). \end{aligned} \tag{4.12}$$

For the LiDAR-camera extrinsics, standard deviations $\sigma_{\text{rot}} = 0.05°$ and $\sigma_{\text{trans}} = 0.01$ m are chosen, considering the capabilities of the state-of-the-art calibration approaches [Kü20, Pus17]. For the camera intrinsics, $\sigma_{\text{f}} = 1$ pixel and $\sigma_{\text{p}} = 1$ pixel are selected empirically, based on the camera intrinsic parameter errors reported in [Kü20]. In addition, all disturbances are clipped to the interval $[-3\sigma, 3\sigma]$ to avoid extreme outliers that may lead to unrealistic sensor setups. During evaluation, calibration errors are generated for each input sample.

Experiments Ref and Exp4 in Table 4.3 summarize the evaluation results when using the reference calibration and the disturbed calibration, respectively. Note that Exp4 employs the same model as in Ref, but uses noise calibration in evaluation. As expected, the APs for all three classes and the mAP decrease, but only slightly, which demonstrates the robustness of SemanticVoxels against potential calibration errors. Note that this robustness is vital for long-term applications on vehicles. Additionally, when further comparing the results of Exp4 with the those of PointPillars in Table 4.2, it can be observed that SemanticVoxels still performs much better than the LiDAR-only baseline, even with disturbed calibration parameters. This indicates the practical value of the proposed fusion-based approach.

nuScenes Evaluation

On the nuScenes dataset, experiments with both anchor-based and center-based SemanticVoxels are conducted, and the results are then analyzed.

Before presenting the object detection results, the image semantic segmentation performance of DDRNet on the nuImages validation set is first introduced.

Table 4.5 shows the evaluation results measured by IoU. With a mIoU of 73.47%, the trained DDRNet model demonstrates competitive segmentation performance. This model is then used to produce image semantic segmentation results on the nuScenes dataset, for further training of SemanticVoxels.

	IoU (%)
Car	89.36
Truck	73.60
Bus	76.92
Trailer	49.03
Constr. Veh.	53.59
Pedestrian	68.41
Motorcycle	78.90
Bicycle	64.33
Traffic cone	72.30
Barrier	83.00
Others	99.10
mean	73.49

Table 4.5: Image semantic segmentation performance of DDRNet-23-slim on the nuImages validation set. Abbreviations: construction vehicle (Constr. Veh.).

Anchor-based First, the anchor-based PointPillars and SemanticVoxels-Anchor are trained on nuScenes. As it is very common in the nuScenes benchmark [CBL+20, ZJZ+19, VLHB20, YZK21], both single LiDAR sweep and 10 sweeps are considered in the experiments. All models are then evaluated on the nuScenes validation set using the Average Precision (AP) and mean AP (mAP) metrics. In addition, the AP differences between SemanticVoxels and PointPillars are calculated. The evaluation results are summarized in Table 4.6 for both 1 and 10 input LiDAR sweeps. Table 4.7 further summarizes nuScenes detection scores (NDS) for all models.

	1 sweep		10 sweeps	
	PP	SV-A	PP	SV-A
AP_{Car}	73.88	76.60	81.27	82.25
Δ	-	+2.72	-	+0.98
AP_{Truck}	42.67	48.35	49.58	54.01
Δ	-	+5.68	-	+4.43
AP_{Bus}	58.96	64.41	61.67	66.66
Δ	-	+5.45	-	+4.99
$AP_{Trailer}$	29.84	31.59	34.00	37.29
Δ	-	+1.75	-	+3.29
$AP_{Constr.\ Veh.}$	5.76	10.36	10.02	13.59
Δ	-	+4.60	-	+3.57
$AP_{Pedestrian}$	59.44	73.26	72.59	79.14
Δ	-	+13.82	-	+6.55
$AP_{Motorcycle}$	23.13	40.84	31.84	47.93
Δ	-	+17.71	-	+16.09
$AP_{Bicycle}$	0.92	19.83	7.50	27.30
Δ	-	+18.91	-	+19.80
$AP_{Traffic\ cone}$	38.36	62.53	44.01	66.51
Δ	-	+24.17	-	+22.50
$AP_{Barrier}$	45.58	53.00	50.38	56.73
Δ	-	+7.42	-	+6.35
mAP	37.85	48.08	44.28	53.14
Δ	-	+10.23	-	+8.86

Table 4.6: Detailed detection performance of PointPillars and SemanticVoxels-Anchor on the nuScenes validation set. Experimental results for both 1 and 10 input LiDAR sweeps are presented. Δ indicates the performance difference between SemanticVoxels and the baseline PointPillars. Abbreviations: PointPilalrs (PP), SemanticVoxels-Anchor (SV-A), and construction vehicle (Constr. Veh.).

Input	NDS (%)		
sweep	PP	SV-A	Δ
1	44.36	49.21	+4.85
10	58.07	62.51	+4.44

Table 4.7: nuScenes Detectin Score (NDS) of anchor-based PointPillars and SemanticVoxels on the nuScenes validation set. Δ indicates the performance difference between SemanticVoxels and the baseline PointPillars. Abbreviations: PointPilalrs (PP) and SemanticVoxels-Anchor (SV-A).

As shown in Table 4.6, SemanticVoxels shows extensive AP boosts on all nuScenes categories, improving mAP by +10.23 (1 sweep) and +8.86 (10 sweeps). This finding is consistent with the observations on KITTI, which provides further evidence demonstrating the benefit of leveraging semantic information from image for enhancing 3D object detection performance.

Among all classes, traffic cones get the largest AP gain of more than 20%, although the LiDAR-only baseline is already competitive. Additionally, strong improvements have been observed for challenging Vulnerable Road Users (VRUs) such as pedestrians, bicycles and motorcycles. These objects are relatively smaller than other nuScenes objects and have fewer LiDAR points on them. Therefore, the semantic information provided by the image modality is particularly beneficial for detecting them accurately.

As seen in Table 4.7, SemanticVoxels notably boosts NDS by +4.85 (1 sweep) and +4.44 (10 sweeps). This confirms the effectiveness of the proposed SemanticVoxels.

Center-based The evaluation results so far have strongly proven the effectiveness of SemanticVoxels with anchor-based detection heads. Then, the center-based CenterPoint and SemanticVoxels are studied on nuScenes. The evaluation results in terms of AP are given in Table 4.8, while the NDS results are described in Table 4.9.

Regarding APs for 1 and 10 sweeps, similar trends are observed to those of the anchor-based SemanticVoxels, i.e., all categories obtain improvements, while

	1 sweep		10 sweeps	
	CP	SV-C	CP	SV-C
AP_{Car}	77.05	79.24	83.68	84.24
Δ	-	+2.19	-	+0.56
AP_{Truck}	42.56	48.68	52.69	54.15
Δ	-	+6.12	-	+1.46
AP_{Bus}	59.12	64.21	62.90	65.64
Δ	-	+5.09	-	+2.74
$AP_{Trailer}$	29.44	29.70	33.51	34.51
Δ	-	+0.26	-	+1.00
$AP_{Constr.\ Veh.}$	6.58	10.31	11.01	14.72
Δ	-	+3.73	-	+3.71
$AP_{Pedestrian}$	70.93	78.10	80.13	84.07
Δ	-	+7.17	-	+3.94
$AP_{Motorcycle}$	36.16	51.31	48.47	57.65
Δ	-	+15.15	-	+9.18
$AP_{Bicycle}$	9.16	34.02	23.90	41.60
Δ	-	+24.86	-	+17.70
$AP_{Traffic\ cone}$	49.94	66.92	59.14	69.96
Δ	-	+16.98	-	+10.82
$AP_{Barrier}$	56.67	61.26	60.54	61.07
Δ	-	+4.59	-	+0.53
mAP	43.76	52.38	51.60	56.76
Δ	-	+8.62	-	+5.16

Table 4.8: Detailed detection performance of CenterPoint and SemanticVoxels-Center on the nuScenes validation set. Experimental results for both 1 and 10 input LiDAR sweeps are presented. Δ indicates the performance difference between SemanticVoxels and the baseline CenterPoint. Abbreviations: CenterPoint (CP), SemanticVoxels-Center (SV-C), and construction vehicle (Constr. Veh.).

Input	NDS (%)		
sweep	CP	SV-C	Δ
1	46.75	50.82	+4.07
10	61.62	63.93	+2.31

Table 4.9: nuScenes Detectin Score (NDS) of center-based CenterPoint and SemanticVoxels on the nuScenes validation set. Δ indicates the performance difference between SemanticVoxels and the baseline CenterPoint. Abbreviations: CenterPoint (CP) and SemanticVoxels-Center (SV-C).

VRUs and traffic cones gain the greatest. When averaged over all classes, the semantic information yields +8.62% (1 sweep) and +5.16% (10 sweeps) increases in mAP. As expected, center-based SemanticVoxels also improves NDS by 4.07% and 2.31% in two comparisons, as listed in Table 4.9. A comparison of the anchor-based PointPilalrs metrics (Table 4.6 and 4.7) with the center-based CenterPoint metrics (Table 4.8 and 4.9) shows that CenterPoint serves as a stronger baseline, while SemanticVoxels still consistently enhances detection performance. These evaluation results demonstrate that the proposed LiDAR-camera fusion approach is effective and generalizes well across both anchor-based and center-based detection heads.

4.3.5 Qualitative Results

This section provides qualitative results on both KITTI and nuScenes.

KITTI Firstly, to show the improvement of SemanticVoxels over PointPillars, a series of qualitative comparisons on KITTI are presented. Note that the models used for inference here correspond to PointPillars and SemanticVoxels with deep fusion, as evaluated in Table 4.2. To highlight the advantages of SemanticVoxels over PointPillars in critical situation, the prediction results in four exemplary scenes are shown in Figures 4.4 – 4.7. The figures include the camera image, the semantic segmentation result, and the predictions of PointPillars and SemanticVoxels. The ground truth is indicated by green boxes.

The predictions exceeding a confidence threshold of 0.3 are indicated by blue, red, and yellow boxes for the classes car, pedestrian, and cyclist, respectively.

Figure 4.4 shows that the LiDAR-only PointPillars wrongly predicts a parking ticket machine as a pedestrian. Such a false positive detection is however eliminated with the fusion-based SemanticVoxels by incorporating additional semantic information from the image.

In Figure 4.5, PointPillars predicts a false negative pedestrian close to the van's door, while SemanticVoxels does not. In addition, SemanticVoxels successfully classifies two fence pillars that are falsely detected as pedestrians by PointPillars. Note that vans were not categorized into the car class in KITTI. Therefore, it is logical that the trained PointPillars does not detect the van in the scene. SemanticVoxels also handles this case correctly, even though the corresponding point cloud is wrongly augmented due to the incorrect image semantic segmentation.

It can be noticed in Figure 4.6 that the LiDAR-only approach detects a partially occluded cyclist as a pedestrian and predicts a false positive car. Conversely, these failures are well handled by the fusion-based approach. Note that it is extremely challenging to correctly detect the cyclist in this sample using only LiDAR data due to the incompleteness of the spatial information. In this case, the image information provided by the segmentation network proves to be greatly valuable.

Figure 4.7 depicts a complex intersection scenario, where PointPillars struggles to distinguish between pedestrians and narrow vertical objects such as traffic sign and outdoor signage. Furthermore, it fails to detect occluded cyclists and a distant car. In contrast, SemanticVoxels correctly detected all traffic participants, even an unlabeled cyclist in the background, and does not predict any false positives.

Through the qualitative comparisons above, SemanticVoxels demonstrates its effectiveness of fusing multi-modal data for 3D object detection. More precisely, by properly exploiting image semantic information, it achieves superior performance in eliminating false positive pedestrian detections as well as detecting distant and occluded traffic participants.

nuScenes Figure 4.8 depicts object detection results of the anchor-based SemanticVoxels on two scenes from the nuScenes validation set. The input LiDAR point cloud and detected 3D bounding boxes are shown in the top view for a clear presentation of the scene. In addition, only cars and pedestrians objects are displayed. In order to provide a visually clear comparison, the ground truth is indicated by green boxes, while the predictions are indicated by blue and red boxes for cars and pedestrians, respectively.

In both samples, there are very few deviations between the detections and the ground truth. When focusing on car objects, it can be observed that SemanticVoxels detects cars at various distances with outstanding performance. The estimation of bounding box size and orientation is particularly accurate. For pedestrians, SemanticVoxels performs admirably in most cases, but struggles in a few extreme challenging cases, such as heavily occluded pedestrians in dense crowds. Overall, SemanticVoxels demonstrates its superior performance in challenging 360° traffic scenes.

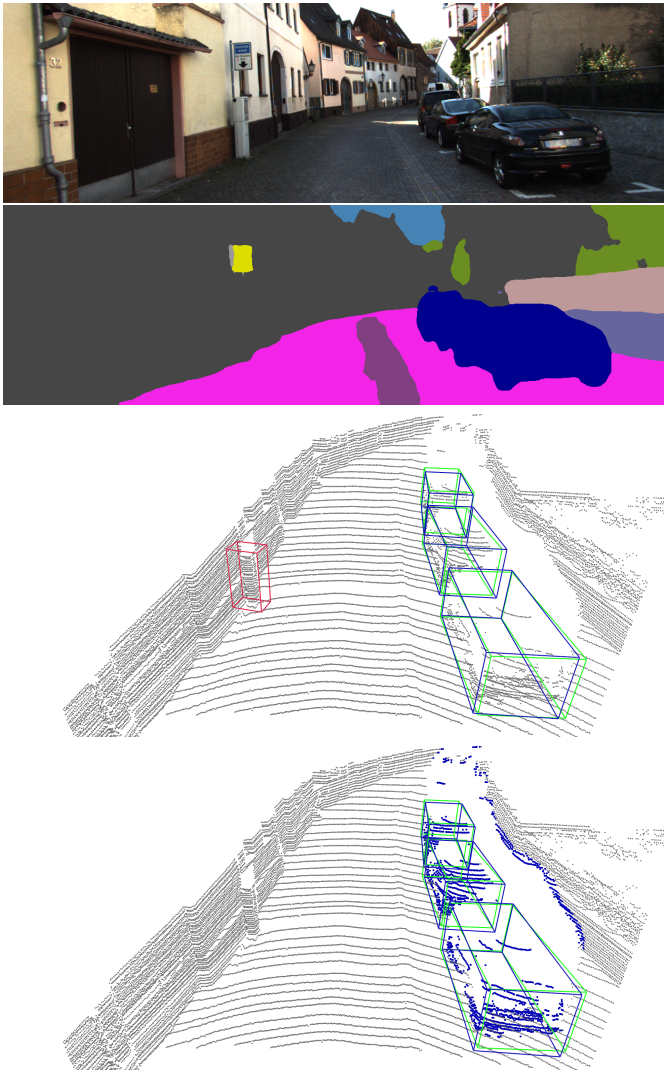


Figure 4.4: Qualitative results on KITTI. From top to bottom: image, semantic segmentation result, detections of PointPillars, and detections of SemanticVoxels. Colormap for 3D bounding boxes: ground truth (green), car (blue), pedestrian (red), and cyclist (yellow). The color scheme for image semantic segmentation is defined in Table 2.1.

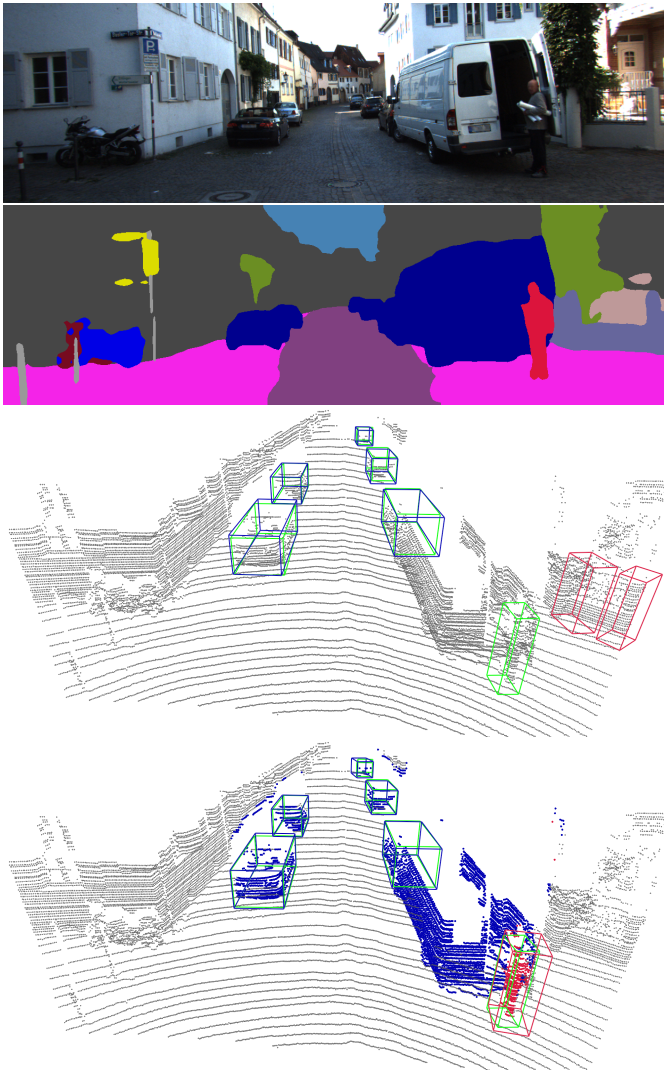


Figure 4.5: Qualitative results on KITTI. From top to bottom: image, semantic segmentation result, detections of PointPillars, and detections of SemanticVoxels. Colormap for 3D bounding boxes: ground truth (green), car (blue), pedestrian (red), and cyclist (yellow). The color scheme for image semantic segmentation is defined in Table 2.1.

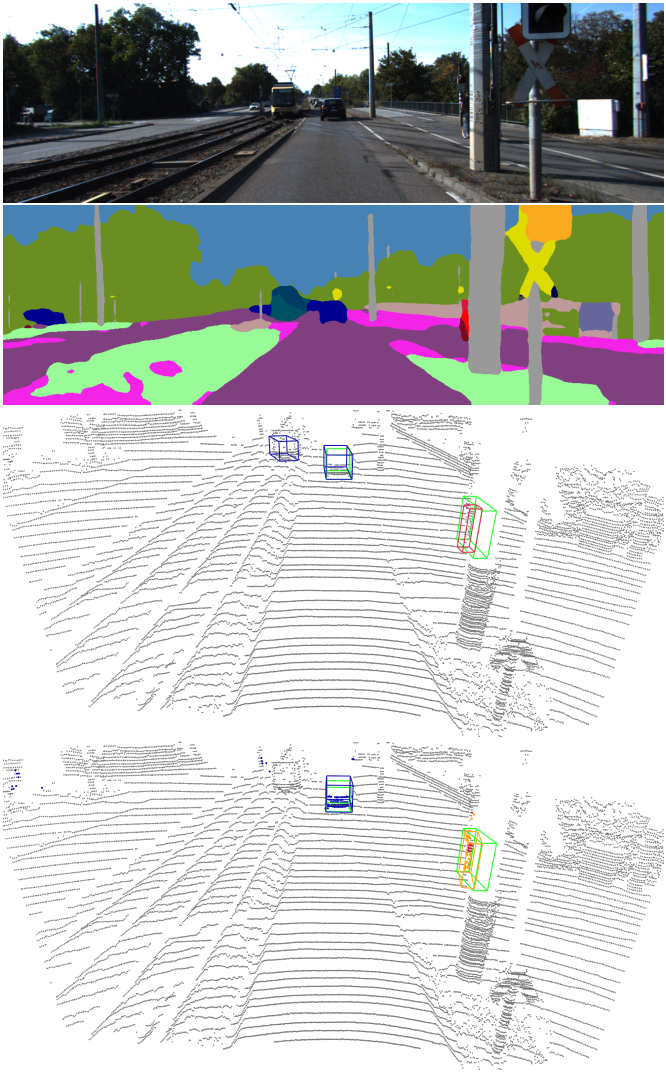


Figure 4.6: Qualitative results on KITTI. From top to bottom: image, semantic segmentation result, detections of PointPillars, and detections of SemanticVoxels. Colormap for 3D bounding boxes: ground truth (green), car (blue), pedestrian (red), and cyclist (yellow). The color scheme for image semantic segmentation is defined in Table 2.1.

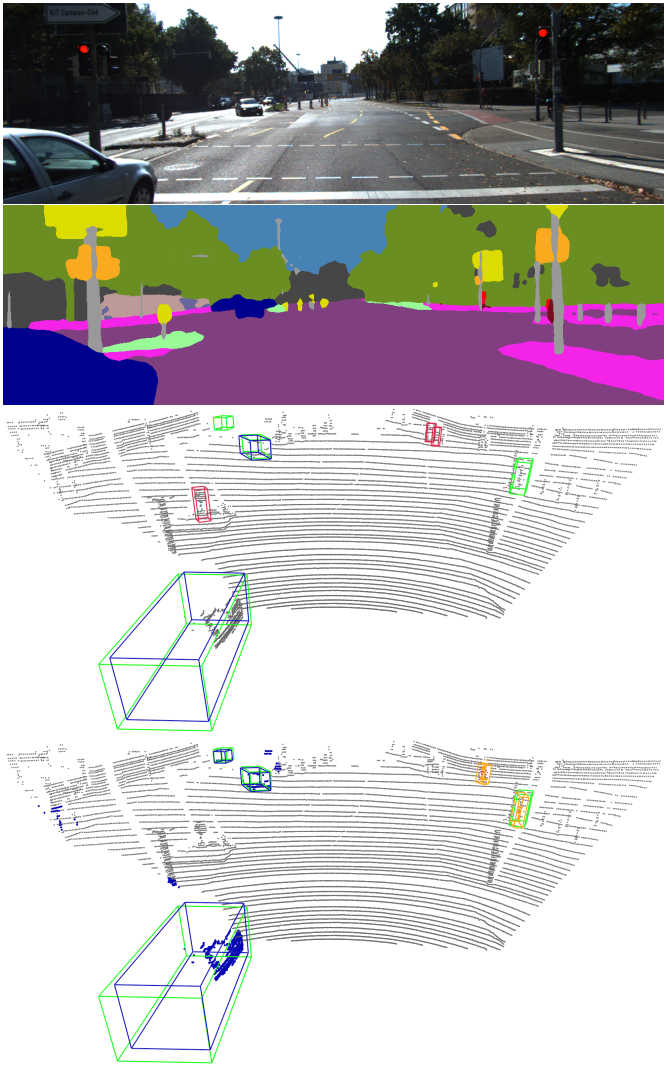
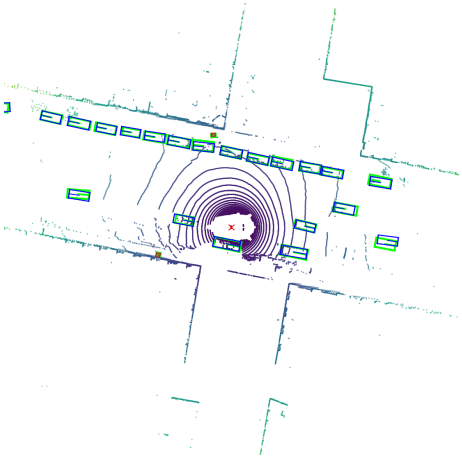
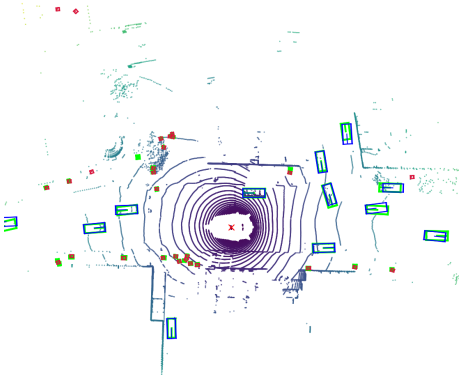


Figure 4.7: Qualitative results on KITTI. From top to bottom: image, semantic segmentation result, detections of PointPillars, and detections of SemanticVoxels. Colormap for 3D bounding boxes: ground truth (green), car (blue), pedestrian (red), and cyclist (yellow). The color scheme for image semantic segmentation is defined in Table 2.1.



(a) Qualitative results on an example frame from the scene 99.



(b) Qualitative results on an example frame from the scene 103.

Figure 4.8: Qualitative results of the anchor-based SemanticVoxels on nuScenes. Note that the network takes 10 accumulated LiDAR sweeps as input. However, only one LiDAR sweep is depicted for clarity. Colormap for 3D bounding boxes: ground truth (green), car (blue), and pedestrian (red).

4.4 Deployment on Edge AI Device

This section presents the deployment of 3D object detection algorithms on a widely used edge AI device, the NVIDIA Jetson AGX Xavier [NVI18]. First, Section 4.4.1 introduces the methods and tools adopted during the deployment. Then, the deployment details and achieved results are presented in Sections 4.4.2 and 4.4.3 for the LiDAR-based PointPillars method and the fusion-based SemanticVoxels method, respectively.

4.4.1 Methods and Tools

As a common practice in machine learning, the DNN models are developed, trained, and evaluated in a Python environment, where PyTorch [PGM+19] has emerged as a popular and widely adopted machine learning library. However, for deployment on embedded systems, where real-time inference is crucial, the DNN model is typically ported to a C++ environment to leverage its performance and efficiency advantages. In this section, an overview of the methods and tools is covered.

Network Conversion To deploy a trained DNN model on an embedded system in a C++ environment, it is necessary to convert it first to a suitable exchange format. A standardized format to represent and share machine learning models across various frameworks or ecosystems is the Open Neural Network Exchange (ONNX) format [BLZ+]. Using a set of ONNX operators, it defines all the necessary operations of a model in a directed computation graph. Converting a PyTorch model to ONNX can be achieved by tracing its execution and structuring a static computation graph based on exemplary input data. The graph serves as an intermediate representation and allows for efficient hardware acceleration with the NVIDIA TensorRT [Van16] SDK (Software Development Kit) for runtime optimized inference. However, tracing cannot cope with dynamic data-dependent control flows, since only the operations that happen during the execution with exemplary input data will be considered. In addition, DNN models often contain sophisticated data processing logics, e.g., data pre-processing and post-precessing, which may also be data-dependent or cannot be easily described by the built-in ONNX operators. Neither the

dynamic control flows nor the data processing logics include trained weights. Therefore, it is a common practice to separate them from the original model and implement them using C++ functions.

As an alternative, PyTorch offers a built-in intermediate representation named TorchScript, which can be created via tracing or scripting. To fairly compare TorchScript with ONNX, tracing is considered to convert the model. In the production environment, the traced model needs to be loaded using the PyTorch C++ API (Application Programming Interface).

Quantization The goal of quantization is to efficiently represent numerical values with a finite number of bits. Reducing precision can be advantageous in an environment with limited computational, memory, and energy resources. In neural networks, single- and half-precision floating-point formats are widely used, known as Float32 and Float16, respectively. There are also fixed-point arithmetic with 8-bit integers, referred to as Int8. It is straightforward to switch from the default Float32 to Float16. However, when going to Int8, additional tuning and calibration processes are required.

For the purpose of simplicity, this work only discusses post-training quantization and does not address quantization-aware training. For the calibration process in Int8 quantization, two calibrators, namely the TensorRT MinMax and Entropy calibrators [Mig17], are considered. The MinMax calibrator measures the maximum absolute value of each layer and constructs an equidistant and symmetric mapping. Similarly, by saturating the activations above a certain threshold, the Entropy calibrator determines a mapping that minimizes the Kullback-Leibler (KL) divergence between the original and quantized activations. Since TorchScript currently does not support Int8 quantization on GPU, only the corresponding TensorRT variant is evaluated.

Further Tools The Robot Operating System (ROS) [QCG+09] is commonly used as a framework on the target platform. ROS provides useful tools for streaming sensor data, communicating across nodes, and visualizing sensor data and DNN outputs. To convert data from public datasets into a ROS compatible format, the tools kitti2bag [Kre] and nuscenes2bag [Com] are used. Two commonly used libraries, OpenCV [Bra00] and Point Cloud Library (PCL) [RC11], are considered for computationally efficient preprocessing of

image and point cloud data in C++, respectively. In addition, many operations in the pre- and post-processing are implemented with Compute Unified Device Architecture (CUDA) [Lue08] for accelerated parallel processing.

4.4.2 PointPillars

As a key building block of SemanticVoxels, this section focuses on the deployment of the LiDAR-based detector PointPillars. Additionally, optimization techniques are studied in order to improve the runtime.

Deployment Details PointPillars [LVC+19] is a LiDAR-based 3D object detection network that achieves state-of-the-art performance on public benchmarks with real-time inference speed. The architecture and detail of PointPillars are introduced in Section 2.2.4. To deploy PointPillars on the target embedded platform, it is divided into five blocks: pre-processing, Pillar Feature Network (PFN), scatter pillar features to the BEV grid, backbone and detection head, and post-processing. The deployment details of the above blocks are listed in Table 4.10.

In the pre-processing stage, the input point cloud is first discretized into a set of pillars. Then, each point in a pillar is further decorated with additional features. All these operations are deployed efficiently with parallel processing on the GPU using the CUDA library.

The PFN is a combination of a simplified PointNet [QSMG17] and a max operation, which learns high-level pillar features. It is deployed using a TensorRT or TorchScript network graph. The learned pillar features are then scattered to the BEV grid, where the scatter operation is parallelized by a C++ function with CUDA.

The backbone and detection head consist of common convolutional layers. Thus, during deployment, they are jointly realized by a TensorRT or TorchScript network graph. To get the final 3D bounding boxes, the output of the detection head is decoded using the anchor information. Then, NMS is applied to select the best predictions from overlapping boxes. These two logical operations are implemented with CUDA-based C++ functions to leverage GPU parallelism.

Block	Deployment
Pre-processing - Discretize point cloud to pillars - Point feature decoration	C++ function with CUDA
Pillar feature network	TensorRT or TorchScript network graph
Scatter pillar features to BEV grid	C++ function with CUDA
Backbone and detection head	TensorRT or TorchScript network graph
Post-processing - Box decoding - Non-maximum suppression	C++ function with CUDA

Table 4.10: Deployment detail of PointPillars on the target platform.

Experimental Setup For the deployment study, PointPillars [LVC+19] is trained and evaluated on the KITTI [GLU12] 3D object detection dataset. For the experimental evaluation, the samples are splitted into train and validation sets, as in Section 4.3. The 3D object detection performance is evaluated using Average Precision (AP) in moderate level with Intersection over Union (IoU) thresholds of 0.7 for Car and 0.5 for Pedestrian and Cyclist. To construct and train the model, the same KITTI-PointPillars configuration as in the OpenPCDet codebase [Tea] is used.

For the deployment on the target platform, the pipeline is implemented based on Autoware [KTM+18] by developing additional features. Unless otherwise specified, runtime measurements are performed on the target platform with the MAXN power mode using the KITTI raw data sequence 0095 with 236 frames [GLSU13]. To capture steady-state computation, the runtime is averaged starting from the tenth frame to account for GPU warm-up on the target platform.

Runtime Analysis The runtime results for both PFN and Backbone & Detection Head (named 2D CNN for short) when deployed with TensorRT and TorchScript using various quantization techniques are presented in Table 4.11. When comparing the runtime of the model deployed using TensorRT and

Network	Quantization	TensorRT	TorchScript
PFN	Float32	30 ms	21 ms
	Float16	26 ms	19 ms
2D CNN	Float32	46 ms	82 ms
	Float16	16 ms	31 ms
	Int8	14 ms	-

Table 4.11: Runtime of PFN and Backbone & Detection Head (named 2D CNN for short) deployed with TensorRT and TorchScript using various quantization techniques.

TorchScript, PFN and 2D CNN show opposite trends. For 2D CNN, TensorRT achieves significantly better runtime optimization. In contrast, TorchScript optimizes the runtime of PFN better. One possible explanation could be slight advantages of TensorRT for convolutional layers and TorchScript for fully connected layers. When comparing the quantization techniques, Float16 offers a significant speed improvement for PFN and 2D CNN across both tools, whereas Int8 only provides additional minor improvements for 2D CNN. Note that Int8 quantization is not considered for PFN, since its inputs are 3D coordinates with an approximate range of $\pm 10^2$ m and an accuracy of 10^{-2} m. Using Int8 quantization would result in significant information loss.

Performance Analysis Based on the runtime analysis, it can be concluded that the combination of PFN with TorchScript and 2D CNN with TensorRT achieves the lowest runtime. Therefore, this setup is considered when further studying the object detection performance of the model deployed using different quantization techniques. The results of the performance evaluation for three classes as well as the runtime for the entire PointPilalrs pipeline are shown in Table 4.12. When going down from Float32 to Float16 precision for PFN or 2D CNN, there are no significant performance changes in terms of AP for all three categories. When using Int8 quantization for 2D CNN, the performance drops considerably for both calibration methods, where the reduction is even more severe for the entropy calibrator. This may be due to the fact that the used pillar feature map representation and subsequent activations require a higher arithmetic range and precision in order to preserve the essential geometric information. Further, when considering the runtime of the complete PointPillars

PFN quantization	2D CNN quantization	AP_{3D} Car	Ped.	Cyc.	Pipeline runtime
Float32	Float32	78.40	51.41	62.81	72 ms
	Float16	78.30	51.31	62.89	43 ms
Float32	Int8, minmax	71.04	48.00	55.94	40 ms
	Int8, entropy	68.91	22.03	29.66	
Float16	Float32	78.40	51.46	62.92	71 ms
	Float16	78.31	51.39	62.97	41 ms
Float16	Int8, minmax	70.99	47.65	56.42	38 ms
	Int8, entropy	69.37	21.86	29.53	

Table 4.12: Performance evaluation of PointPillars with PFN and 2D CNN combinations using various quantization techniques on the KITTI validation set. Abbreviations: Pedestrian (Ped.) and Cyclist (Cyc.).

detection pipeline, it is evident that deploying PFN and 2D CNN using Float16 precision with TorchScript and TensorRT, respectively, offers the best trade-off between detection and runtime performance. Hence, it is referred to as the optimal variant in the following study. Note that due to compatibility and code simplicity, a combination of two frameworks might not be the best choice for deployment on a real system, even if it has the lowest combined runtime.

Further Study on PFN As observed in the preceding analysis, the optimal variant requires a total runtime of 41 ms, while the single PFN costs 19 ms, being the bottleneck in the deployment. Thus, additional experiments are conducted to investigate the potential of further reducing the runtime of PFN by modifying the network input. This is achieved by adjusting two parameters of PFN, namely the number of non-empty pillars P, and the number of points per pillar N. Besides $P = 16000$ and $N = 32$ used in the previous experiments, $P = 12000$ and $N \in \{24, 16\}$ are additionally considered. Both the runtime of PFN and the overall detection performance are measured. As shown in Table 4.13, keeping the number of points while lowering the number of pillars from 16000 to 12000 does not introduce notable change on the performance metrics, while it improves the runtime considerably from 19 ms to 14 ms for

the case with 32 points per pillar. Additional runtime boost can be achieved by decreasing the number of points per pillar without remarkable performance loss. According to this study, the setup with 12000 pillars and 24 points per pillar appears to be the most appropriate for deploying PFN with a runtime of 9 ms.

Pillar number	Point number	AP_{3D} Car	Ped.	Cyc.	PFN runtime
16000	32	78.31	51.39	62.97	19 ms
16000	24	78.28	50.68	62.20	12 ms
16000	16	78.14	50.81	61.33	10 ms
12000	32	78.29	51.37	62.98	14 ms
12000	24	78.25	51.92	62.19	9 ms
12000	16	78.17	49.02	61.38	8 ms

Table 4.13: Performance and runtime evaluation of PFN with different parameters. Abbreviations: Pedestrian (Ped.) and Cyclist (Cyc.).

Summary Based on extensive studies, it can be concluded that deploying PointPillars using separate PFN and 2D CNN network graphs on Float16 precision with Torchscript and TensorRT, respectively, is the optimal solution for the deployment on the target platform. Moreover, the input parameters of PFN have a significant impact on the runtime, where the parameters $P = 12000$ and $N = 24$ appear to be the most suitable in this study. Finally, Table 4.14 reports the detailed runtime of the optimal PointPillars pipeline using various power modes on the target platform.

4.4.3 SemanticVoxels

In Section 4.4.2, the PointPillars object detection pipeline has been successfully deployed, and the most appropriate optimization methods have been identified to achieve real-time performance on the NVIDIA Jetson platform. In order to deploy SemanticVoxels, this pipeline is extended by adding additional functionalities.

Block	MAXN	30W	15W	10W
Pre-processing	4 ms	6 ms	8 ms	14 ms
PFN	9 ms	14 ms	21 ms	52 ms
Scatter	1 ms	2 ms	2 ms	4 ms
2D CNN	16 ms	25 ms	32 ms	71 ms
Post-processing	1 ms	1ms	2 ms	2 ms
Total	31 ms	48 ms	65 ms	143 ms

Table 4.14: Detailed runtime of each block in the optimal PointPillars detection pipeline using different power modes.

Deployment Details Based on the architecture introduced in Section 4.2, SemanticVoxels is divided into multiple blocks during deployment to facilitate efficient data processing and improve pipeline performance. Each block is deployed using appropriate technique. These blocks and their deployment details are presented in Table 4.15.

The image semantic segmentation is implemented based on the Bonnetal [MS19] framework as a standalone ROS node. The trained DDRNet-23-slim model, introduced in Section 4.3.2, is deployed using TensorRT with half-precision. The subsequent projection operations and augmentation of the input point cloud are performed using C++ functions.

During the pre-processing, the augmented point cloud is first discretized into pillars and then into voxels. Spatial and semantic information are then encoded in the generated pillars and voxels, respectively. These operations are realized by C++ functions with CUDA on the GPU.

Based on the deployment study of PointPillars, the spatial feature network is deployed using a TorchScript network graph. The scatter operation is implemented identically as in PointPillars. The semantic feature network and the fusion of spatial and semantic features are integrated into the backbone and detection head, which are deployed using a TensorRT network graph. In the post-processing, the implementation of box decoding and NMS are reused from the PointPillars pipeline.

Block	Deployment
Image semantic segmentation	TensorRT network graph
Point cloud augmentation	C++ function
Pre-processing - Discretize point cloud to pillars and voxels - Point feature decoration	C++ function with CUDA
Spatial feature network	TorchScript network graph
Scatter spatial features to BEV grid	C++ function with CUDA
Semantic feature network Backbone with fusion and detection head	TensorRT network graph
Post-processing - Box decoding - Non-maximum suppression	C++ function with CUDA

Table 4.15: Deployment detail of the SemanticVoxels pipeline on the target platform.

Experimental Setup The anchor-based SemanticVoxels is deployed on both the KITTI [GLU12] object detection dataset and data collected by an experimental vehicle from Opel (referred as Opel data). Note that the deployed models adopt early fusion and do not apply label remapping or elevation correction.

When deploying SemanticVoxels on the KITTI data, the image semantic segmentation and object detection networks are trained as described in Section 4.3.2 and Section 4.3.3, respectively. All modules in the SemanticVoxels pipeline run online on the target platform. To measure the runtime performance and achieve a fair comparison, the same method used in the PointPillars pipeline is applied. In addition, the runtime of the image semantic segmentation node is measured similarly.

When switching to the Opel data, the lack of ground-truth annotations should be taken into account. Considering that the point cloud data in the nuScenes [CBL+20] and Opel datasets were collected using LiDARs of similar types and with the same number of channels, the SemanticVoxels network is first trained on nuScenes and then deployed on the Opel data. For the training, the

Figure 4.9: A sample image in the Opel data and its semantic segmentation mask predicted by DeepLabv3+. The hood area of the ego car is blacked out in the prediction image.

setup presented in Section 4.3.3 is followed. Note that the network takes one LiDAR sweep as input and is only trained on the car, pedestrian, bicycle and motorcycle classes.

Without semantic segmentation labels, DDRNet-23-slim cannot be trained on Opel data. Furthermore, when trained on Cityscapes or KITTI, the predictions of DDRNet-23-slim models on the Opel data turn out to be unsatisfactory, which prevents successful deployment with this semantic segmentation network. Thus, a more powerful pre-trained DeepLabv3+ model is used for offline semantic segmentation. Its generalization results on Opel data are surprisingly good. A sample image in the Opel data and its segmentation mask predicted by DeepLabv3+ is shown in Figure 4.9. The lack of generalization ability is presumed to be a downside for networks tuned for real-time performance, such as DDRNet-23-slim. After obtaining segmentation results from DeepLabv3+, point cloud augmentation is also performed offline. When inferring online on

the target platform, each time the SemanticVoxels pipeline takes an augmented point cloud as input and outputs detected 3D objects.

Runtime Results When inferring on the KITTI data, it takes in total 58 ms for the whole SemanticVoxels pipeline, of which the image semantic segmentation part costs 20 ms. On the Opel data, the online object detection pipeline achieves a runtime of 46 ms on the Jeston AGX Xavier platform.

Qualitative Results Figure 4.10 shows the object detection results of the SemanticVoxels pipeline deployed on the target platform on a KITTI data sample. The deployed algorithm correctly detects different classes of traffic participants and accurately estimates their sizes, locations and headings.

Figure 4.11 depicts the object detection results of the SemanticVoxels pipeline on an Opel data sample. Despite being trained on the nuScenes dataset, the deployed network successfully detects and classifies all relevant objects in the unseen scene captured in Rüsselsheim, Germany. This demonstrates the outstanding generalization ability of SemanticVoxels.

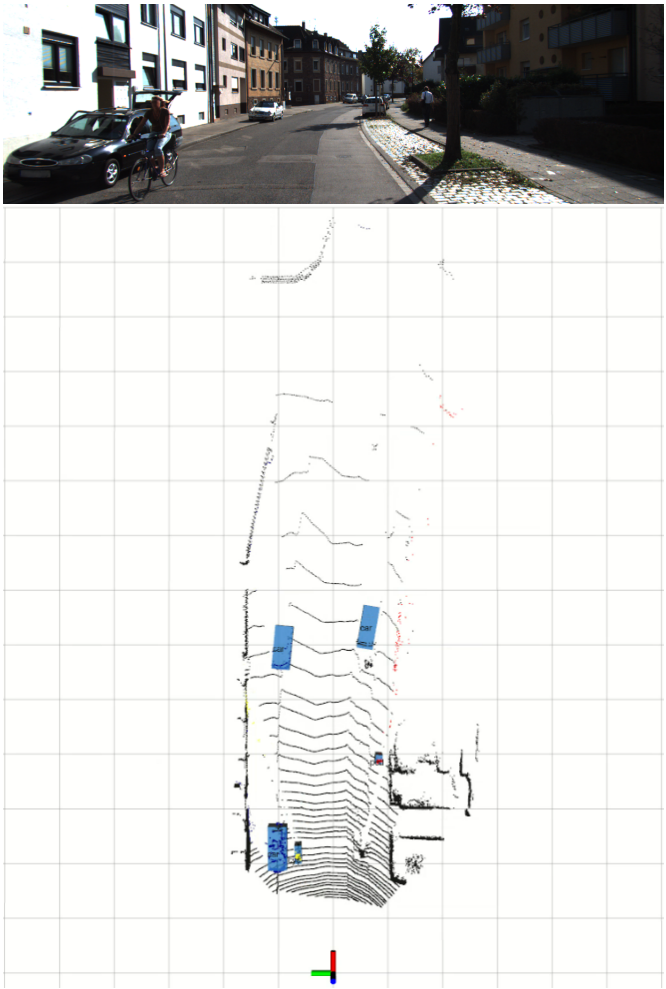


Figure 4.10: Detections on the KITTI data. The predicted 3D bounding boxes and point cloud data are visualized on the top view in Rviz on the Jeston AGX Xavier platform.

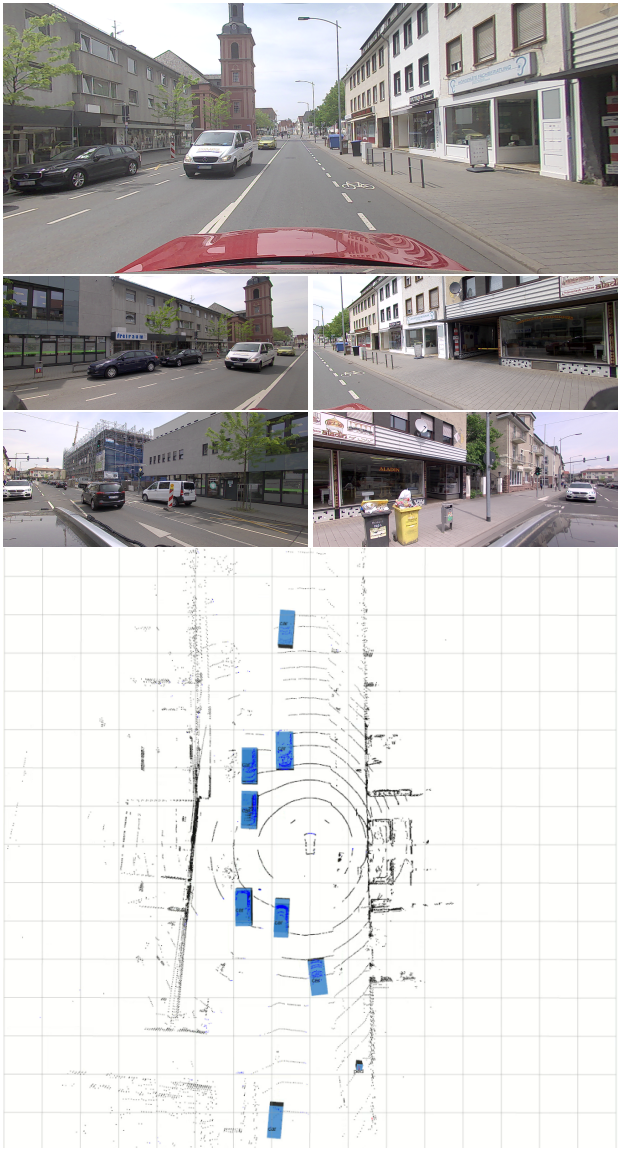


Figure 4.11: Detections on the Opel data. The predicted 3D bounding boxes and point cloud data are visualized on the top view in Rviz on the Jeston AGX Xavier platform.

5 Conclusions and Future Work

5.1 Conclusions

This thesis has presented deep learning approaches for semantic environment perception for automated driving using onboard sensor data. Specifically, this includes an end-to-end semantic grid map estimation approach for static environment perception based on sparse LiDAR data, and an effective LiDAR-camera fusion-based 3D object detection approach for perceiving dynamic traffic participants. In addition, the latter model has been successfully deployed on an embedded edge AI device, achieving real-time inference speed and demonstrating outstanding generalization on real-world data.

The semantic grid map estimation approach, named PillarSegNet, has been proposed in Chapter 3. It takes a sparse single-sweep LiDAR point cloud as input and predicts a dense top-view semantic grid map. PillarSegNet adopts a pillar feature network to directly learn features from the input point cloud, thereby avoiding potential information loss caused by hand-crafted feature engineering. The proposed method has been trained and evaluated on the SemanticKITTI [BGM+19] dataset, using sparse and dense ground-truth semantic grid maps generated from a single point-wise labelled point cloud and multiple superimposed scans, respectively. Experimental results have demonstrated that PillarSegNet outperforms a state-of-the-art grid map method [BWJ+20] by a large margin in terms of mIoU, highlighting the effectiveness of PillarSegNet and the superiority of its learned pillar features. Furthermore, it has been proven that training PillarSegNet with dense ground truth yields notable performance improvements. In addition to the learned pillar features, PillarSegNet has been further extended by 2D or 3D occupancy features, obtained via model-based ray-casting, which results in additional performance improvements. It has been found that aggregating the pillar features and 2D occupancy features offers the best trade-off between performance and computational efficiency for practical applications.

The LiDAR-camera fusion-based 3D object detection approach, named SemanticVoxels, has been proposed in Chapter 4. It effectively fuses LiDAR spatial and image semantic information on the BEV. To lift image semantic features originally in the image view into the 3D space, SemanticVoxels projects 3D LiDAR points onto the image plane and augments them with image semantic segmentation output. To preserve their spatial distribution in the height direction, the lifted image features are encoded in voxels, which are further fused with LiDAR spatial features encoded in pillars. SemanticVoxels has been trained and evaluated on both the KITTI [GLU12] and nuScenes [CBL+20] datasets. Experimental results have shown that SemanticVoxels significantly improves 3D object detection performance over the LiDAR-only baseline, especially on challenging object categories, thus reflecting the advantages of exploiting image semantic information. Under simulated calibration errors, the performance of SemanticVoxels drops only slightly while still being much better than the LiDAR-only baseline. These findings demonstrate the robustness and practical value of SemanticVoxels. Besides, SemanticVoxels has proven its effectiveness and good generalization ability for both anchor-based and center-based detection heads. Finally, qualitative results have illustrated superior performance of SemanticVoxels in eliminating false positive detections as well detecting distant and occluded traffic participants.

In Chapter 4.4, the deployment of the LiDAR-only detector PointPillars [LVC+19] and the fusion-based detector SemanticVoxels on an edge AI platform has been presented. To achieve real-time inference performance, various optimization techniques have been studied. It has been found that deploying fully connected layers with TorchScript and convolutional layers with TensorRT achieves the best runtime. Moreover, quantization significantly reduces the runtime and Float16 has been identified as the best trade-off between detection performance and inference speed. In the end, a runtime of 31 ms and 58 ms has been achieved for PointPillars and SemanticVoxels on the KITTI data, respectively. SemanticVoxels has been further deployed on unseen Opel data and good object detection results have been achieved, demonstrating its outstanding generalization ability.

5.2 Future Work

Despite the above promising results, several research topics remain to be explored in the future. SemanticVoxels relies on LiDAR point clouds as queries to lift image features back to 3D space. This process provides accurate positional information for image features but also leads to a potential loss of image information due to the sparsity of point clouds. Very recently, camera-only approaches that convert 2D image features to 3D and obtain a BEV representation have advanced and shown promising results [HHZD21, LWL+22]. Therefore, it may be beneficial to adopt those advanced view transformation methods in SemanticVoxels. However, they require greater computation and may be challenging to deploy in real-time on embedded platform.

In this thesis, semantic grid map estimation and 3D object detection are performed separately using PillarSegNet and SemanticVoxels. Both approaches represent features in the BEV, and the final predictions are also done in the top view. Hence, it is worth developing a multi-task DNN to jointly perform semantic grid map estimation and 3D object detection. Additional loss term that measures the consistency between the outputs of the two tasks may be applied to better supervise the training.

Lastly, both PillarSegNet and SemanticVoxels only rely on single frame sensor data and do not consider the time domain. Temporal information provides more clues about the surrounding environment, and is highly likely to enhance the performance and robustness of the proposed approaches. In SemanticVoxels and PillarSegNet, features are represented in the BEV. By utilizing ego-motion information, BEV features from adjacent timestamps can be efficiently aligned and fused.

Bibliography

[AAB+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. arXiv: `https://arxiv.org/abs/1603.04467`, 2016. (visited on 02/03/2024).

[AMM+18] Hassan Alhaija, Siva Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes. *International Journal of Computer Vision (IJCV)*, 126:961 – 972, 2018.

[AW87] John Amanatides and Andrew Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. *Eurographics*, 1987.

[BGM+19] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jürgen Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9296–9306, Seoul, South Korea, 2019.

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[BLR+21] Frank Bieder, Maximilian Link, Simon Romanski, Haohao Hu, and Christoph Stiller. Improving Lidar-Based Semantic Segmentation of Top-View Grid Maps by Learning Features in Complementary Representations. *IEEE 24th International Conference on Information Fusion (FUSION)*, pages 64–70, 2021.

[BLZ+] Junjie Bai, Fang Lu, Ke Zhang, et al. ONNX: Open Neural Network Exchange. GitHub repository: `https://github.com/onnx/onnx`. (accessed on 06/19/2022).

[Bra00] G. Bradski. The OpenCV Library. In *Dr. Dobb's Journal of Software Tools*, 2000.

[BWJ+20] Frank Bieder, Sascha Wirges, Johannes Janosovits, Sven Richter, Zheyuan Wang, and Christoph Stiller. Exploiting Multi-Layer Grid Maps for Surround-View Semantic Segmentation of Sparse LiDAR Data. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1892–1898, Las Vegas, NV, USA, 2020.

[BWRS21] Frank Bieder, Sascha Wirges, Sven Richter, and Christoph Stiller. Fusion of Sequential LiDAR Measurements for Semantic Segmentation of Multi-layer Grid Maps. *tm - Technisches Messen*, 88(6):352–360, 2021.

[cAL+16] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9901, pages 424–432, Athens, Greece, 2016.

[CBL+20] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, Seattle, WA, USA, 2020.

[Cho17] François Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, Honolulu, HI, USA, 2017.

[CKZ+15] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3D Object Proposals for Accurate Object Class Detection. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, pages 424–432, Montreal, Quebec, Canada, 2015.

[CKZ+16] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3D Object Detection for Autonomous Driving. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2147–2156, Las Vegas, NV, USA, 2016.

[CMW+17] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D Object Detection Network for Autonomous Driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6526–6534, Honolulu, HI, USA, 2017.

[Com] Vincenzo Comito. Nuscenes2bag. GitHub repository: `https://github.com/clynamen/nuscenes2bag`. (accessed on 07/09/2021).

[COR+] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, Las Vegas, NV, USA.

[CPK+15] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.

[CPK+18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. DeepLab: Semantic Image Segmentation With Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.

[CPSA17] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic

Image Segmentation. arXiv: https://arxiv.org/pdf/1706.05587, 2017. (visited on 03/11/2022).

[CZP+18] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-Decoder With Atrous Separable Convolution for Semantic Image Segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 11211, pages 833–851, Munich, Germany, 2018. Springer.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, Miami, Florida, USA, 2009.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research (JMLR)*, 12(7), 2011.

[DM12] S.B. Damelin and W. Miller. *The Mathematics of Signal Processing*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2012.

[Elf89] Alberto Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, 22(6):46–57, 1989.

[ERW+17] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3Deep: Fast Object Detection in 3D Point Clouds using Efficient Convolutional Neural Networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1355–1361, Singapore, 2017.

[EVGW+10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision (IJCV)*, 88:303–338, 2010.

[EWL+18] Özgür Erkent, Christian Wolf, Christian Laugier, David Sierra Gonzalez, and Victor Romero Cano. Semantic Grid Estimation with a Hybrid Bayesian and Deep Neural Network Approach. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 888–895, Madrid, Spain, 2018.

[FCH+20] Juncong Fei, Wenbo Chen, Philipp Heidenreich, Sascha Wirges, and Christoph Stiller. SemanticVoxels: Sequential Fusion for 3D Pedestrian Detection using LiDAR Point Cloud and Semantic Segmentation. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 185–190, Karlsruhe, Germany, 2020.

[Fei19] Juncong Fei. 3D Object Detection and Tracking on Multi-sensor Data with Deep Learning. Master's thesis, Institute for Information Processing Technologies (ITIV), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 2019.

[FPH+21] Juncong Fei, Kunyu Peng, Philipp Heidenreich, Frank Bieder, and Christoph Stiller. PillarSegNet: Pillar-based Semantic Grid Map Estimation using Sparse LiDAR Data. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 838–844, Nagoya, Japan, 2021.

[FXW+21] Lue Fan, Xuan Xiong, Feng Wang, Nai long Wang, and Zhaoxiang Zhang. RangeDet: In Defense of Range View for LiDAR-based 3D Object Detection. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2898–2907, Montreal, QC, Canada, 2021.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GDH+20] Runzhou Ge, Zhuangzhuang Ding, Yihan Hu, Yu Wang, Sijia Chen, Li Huang, and Yuan Li. AFDet: Anchor Free One Stage 3D Object Detection. arXiv: `https://arxiv.org/abs/2006.12671`, 2020. (visited on 03/24/2023).

[Gir15] Ross Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, Santiago, Chile, 2015.

[GLSU13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 32(11):1231 – 1237, 2013.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *2012 IEEE Conference on Computer Vision and Pattern*

Recognition (CVPR), pages 3354–3361, Providence, RI, USA, 2012.

[GLY+24] Shuo Gu, Jiacheng Lu, Jian Yang, Cheng-Zhong Xu, and Hui Kong. Dense Top-View Semantic Completion with Sparse Guidance and Online Distillation. *IEEE Transactions on Intelligent Vehicles*, 9(1):481–491, 2024.

[Gol18] Pete Goldin. 10 Advantages of Autonomous Vehicles. In *ITS-digest*, 2018. URL: `https://www.itsdigest.com/10-advantages-autonomous-vehicles`. (visited on 03/11/2023).

[HHZD21] Junjie Huang, Guan Huang, Zheng Zhu, and Dalong Du. BEVDet: High-performance Multi-camera 3D Object Detection in Bird-Eye-View. arXiv: `https://arxiv.org/2112.11790v3`, 2021. (visited on 12/18/2022).

[Hin12] Geoffrey Hinton. Lecture 6e - RMSProp: Divide the gradient by a running average of its recent magnitude. In *Coursera Course: Neural Networks for Machine Learning*, 2012. URL: `http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf`. (visited on 10/14/2023).

[HKOB10] Florian Homm, Nico Kaempchen, Jeff Ota, and Darius Burschka. Efficient Occupancy Grid Computation on the GPU with Lidar and Radar for Road Boundary Detection. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1006–1013, La Jolla, CA, USA, 2010.

[HPSJ21] Yuanduo Hong, Huihui Pan, Weichao Sun, and Yisong Jia. Deep Dual-resolution Networks for Real-time and Accurate Semantic Segmentation of Road Scenes. arXiv: `https://arxiv.org/abs/2101.06085`, 2021. (visited on 02/17/2022).

[HYX+20] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Agathoniki Trigoni, and A. Markham. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11105–11114, Seattle, WA, USA, 2020.

[HZHR20] Peiyun Hu, Jason Ziglar, David Held, and Deva Ramanan. What You See Is What You Get: Exploiting Visibility for 3D Object Detection. In *2020 IEEE/CVF Conference on Computer Vision*

and Pattern Recognition (CVPR), pages 11001–11009, Seattle, WA, USA, 2020.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning (ICML)*, volume 37, pages 448–456, Lille, France, 2015.

[KB15] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.

[KML+18] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L. Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, Madrid, Spain, 2018.

[Kre] Tomas Krejci. Kitti2bag. GitHub repository: `https://github.com/tomas789/kitti2bag`. (accessed on 07/09/2021).

[KTM+18] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296, Porto, Portugal, 2018.

[Kü20] Kümmerle, Julius Valentin. *Multimodal Sensor Calibration with a Spherical Calibration Target*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2020.

[LAE+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot Multibox Detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 9905, pages 21–37, Amsterdam, Netherlands, 2016. Springer.

[LD20] Hei Law and Jia Deng. CornerNet: Detecting Objects as Paired Keypoints. *International Journal of Computer Vision (IJCV)*, 128(3):642 – 656, 2020.

[LGG+17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, Venice, Italy, 2017.

[LGY+23] Yinhao Li, Zheng Ge, Guanyi Yu, Jinrong Yang, Zengran Wang, Yukang Shi, Jianjian Sun, and Zeming Li. BEVDepth: Acquisition of Reliable Depth for Multi-view 3D Object Detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 1477–1485, Washington, DC, USA, 2023.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA, 2019. OpenReview.net.

[Li17] Bo Li. 3D Fully Convolutional Network for Vehicle Detection in Point Cloud. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1513–1518, Vancouver, Canada, 2017.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, Boston, MA, USA, 2015.

[LTA+23] Zhijian Liu, Haotian Tang, Alexander Amini, Xingyu Yang, Huizi Mao, Daniela Rus, and Song Han. BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2774–2781, London, UK, 2023.

[Lue08] David Luebke. CUDA: Scalable Parallel Programming for high-performance Scientific Computing. In *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 836–838, Paris, France, 2008.

[LVC+19] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast Encoders for Object Detection From Point Clouds. In *2019 IEEE Conference*

on Computer Vision and Pattern Recognition (CVPR), pages 12689–12697, Long Beach, CA, USA, 2019.

[LvdMD18] Chenyang Lu, M. J. G. van de Molengraft, and Gijs Dubbelman. Monocular Semantic Occupancy Grid Mapping With Convolutional Variational Encoder-Decoder Networks. *IEEE Robotics and Automation Letters*, 4:445–452, 2018.

[LWL+22] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Yu Qiao, and Jifeng Dai. BEVFormer: Learning Bird's-Eye-View Representation from Multi-camera Images via Spatiotemporal Transformers. In *European Conference on Computer Vision (ECCV)*, volume 13669, pages 1–18, Tel Aviv, Israel, 2022. Springer.

[LXY+22] Tingting Liang, Hongwei Xie, Kaicheng Yu, Zhongyu Xia, Zhiwei Lin, Yongtao Wang, Tao Tang, Bing Wang, and Zhi Tang. BEVFusion: A Simple and Robust LiDAR-Camera Fusion Framework. In *Neural Information Processing Systems (NeurIPS)*, volume 35, pages 10421–10434, New Orleans, LA, USA, 2022.

[LZX16] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle Detection from 3D Lidar Using Fully Convolutional Network. In *Robotics: Science and Systems*, Ann Arbor, Michigan, USA, 2016.

[MAFK16] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3D Bounding Box Estimation Using Deep Learning and Geometry. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5632–5640, Honolulu, HI, USA, 2016.

[Mig17] Szymon Migacz. 8-bit Inference with TensorRT. In *GPU Technology Conference*, San Jose, CA, USA, 2017. URL: `https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf`. (visited on 10/17/2023).

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[MS15] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D Convolutional Neural Network for Real-time Object Recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, Hamburg, Germany, 2015.

[MS19] Andres Milioto and Cyrill Stachniss. Bonnet: An Open-Source Training and Deployment Framework for Semantic Segmentation in Robotics using CNNs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7094–7100, Montreal, QC, Canada, 2019.

[MVBS19] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4213–4220, Macau, SAR, China, 2019.

[NHH15] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning Deconvolution Network for Semantic Segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1528, Santiago, Chile, 2015.

[NOBK17] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kontschieder. The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5000–5009, Venice, Italy, 2017.

[NVI18] NVIDIA Corporation. Jetson AGX Xavier. URL: `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/`, 2018. (visited on 06/17/2022).

[PAD+17] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H. Ang. Perception, Planning, Control, and Coordination for Autonomous Vehicles. In *Machines*, volume 5, 2017.

[PD18] Andreas Pfeuffer and Klaus Dietmayer. Optimal Sensor Data Fusion Architecture for Object Detection in Adverse Weather Conditions. In *2018 21st International Conference on Information Fusion (FUSION)*, pages 1–8, Cambridge, UK, 2018.

[PF20] Jonah Philion and Sanja Fidler. Lift, Splat, Shoot: Encoding Images from Arbitrary Camera Rigs by Implicitly Unprojecting to 3D. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 12359, pages 194–210, Glasgow, UK, 2020. Springer.

[PFY+22] Kunyu Peng, Juncong Fei, Kailun Yang, Alina Roitberg, Jiaming Zhang, Frank Bieder, Philipp Heidenreich, Christoph Stiller, and Rainer Stiefelhagen. MASS: Multi-Attentional Semantic Segmentation of LiDAR Data for Dense Top-View Understanding. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):15824–15840, 2022.

[PGA+16] Cristiano Premebida, Luis Garrote, Alireza Asvadi, A. Pedro Ribeiro, and Urbano Nunes. High-resolution LIDAR-based depth mapping using bilateral filter. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2469–2474, Rio de Janeiro, Brazil, 2016.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, Vancouver, BC, Canada, 2019.

[Pus17] Pusztai, Zoltan and Hajder, Levente. Accurate Calibration of LiDAR-Camera Systems Using Ordinary Boxes. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 394–402, Venice, Italy, 2017.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. ROS: an open-source Robot Operating System. In *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, volume 3, page 5, Kobe, Japan, 2009.

[Qia99] Ning Qian. On the Momentum Term in Gradient Descent Learning Algorithms. *Neural Networks*, 12(1):145–151, 1999.

[QLW+18] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 918–927, Salt Lake City, UT, USA, 2018.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision*

and Pattern Recognition (CVPR), pages 77–85, Honolulu, HI, USA, 2017.

[QYSG17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5099–5108, Long Beach, CA, USA, 2017.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4, Shanghai, China, 2011.

[RF17] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, Honolulu, HI, USA, 2017.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, pages 234–241, Munich, Germany, 2015. Springer.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323:533–536, 1986.

[RKC19] Thomas Roddick, Alex Kendall, and Roberto Cipolla. Orthographic Feature Transform for Monocular 3D Object Detection. In *30th British Machine Vision Conference (BMVC)*, Cardiff, UK, 2019.

[Sam59] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[SBP+19] Andrea Simonelli, Samuel Rota Bulo, Lorenzo Porzi, Manuel López-Antequera, and Peter Kontschieder. Disentangling Monocular 3D Object Detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1991–1999, Seoul, South Korea, 2019.

[Sch18] Matthias Schreier. Environment Representations for Automated On-road Vehicles. *at - Automatisierungstechnik*, 66:107–118, 2018.

[SFH+21] Lukas Stäcker, Juncong Fei, Philipp Heidenreich, Frank Bonarens, Jason Rambach, Didier Stricker, and Christoph Stiller. Deployment of Deep Neural Networks for Object Detection on Edge AI Devices with Runtime Optimization. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 1015–1022, Montreal, BC, Canada, 2021.

[SKD+20] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott M. Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2443–2451, Seattle, WA, USA, 2020.

[SM86] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., USA, 1986.

[SMAG18] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-YOLO: An Euler-Region-Proposal for Real-time 3D Object Detection on Point Clouds. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, volume 11129, pages 197–209, Munich, Germany, 2018. Springer.

[ST19] Leslie N. Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks using Large Learning Rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, pages 369 – 386, Baltimore, MA, USA, 2019. SPIE.

[SWL19] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–779, Long Beach, CA, USA, 2019.

[SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.

[SZT19] Vishwanath A. Sindagi, Yin Zhou, and Oncel Tuzel. MVX-Net: Multimodal VoxelNet for 3D Object Detection. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7276–7282, Montreal, QC, Canada, 2019.

[Tea] OpenPCDet Development Team. OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds. GitHub repository: `https://github.com/open-mmlab/OpenPCDet`. (accessed on 05/20/2022).

[TPRZ20] Larissa T. Triess, David Peter, Christoph B. Rist, and Johann Marius Zöllner. Scan-based Semantic Segmentation of LiDAR Point Clouds: An Experimental Study. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1116–1121, Las Vegas, NV, USA, 2020.

[Van16] Han Vanholder. Efficient Inference with TensorRT. In *GPU Technology Conference*, San Jose, CA, USA, 2016. URL: `https://on-demand.gputechconf.com/gtc-eu/2017/presentation/23425-han-vanholder-efficient-inference-with-tensorrt.pdf`. (visited on 10/18/2023).

[VLHB20] Sourabh Vora, Alex H Lang, Bassam Helou, and Oscar Beijbom. PointPainting: Sequential Fusion for 3D Object Detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4603–4611, Seattle, WA, USA, 2020.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008, Long Beach, CA, USA, 2017.

[WCG+19] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-LiDAR from

Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8445–8453, Long Beach, CA, USA, 2019.

[WK19] Xinshuo Weng and Kris Kitani. Monocular 3D Object Detection with Pseudo-LiDAR Point Cloud. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 857–866, Seoul, South Korea, 2019.

[WP15] Dominic Zeng Wang and Ingmar Posner. Voting for Voting in Online Point Cloud Object Detection. In *Robotics: Science and Systems*, Rome, Italy, 2015.

[WSK+15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, Boston, MA, USA, 2015.

[WSVDH19] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. Wider or deeper: Revisiting the ResNet Model for Visual Recognition. *Pattern Recognition*, 90:119–133, 2019.

[WWYK18] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893, Brisbane, Australia, 2018.

[WXC+21] Mark Weber, Jun Xie, Maxwell Collins, Yukun Zhu, Paul Voigtlaender, Hartwig Adam, Bradley Green, Andreas Geiger, Bastian Leibe, Daniel Cremers, Aljosa Osep, Laura Leal-Taixe, and Liang-Chieh Chen. STEP: Segmenting and Tracking Every Pixel. In *Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, virtual, 2021.

[WZPL21] Tai Wang, Xinge Zhu, Jiangmiao Pang, and Dahua Lin. FCOS3D: Fully Convolutional One-Stage Monocular 3D Object Detection. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 913–922, Montreal, BC, Canada, 2021.

[WZZ+19] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4376–4382, Montreal, QC, Canada, 2019.

[XC18] Bin Xu and Zhenzhong Chen. Multi-level Fusion Based 3D Object Detection from Monocular Images. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2345–2353, Salt Lake City, UT, USA, 2018.

[XWW+20] Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Péter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 12373, pages 1–19, Glasgow, UK, 2020. Springer.

[Yan20] Yulin Yang. 3D Pedestrian Detection using LiDAR Point Sets and Image Semantic Segmentation. Master's thesis, Institute of Measurement and Control Systems (MRT), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 2020.

[YLU18] Bin Yang, Wenjie Luo, and Raquel Urtasun. PIXOR: Real-time 3D Object Detection from Point Clouds. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7652–7660, Salt Lake City, UT, USA, 2018.

[YML18] Yan Yan, Yuxing Mao, and Bo Li. SECOND: Sparsely Embedded Convolutional Detection. *Sensors*, 18(10):3337, 2018.

[YSL+18] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. IPOD: Intensive Point-based Object Detector for Point Cloud. arXiv: `https://arxiv.org/abs/1812.05276`, 2018. (visited on 01/12/2023).

[YSLJ20] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3DSSD: Point-Based 3D Single Stage Object Detector. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11037–11045, Seattle, WA, USA, 2020.

[YWC+20] Yurong You, Yan Wang, Wei-Lun Chao, Divyansh Garg, Geoff Pleiss, Bharath Hariharan, Mark Campbell, and Kilian Q Wein-

berger. Pseudo-LiDAR++: Accurate Depth for 3D Object Detection in Autonomous Driving. In *8th International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, 2020. OpenReview.net.

[YZK21] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3D Object Detection and Tracking. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11779–11788, virtual, 2021.

[ZJZ+19] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection. arXiv: `https://arxiv.org/abs/1908.09492`, 2019. (visited on 05/19/2022).

[ZSL+21] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable Transformers for End-to-End Object Detection. In *9th International Conference on Learning Representations (ICLR)*, virtual, 2021. OpenReview.net.

[ZSR+19] Yi Zhu, Karan Sapra, Fitsum A. Reda, Kevin J. Shih, Shawn Newsam, Andrew Tao, and Bryan Catanzaro. Improving semantic segmentation via video propagation and label relaxation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8848–8857, Long Beach, CA, USA, 2019.

[ZT18] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4490–4499, Salt Lake City, UT, USA, 2018.

[ZWK19] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as Points. arXiv: `https://arxiv.org/abs/1904.07850`, 2019. (visited on 01/15/2022).

[ZZW+21] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Wei Li, Yuexin Ma, Hongsheng Li, Ruigang Yang, and Da Lin. Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR-Based Perception. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:6807–6822, 2021.