FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Multi-modal Object Detection for Autonomous Driving

Ana Margarida Braga de Oliveira

Mestrado em Visão por Computador

Supervisor @ FEUP: Luís Filipe Teixeira

Supervisor @ Bosch: Joaquim Fonseca

Co-Supervisor @ Bosch: Pedro Girão

March 31, 2022

© Ana Margarida Braga de Oliveira, 2022

Resumo

Com aos avanços na área da Inteligência Artificial, mais concretamente em *Deep Learning*, a Condução Autónoma tem-se tornardo um crescente foco na comunidade científica. Os veículos autónomos necessitam de entender o ambiente que os rodeia, e portanto, é imperativo existirem sistemas inteligentes desenvolvidos para detetar os diversos elementos integrantes da cena. Estes sistemas têm de ser capazes de detetar e identificar os objetos relevantes na cena, que tipicamente consistem nas classes carros, motociclos, ciclistas e peões. Neste sentido, duas modalidades de sensores são utilizadas: LiDAR e imagem RGB.

Esta dissertação é desenvolvida no âmbito do projeto de Inovação THEIA, que consiste na investigação e desenvolvimento de diversos módulos que incorporam um sistema de perceção para Condução Autónoma. Mais especificamente, este trabalho aborda a necessidade da existência de um sistema de perceção para detetar e reconhecer os objectos em potencial movimento integrantes da cena 3D. Assim, foi realizado um estudo comparativo das vantagens e desvantagens dos métodos mais recentes e com melhor resultados na literatura. Além disso, uma solução baseada em dados multi-modais foi proposta e desenvolvida para detetar e classificar os objetos mais relevantes. A arquitetura proposta consiste no modelo de fusão MVX-Net com o PointPillars. Adicionalmente, uma outra abordagem foi desenvolvida, implementando a rede SECOND com a MVX-Net. Finalmente, foram realizadas experiências utilizando dados mais desafiantes, através do treino e validação dos modelos com o *dataset* nuScenes.

A solução foi avaliada relativamente à sua *performance* e velocidade de inferência. Várias experiências foram realizadas para aumentar os objetos com menor representação, aplicando uma estratégia de amostragem proposta em [1]. O modelo obteve resultados notáveis, superando os resultados obtidos com a *baseline* e com a 3DSSD para a classe Peões. Além disso, obteve também melhores resultados relativamente à velocidade de inferência, executando a 15.2 Hz nos testes realizados. De forma geral, a arquitetura proposta provou ser uma abordagem adequada para aplicações em tempo-real, alcançando um equilíbrio entre *performance* e velocidade de inferência.

Palavras-chave: Condução Autónoma; *Deep Learning*; Deteção de Objetos.

Abstract

With the advances in Artificial Intelligence, more specifically in Deep Learning, Autonomous Driving has become an increased focus in the scientific community. Self-driving vehicles need to accurately understand its surrounding environment and therefore, it is imperative to have intelligent systems designed to detect the diverse elements integrating the scene. In particular, these vehicles have to be able to detect and identify the relevant objects in the scene, which usually consist in cars, motocycles, cyclists and pedestrians. For that, two sensor modalities can be used: LiDAR and RGB image.

This dissertation is developed under the THEIA Innovation project, which consists in the research of several modules to incorporate a perception system for Autonomous Driving. More specifically, this work addresses the necessity of having a perception system to detect and recognize the Potentially Moving Objects (PMO) integrating the 3D scene. Therefore, a study was conducted comparing the advantages and disadvantages of the more recent top-performing methods in the literature. Moreover, a solution based on multi-modal data was proposed and developed to detect and classify the more relevant objects in the scene. The proposed architecture consists in the MVX-Net fusion model with PointPillars detector. The model was trained and tested using the benchmark KITTI dataset and an in-depth analysis and comparison with the top-performing single-modality detector 3DSSD was conducted. Additionally, another approach was developed, which implements the SECOND network on the top of MVX-Net. At the end, experiments were conducted using more challenging data, by training and evaluating the models with nuScenes dataset.

The solution was assessed regarding its performance and inference speed. Experiments were performed to increase the underrepresented objects in the scene, by applying a sample strategy proposed in [1]. The model achieved interesting results, outperforming both the baseline implementation and 3DSSD for pedestrians class. Moreover, it achieved the best results regarding inference speed, running at 15.2 Hz in the tests performed. Overall, the proposed architecture proved to be a suitable approach for real-world application, achieving a trade-off balance between performance and inference speed.

Keywords: Autonomous Driving; Deep Learning; Object Detection.

Acknowledgements

I would like to thank my supervisor Prof. Luís Filipe Teixeira from the Faculty of Engineering of University of Porto for providing guidance and assistance during this dissertation. To Eng. Joaquim Fonseca and Eng. Pedro Girão from Bosch Car Multimedia for their continuous support and availability, orienting me to the right direction.

I would also like to thank Bosch Car Multimedia for providing me a good working environment, and to my colleagues of the "THEIA Perception Innovation" team for their availability and help during this process.

Finally, I would like to express my deep gratitude to my parents for being always present, for their encouragement and motivation, and for their unconditional support, always acknowledging my achievements.

Margarida Oliveira

This work is supported by European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project nº 047264; Funding Reference: POCI-01-0247-FEDER-047264].

"Os meus heróis na vida real
são os que desafiam a lei
em nome de um ideal."

Natália Correia

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 3
- 1.3 Objectives . . . 3
- 1.4 Contributions . . . 4
- 1.5 Document Structure . . . 4

2 Background **5**
- 2.1 Object Detection . . . 6
  - 2.1.1 Traditional Approaches . . . 6
- 2.2 Machine Learning . . . 7
  - 2.2.1 Artificial Neural Networks . . . 8
  - 2.2.2 The Learning System . . . 11
  - 2.2.3 Underfitting and Overfitting . . . 17
- 2.3 Object Detection Architectures . . . 18

3 Point Cloud Processing for 3D Object Detection **21**
- 3.1 Point Cloud . . . 22
  - 3.1.1 Properties . . . 22
  - 3.1.2 Representation . . . 23
- 3.2 Benchmark Datasets . . . 24
  - 3.2.1 KITTI Vision Benchmark Suite . . . 24
  - 3.2.2 Waymo Open Dataset . . . 25
  - 3.2.3 nuScenes . . . 26
- 3.3 3D Object Detection Task . . . 26
  - 3.3.1 Regression Targets with Anchor Boxes . . . 26
  - 3.3.2 Data Augmentation . . . 27
- 3.4 Evaluation Metrics . . . 28
  - 3.4.1 3D Intersection over Union (IoU) . . . 28
  - 3.4.2 3D Average Precision . . . 29
  - 3.4.3 Average Orientation Similarity (AOS) . . . 29
  - 3.4.4 NuScenes Detection Score (NDS) . . . 29
- 3.5 State-of-the-Art Approaches . . . 30
  - 3.5.1 Point-based Methods . . . 30
  - 3.5.2 Grid-based Methods . . . 32
  - 3.5.3 Graph-based Methods . . . 33
- 3.6 Sensor Fusion . . . 34
  - 3.6.1 Early Fusion . . . 35

3.6.2 Late Fusion . . . 36
3.6.3 Deep Fusion . . . 37
3.7 Final Remarks . . . 37

4 Methodology **39**
4.1 Framework Implementation . . . 41
4.1.1 MVX-Net Architecture . . . 41
4.2 Data Processing . . . 42
4.2.1 KITTI 3D Object Detection . . . 43
4.2.2 nuScenes . . . 43
4.3 Experimental Setup . . . 44
4.3.1 Hardware . . . 45
4.3.2 Model Generalization . . . 45

5 Results and Analysis **47**
5.1 Inference Rate . . . 50
5.2 Model Generalization . . . 52
5.3 Results Analysis . . . 53
5.4 Final Remarks . . . 58

6 Conclusions and Future Work **61**
6.1 Future Work . . . 62

A Training Results **63**

B Visual Results **65**

References **67**

List of Figures

1.1 The six levels of automation defined in SAE J3016 standard (SAE International 2016), extracted from [4] . . . 2

2.1 Example of the object detection task, extracted from [9]. . . . 6
2.2 Result of applying Non-Maximum Suppression method, extracted from [9]. . . . 7
2.3 Neuron representation through a mathematical model, proposed in [20] . . . 9
2.4 ANN architecture for a supervised learning multi-layer perceptron approach, extracted from [21]. . . . 9
2.5 The optimization algorithms may fail to find a global minimum when there are multiple local minima. Therefore, in deep learning, a local minimum that correspond to significantly low values of the cost function is acceptable solution. Figure extracted from [18]. . . . 10
2.6 Intersection over Union (IoU) formulation, represented through a visual aid. The blue box refers to the predicted bounding box and the orange box refers to the ground truth bounding box. . . . 12
2.7 Example of a confusion matrix computed from a binary classifier, extracted from [26]. The classes Positive (P) and Negative (N) are defined and the predicted output is True (T) or False (F). . . . 15
2.8 Example of the confusion matrix computed from a multi-class classifier, extracted from [26]. . . . 16
2.9 Example of the fit of three models in the training set. (Left) A linear function fit to the data experiences underfitting, once the curvature present in the data can not be captured. (Center) A quadratic function fit to the data shows to generalize well the unobserved points. (Right) A polynomial function fit to the data experiences overfitting, since the model is too specific for these data points and it will not be suitable for unobserved samples. . . . 17
2.10 Overview of the Faster R-CNN, composed of a RPN module which serves as the 'attention' of the network. . . . 19
2.11 Overview of the YOLO architecture, extracted from [32]. This approach first divides the image into an S x S grid, predicting B bounding boxes for each grid cell, confidence for those boxes, and C class probabilities. . . . 20
2.12 Comparison between the SSD and the YOLO approaches, extracted from [33]. The SSD approach adds a set of feature layers to the end of the base network, which predict the offsets to default boxes of different scales and aspect ratios, as well as, their confidence scores. The SSD detector significantly outperforms the YOLO approach in both accuracy and speed, when tested with the PASCAL VOC2007 dataset. . . . 20

3.1 Point Cloud captured by a LiDAR scan, adapted from [36]. 23
3.2 Overview of the most used approaches to discretize the 3D space, extracted from [35]. (a) The 3D voxelization representation consists in discretizing the 3D space into equally-spaced non overlapping voxels from each of the three dimensions. (b) The range-view representation discretizes the 3D space along the azimuth and elevation angles. (c) The Bird's Eye-view representation consists in discretizing the 3D space along the x and y axes, ignoring the height dimension. 24
3.3 Recording platform with sensors, extracted from [37]. A stereo camera, a 360º Velodyne scan and a GPS module were the setup used to capture the dataset. . . . 25
3.4 Sensor setup in the data collection vehicle, extracted from [39]. 26
3.5 Illustration of the data augmentation transformations for LiDAR point clouds, extracted from [41]. In the top images, examples of ground truth sampling, random flips, world scaling and global translation noise are presented. In the bottom images, examples of frustum dropout and noise, random rotation and random drop of laser points are shown. 28
3.6 Representation of the 3D IoU, extracted from [42]. The pink and the green boxes represent the predicted bounding box B_p and the ground truth B_g, respectively, and the yellow defines the 3D intersection between them I_{3D}. 29
3.7 Overview of a lightweight PointNet architecture, extracted from [46]. n refers to the number of input points, M refers to the dimension of the learned features for each point. 31
3.8 Overview of the 3DSSD architecture, extracted from [48]. It comprehends a backbone and a box prediction network, which includes a candidate generation layer and an anchor-free prediction head. (a) Backbone network, which generates global features for all representative points through several Set Abstraction (SA) layers with fusion sampling (FS) strategy, taking as input the raw point cloud. (b) Candidate generation layer (CG), which generates the features for representative points. (c) Anchor-free prediction head. 32
3.9 Overview of the PointPillars approach, extracted from [51]. The point cloud is transformed into a stacked pillar tensor and pillar index tensor. The stacked pillars are used by the encoder to learn a set of features that can be projected back into a 2D pseudo-image for a CNN. Finally, the detection head predicts 3D bounding boxes for objects. 33
3.10 Architecture of the Point-GNN, extracted from [45]. (a) Graph generation from the point cloud; (b) Graph neural network for object detection; (c) Bounding box merging and scoring. 34
3.11 Overview of the different types of data fusion methods, adapted from [53]. It includes early fusion, late fusion and deep fusion. 35
3.12 Overview of the MVX-Net PointFusion architecture, extracted from [57]. 36

4.1 Overview of the MVX-Net architecture with the most relevant 2D and 3D object detectors available to integrate this approach. The networks used in the original MVX-Net proposal are highlighted, which consist in Faster R-CNN and VoxelNet. 40
4.2 Overview of the proposed implementation of MVX-Net architecture with VoxelNet, PointFusion and SECOND detectors. The Faster R-CNN and the Voxel-Net are the detectors proposed in the paper [54], which are further referred as the baseline architecture. Additionally, the PointPillars and the SECOND networks are implemented in this framework. 41
4.3 Overview of the MVX-Net PointFusion architecture, extracted from [54]. 42

4.4 Class distribution of the KITTI dataset, extracted from [37]. 43

5.1 Loss obtained when training the 3DSSD during 120 epochs. 50
5.2 Loss obtained when training the MXV-Net + PP and MVX-Net + S during 120 epochs. MVX-Net + PP (SS2) refers to the fusion architecture implemented with PointPillars applying a sample strategy and MVX-Net + S refers to the fusion architecture implemented with SECOND. 50
5.3 Trade-off between performance and inference rate. The horizontal axis presents the inference rate (in Hz) and the vertical axis presents the performance (in %) in moderate difficulty of pedestrians class, using the metric AP_{BEV}. MVX-Net Baseline refers to MVX-Net with VoxelNet; MVX-Net + PP (SS2) refers to MVX-Net with PointPillars applying a sampling strategy; MVX-Net + S refers to MVX-Net with SECOND. 52
5.4 Confusion matrix obtained with MVX-Net with PointPillars (SS2). The columns refer to the true classes and the rows to the predicted classes. The green cells are the TP values. 53
5.5 Example of sample image 2206. The two pedestrians within the selected area are misclassified as cyclists. 54
5.6 Example of sample image 4954, in which two pedestrians are misclassified as cyclists. Presented within the selected area, these objects are zoomed in for better visualization. 55
5.7 Example of sample image 4928, which includes a set of labeled pedestrians. The pedestrians within the selected area are misclassified as cyclists, zoomed in for better visualization. 55
5.8 Example of sample image 63, which includes a set of labeled pedestrians. The pedestrian within the selected area is misclassified as car, zoomed in for better visualization. 56
5.9 Confusion matrix obtained with MVX-Net with PointPillars. The columns refer to the true classes and the rows to the predicted classes. The green cells are the TP values. 56
5.10 Example of sample image 2091, which includes one labeled pedestrian in the scene. The pedestrians within the selected area is misclassified as car, zoomed in for better visualization. 57
5.11 Example of sample image 7080. In this scene, there is one labeled pedestrian, shown within the selected area. This pedestrian is misclassified as cyclist. 58
5.12 Example of sample image 6785. In this scene, there is one labeled pedestrian, shown within the selected area. This pedestrian is misclassified as car. 58

A.1 Loss obtained when training the MXV-Net + PP during 120 epochs. MVX-Net + PP refers to the fusion architecture implemented with PointPillars without any sampling strategy. 63
A.2 Loss obtained when training the MXV-Net Baseline during 120 epochs. 63

B.1 Example of sample image 5812. The labeled pedestrian in the scene, shown within the selected area, is correctly classified by the model MVX-Net PointPillars (SS2). 65
B.2 Example of sample image 76. The labeled cyclist in the scene is correctly classified by the model MVX-Net PointPillars (SS2). 65
B.3 Example of sample image 811. The labeled cyclist in the scene is correctly classified by the model MVX-Net PointPillars (SS2). 66

B.4 Example of sample image 1008. There is a labeled car in the scene, which is correctly classified by the model MVX-Net PointPillars (SS2). 66

List of Tables

3.1 Definition of the object difficulty in the KITTI dataset. It is composed of Easy, Moderate and Hard, which are defined regarding the minimum bounding box height, the occlusion level and the maximum truncation. 25

3.2 Comparison of the results obtained for 3D object detection on the KITTI test 3D detection benchmark, reported by [44]. The 3D bounding box IoU threshold is 0.7 for cars and 0.5 for pedestrians and cyclists. In the modality column, L refers to the LiDAR and I refers to the Image. Also, the letters 'E', 'M' and 'H' state for easy, moderate and hard difficulties, respectively. The results are presented in percentage (%). 38

4.1 Results of MVX-Net on the KITTI validation set, using mean Average Precision (in %) with IoU=0.7, adapted from [54]. MVX-Net (PF) stands for PointFusion strategy. 42

4.2 Split of KITTI 3D Object Detection dataset. 43

4.3 Organization of the nuScenes dataset. 44

4.4 Hyperparameters setup used for train each of the architectures. PP refers to PointPillars network; MVX-Net Baseline refers to the original architecture, which uses VoxelNet as the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars; MVX-Net + S refers to the MVX-Net with SECOND. 45

5.1 Comparison of the results obtained on the KITTI validation set, applying the sampling strategy proposed in [1]. The values assigned to each configuration parameter are detailed, which are the same for all the difficulties in each class. The AP_{BEV} with IoU threshold of 0.7 for cars and 0.5 for pedestrians and cyclists was used for evaluation, presented in percentage (%). 48

5.2 Comparison of the results obtained on the KITTI validation set. The AP_{BEV} IoU threshold is 0.7 for cars and 0.5 for pedestrians and cyclists. The results are presented in percentage (%). MVX-Net baseline refers to the original architecture, which uses VoxelNet as the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars as the 3D detector; MVX-Net + S refers to the MVX-Net with SECOND network. The SS2 and SS4 refers to the Sampling Strategy applied to the data. 49

5.3 Comparison of the inference rate results obtained with each architecture. MVX-Net baseline refers to the original architecture, which uses VoxelNet the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars as the 3D detector; MVX-Net + S refers to the MVX-Net with SECOND network. 51

5.4 Comparison of the results obtained with the nuScenes validation set. The metrics used to evaluate consist in the NDS (NuScenes Detection Score), mAP and five error metrics regarding translation, scale, orientation, velocity and attribute, corresponding to mATE, mASE, mAOE, mAVE and mAAE, respectively. MVX-Net + PP refers to the MVX-Net with PointPillars. 53

A.1 Comparison of the results obtained on the KITTI validation set. The AP_{3D} IoU threshold is 0.7 for cars and 0.5 for pedestrians and cyclists. The results are presented in percentage (%). MVX-Net baseline refers to the original architecture, which uses VoxelNet the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars as the 3D detector; MVX-Net + S refers to the MVX-Net with SECOND network. The SS2 refers to the Sampling Strategy applied to the data. 64

A.2 Comparison of the results obtained on the KITTI validation set. The AP_{AOS} IoU threshold is 0.7 for cars and 0.5 for pedestrians and cyclists. The results are presented in percentage (%). MVX-Net baseline refers to the original architecture, which uses VoxelNet the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars as the 3D detector; MVX-Net + S refers to the MVX-Net with SECOND network. The SS2 refers to the Sampling Strategy applied to the data. 64

Abbreviations and Symbols

2D	Two Dimensions
3D	Three Dimensions
AD	Autonomous Driving
ADAS	Advanced Driver Assistance System
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
AVOD	Aggregate View Object Detection
BEV	Bird's eye view
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DPM	Deformable Part-based Model
FN	False Negative
FP	False Positive
FPS	Farthest Point Sampling
FS	Fusion Sampling
HOG	Histogram of Oriented Gradients
IoU	Intersection over Union
LiDAR	Light Detection and Ranging
mAP	Mean Average Precision
ML	Machine Learning
MLP	Multi-Layer Perceptron
mTP	mean True Positive
MV3D	Multi-View 3D Object Detection
NDS	NuScenes Detection Score
NMS	Non-Maximum Suppression
RPN	Region Proposal Network
SA	Set Abstraction
SAV	Shared Autonomous Vehicles
SIFT	Scale-Invariant Feature Transform
SSD	Single Shot MultiBox Detector
TP	True Positive
TN	True Negative
VFE	Voxel Feature Encoding
YOLO	You Only Look Once

Chapter 1

Introduction

The research study reported by the National Highway Traffic Safety Administration (NHTSA), shows that 94% of road accidents are caused by human errors [2]. In this sense, Autonomous Driving systems intend to address the safety of the passengers in the daily transportation, by developing efficient and accurate systems that can perform the driving activity and thus, prevent from human failures. During the last years, with the improvements in Artificial Intelligence and the rise of Deep Learning technologies, a revolution in the Autonomous Driving field was noticed. Recent progress has been made, however, the development of effective Autonomous Driving systems is still a very challenging task.

1.1 Context

The driving automation systems are rated in six levels of automation, ranging from 0 to 5, which corresponds respectively, from vehicles with no degree of automation to a fully automated vehicle. This is a standard adopted by SAE (SAE International 2016) and an illustration is provided in figure 1.1. Level 0 only depends on the driver to operate the vehicle. The majority of the current vehicles on the road are on this level; systems manually controlled by a human, that may be incorporated with some aid system, such as the emergency braking system, which is not considered an automation as it does not drive the vehicle. The lowest level of automation is level 1, where the vehicle provides driver assistance, such as the Adaptive Cruise Control. This feature consists in keeping the vehicle at a safe distance behind the next car. The following level is the Partial Driving Automation, level 2, commonly known as driver assistance systems or ADAS. In this case, the vehicle is able to control the steering and acceleration, however, at any time the driver can take control of the driving activity. From level 2 to level 3, Conditional Automation, there are major advances regarding the technology perspective. These vehicles are able to take informed self-decisions, by perceiving the surrounding environment, such as accelerating past a slow-moving vehicle. Level 4 vehicles, High Automation, have an active role in the case of a system failure. These cars can operate in self-driving mode, although due to legislation issues these prototypes can only ride within a limited area, usually with high speed limitations and across urban areas,

which is known as geofencing. Most of these transportation concepts are designed for ridesharing, which is a paradigm where the mobility service is performed by shared transportation, instead of private vehicles. A particularly relevant mobility service is the Shared Autonomous Vehicles (SAV), which consists in a shared vehicle, such as a taxi, with the capacity to drive autonomously by itself [3]. Finally, level 5 vehicles, Fully Automated, do not require any human intervention. The components that integrate the driving activity are eliminated, such as the steering wheel and the acceleration/braking pedals. There are some prototypes currently under development that rely on this level of automation, however, they are not publicly available yet.

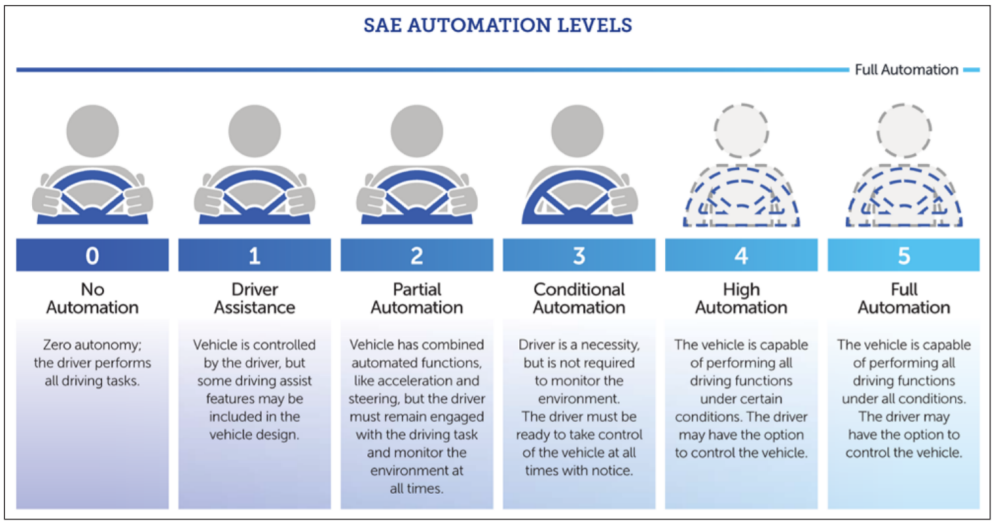


Figure 1.1: The six levels of automation defined in SAE J3016 standard (SAE International 2016), extracted from [4]

The implementation of self-driving cars faces a lot of bureaucratic constrains. These systems are designed to interact with all the integrating parts of the urban scene and therefore, they need to address straight requirements regarding the human safety. In a world conceived for humans, the employment of such intelligent systems is a huge challenge. Divergent positions are taken regarding the employment of these applications, since in the general community there is some reluctance in adopting intelligent systems that do not need human supervision. This is justified by the lack of trust on self-decidery applications that interfere with the human activities. In this regard, it is mandatory that autonomous vehicles are designed to attend the human security and safety as a top priority. Mainly when these systems interact with pedestrians that are present in the scene, the caution is even more important and for that, their detection and identification is essential.

For this reason, it is imperative to have intelligent systems designed to detect the diverse elements integrating the surrounding environment of self-driving cars. Thus, object detection applications are developed to detect and recognize the most relevant parts of the scene, which usually consist in the vehicles, that can be cars, motorcyclists and cyclists, and the pedestrians.

This last category is often the focus when developing Autonomous Driving systems, as they are the most vulnerable element integrating the scene.

1.2 Motivation

In order to understand the surrounding environment and use this knowledge to make driving decisions, autonomous vehicles must analyze data streams generated by their integrated sensors. Typically, the environment understanding involves solving various perception tasks, in which the ability to detect objects, consisting in localization and classification, is a core part for autonomous vehicles to be able to perceive its surrounding, such as the pedestrians, cars, trucks, cyclists and trees, among others. In this sense, it is necessary to have accurate and precise systems to implement object detection, in order to extract structured information from the raw data, specifically position, size, orientation, and class of the objects detected in the scene. The literature has been well documented regarding image-based 2D object detection, however, the use of multi-modal data, consisting in the conjunction of 2D image with 3D Point Clouds is still comparatively new and not as well studied [5]. Therefore, this work aims to explore and assess the feasibility of using LiDAR in conjunction with image-based inputs for Autonomous Driving 3D object detection approaches.

This dissertation is developed under the THEIA Innovation project, which comprehends the research and development of a perception system to address the different tasks involved in the Autonomous Driving activity. This project aims to investigate emerging Deep Learning methods based on datasets provided. For this purpose, a system is built in a test vehicle to collect the target data. Motivated by this proposal, this work centers in the specific task of object detection, focusing on Potentially Moving Objects, which comprehend cars, cyclists and pedestrians. Moreover, one of the goals of this project is to benchmark the best models on open source datasets, which consider the same sensor modalities installed in the test car used in the THEIA project for object detection research purposes, consisting on RGB cameras and LiDAR sensors. Therefore, this work relies on the use of RGB images and LiDAR point clouds for the detection task.

1.3 Objectives

The main goal of this dissertation is to study the 3D object detection in the context of Autonomous Driving. In this sense, two main steps are highlighted. In the first stage, this work aims to establish a comparison between the single-modality and multi-modality state-of-the-art architectures for 3D object detection. The 2D image based object detection is first described, providing some state-of-the-art architectures. The point clouds in the context of object detection are presented, in order to understand its structure and representation. Additionally, the relevant datasets and their evaluation metrics are covered. Finally, the state-of-the-art architectures based on 3D object detection are studied, conducting an in-depth comparison between the LiDAR-based and fusion-based methodologies. In the second stage, based on the study previously conducted,

this dissertation aims to develop a fusion approach, by selecting a state-of-the-art 3D detection architecture.

1.4 Contributions

The main contributions regarding this dissertation are to conduct a benchmark study regarding 3D object detection architectures, as well as propose and develop an end-to-end trainable fusion approach for object detection in the context of Autonomous Driving. For that, a deep learning 3D object detection approach is proposed and developed. Based on multi-modal data, this work implements a fusion architecture to combine LiDAR and RGB image inputs. To achieve this goal, a publicly available dataset widely studied in the state-of-the-art methods was employed to train and test this approach. Additionally, another public dataset was also used to train and test the proposed methodology, which enables the system validation in a large dataset that incorporates more realistic scenarios, as well as, the models ability to generalize given new information.

Moreover, this work is focused in the benchmark of state-of-the-art architectures for 3D object detection and, therefore, finding the best possible hyperparameters for the proposed and implemented approaches is not the main purpose under this dissertation. In this sense, a comparison between the state-of-the-art methodologies is conducted. An evaluation and in-depth analysis of the proposed approach is developed, as well as, its feasibility to be implemented in the context of Autonomous Driving, regarding the model performance and inference speed, considering the required resources to run the algorithm.

1.5 Document Structure

This document is organized in five additional chapters, aside from the Introduction. Chapter 2 presents a contextualization of the 2D object detection task, as well as, relevant state-of-the-art architectures. Moreover, important concepts regarding Deep Learning are described. Chapter 3 provides a review of the 3D object detection task. The point clouds in the context of object detection are described, the benchmark datasets and the relevant metrics used are presented, and a study of the state-of-the-art architectures is conducted. Chapter 4 describes the methodology developed and the implementation details. Chapter 5 presents the results, conducting a comparison between the state-of-the-art architectures. An in-depth analysis of the developed methodologies is conducted, taking into consideration the requirements of Autonomous Driving applications. Finally, chapter 6 draws the conclusions and raises topics for future work.

Chapter 2

Background

This chapter provides a contextualization of the object detection task. Some of the most relevant traditional based approaches are introduced. Moreover, a description of the main concepts in the context of Machine Learning is provided. Finally, the state-of the-art architectures for 2D Object Detection are presented. An overview of the sections described is provided in the following list:

- **Object Detection:** Context of the Object Detection problematic, section 2.1.
- **Machine Learning:** Relevant concepts related with Machine Learning, section 2.2.
- **Object Detection Architectures:** Relevant 2D object detection architectures, section 2.3.

2.1 Object Detection

Object detection is the task of detecting instances of visual objects of a certain class in images or videos [6]. It estimates the location and the class of the objects in the image, which consists in a regression problem, to estimate the position and size of the corresponding object's bounding box and a classification task, which consist in categorizing the object into a certain class. Object detection is the basis for other computer vision tasks, such as instance segmentation, image captioning and object tracking [7]. Figure 2.1 shows an example of the object detection task.

Computer Vision approaches based on object detection fall into two main categories: Traditional based methods, mostly developed until 2015 and Deep Learning based algorithms, which were the main focus after 2015 [8].

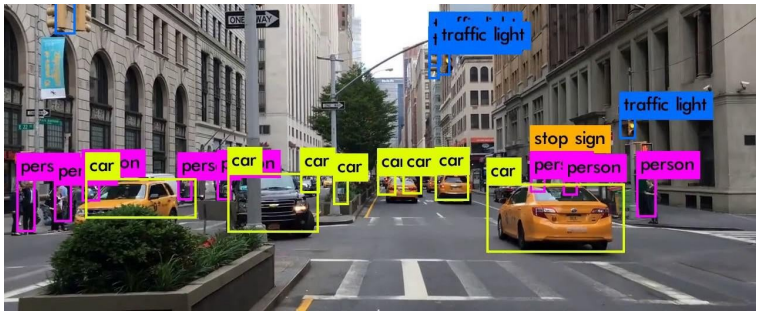


Figure 2.1: Example of the object detection task, extracted from [9].

2.1.1 Traditional Approaches

Traditional approaches for image and video processing were the focus in Computer Vision based applications until the arise of Machine Learning and Deep Learning algorithms. Part of these Computer Vision based algorithms appeared in the 1970s, with the emergence of Computer Vision as a separated field of research [10]. Hand-craft based feature methods consist in algorithms developed to extract relevant properties from a visual representation of the input data. These methods usually include three main steps: informative region selection, feature extraction, and classification [11].

2.1.1.1 Region Selection

In Traditional approaches, the region selection consists in applying a multi-scale **sliding window** to scan the whole image, in order to find all the existent objects. An image can be composed of different objects at any position with different aspect ratio or size and thus, it is a suitable approach to search the whole image. However, the large amount of candidate windows makes this technique computationally expensive and also leads to numerous redundant windows.

2.1.1.2 Feature Extraction

In order to recognize different objects, it is necessary to extract visual features that can provide a semantic and robust representation. Several descriptors can be computed to extract relevant features, although, as different objects have different appearances, illumination conditions and backgrounds, it is a challenging task to manually design a robust feature descriptor that accurately describe all types of objects. The most relevant algorithms for feature extraction are: Histogram of Oriented Gradients [12], Scale-Invariant Feature Transform (SIFT) [13] and Deformable Part-based Model (DPM) [14], among others.

2.1.1.3 Non-Maximum Suppression

After features extraction, multiple candidate bounding boxes are assigned for each object. In order to select one single box corresponding to each object, the Non-Maximum Suppression (NMS) technique is commonly used. It is a Computer Vision method that selects a single bounding box from many overlapping entities, using a criteria based on the highest probability score. This method compares the classification probability of each entity and discards the lowest one, interactively until only one bounding box remains for the object. Figure 2.2 shows an example of applying Non-Maximum Suppression.

Figure 2.2: Result of applying Non-Maximum Suppression method, extracted from [9].

2.2 Machine Learning

Machine learning is a branch of Artificial Intelligence which studies computer algorithms that automatically improve through experience and data usage [15]. It consists in developing mathematical models based on sample data, known as training data. The goal is to have a model capable of making predictions or decisions, without being explicitly programmed to accomplish the task [16]. Machine Learning techniques have been implemented for diverse fields and applications, which include pattern recognition, computer vision, spacecraft engineering, finance, entertainment, and biomedical applications. [17].

Machine Learning algorithms can be categorized according to the data used for the training stage:

- **Supervised learning:** A dataset with labelled data is used, as such, each sample is associated with a label or target [18]. In supervised learning, the algorithms are instructed to learn a general rule that maps the inputs to the outputs.

- **Unsupervised learning:** The training set is composed of unlabeled data. The samples contain many features and thus, the models learn to classify each case based on its similarity with other cases properties.

- **Semi-supervised learning:** The training set is composed of a small amount of labeled data together with a large amount of unlabeled data. Semi-supervised learning is a concept between unsupervised learning, which is composed of unlabeled training data, and supervised learning, which is composed of completely labeled training data.

- **Reinforcement learning:** The training data is only provided as feedback to an intelligent agent in a dynamic environment. The task performance is improved through this feedback between the learning system and the interaction experience.

- **Transfer learning:** A transfer of knowledge is applied from a source domain to a target domain.

Deep Learning

Deep Learning is an extent of Machine Learning, which relies on **Artificial Neural Networks** with multiple layers. Deep refers to the depth of the Artificial Neural Networks, that have a large amount of layers. In Deep Learning, as in traditional Machine Learning algorithms, handcrafted features can be fed into the model, however Deep Learning models can also learn relevant features from raw data. Deep Learning algorithms benefit when trained with large amounts of data, since they can learn more relevant characteristics when more samples are provided. Deep Neural Networks (DNNs) have been developed and advanced during the last decade. Their research increased since the published work of Geoffrey E. Hinton and R. Salakhutdinov in 2006 [19], which presented a novel approach to train a multi-layer neural network.

2.2.1 Artificial Neural Networks

Neural networks are composed of nodes or units, which are directly connected through links. The link from unit i to unit j has the purpose to propagate the activation a_i in this direction. Each link has an associated weight value w_i, which regulates the strength and sign of the connection. Each unit j first computes a weighted sum of its inputs and subsequently, an activation function g is applied to the result. Figure 2.3 shows a neuron representation through a mathematical model.

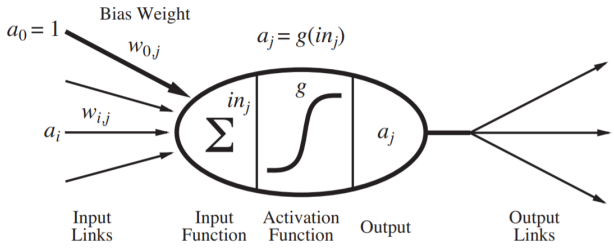


Figure 2.3: Neuron representation through a mathematical model, proposed in [20]

In order to build a network, the neurons need to be connected with each other. For that, two main techniques were developed: feed-forward and recurrent. The **Feed-forward Network** defines the connections in only one direction, and thus, the information is passed from the input nodes to the output nodes only. In opposite, the **Recurrent Network** feeds the output data back into the input, which implies that the activation levels form a dynamical system. As the network's response to a given input depends on its initial state and also may depend on previous inputs, the recurrent networks are able to support short-term memory. As such, these networks are more appealing, as they better replicate the model of the brain, however, they are more difficult to understand.

A perceptron network consists in a single-layer neural network which has a feed-forward architecture, and therefore, all inputs are directly connected to the outputs. In turn, a multi-layer perceptron is a feed-forward network composed of four main elements: an input layer with the input vector, the hidden processing layers, the weighted connections between nodes of adjacent layers and the output layer. Each neuron or processing element in a layer, is connected with all the processing elements in the next layer (feed-forward), until the neurons in the last hidden layer reach the neurons in the output layer. All these connections carry a weight value. Figure 2.4 presents this architecture.

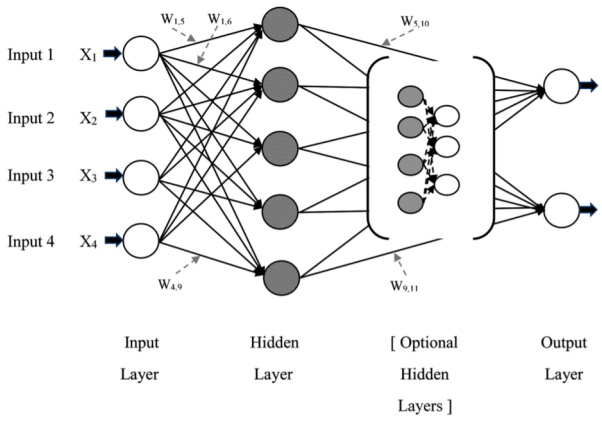


Figure 2.4: ANN architecture for a supervised learning multi-layer perceptron approach, extracted from [21].

With the arise of multi-layer networks, a major struggle appears from the additional hidden

layers of the network. Although the error at the output layer is clear, it is unknown at the hidden layers, since the training data does not specify the value that the hidden nodes should obtain. To overcome this problem, a technique known as **back-propagation** emerged. It was first introduced in the 1980's [22] and consists in flowing the information backward through the network to compute the gradient. The back-propagation algorithm allows the hidden layers to adapt the weights, by back propagating the errors from the output layers. As such, this technique tries to minimize the loss, through the analysis of the difference between the model predictions and the ground truth.

2.2.1.1 Optimization

The majority of deep learning algorithms require optimization. This consists in minimizing or maximizing a function $f(x)$ by modifying x. Most of optimization problems rely on minimizing the function $f(x)$, by finding the x^*, so that $x^* = argmin f(x)$. Considering a function $y = f(x)$, where x and y are real numbers, the derivative of this function, $f'(x)$, gives the slope of $f(x)$ at the point x. It defines how to scale a small change in the input in order to obtain the corresponding change in the output. Therefore, this technique is used to minimize a function, by specifying how to modify x to achieve a small improvement in y, which is known as **gradient descent**. Most algorithms use optimization functions based on the gradient descent.

The absolute lowest value of the function $f(x)$ is a **global minimum**, although, a point where $f(x)$ is lower than all its neighboring points is a **local minimum**. Multiple global minima can exist, and also, a local minima may not be a global minima. In this regard, the functions to be optimized may have many local minima that are not optimal. Therefore, the task is to find a value of f that is very low, but not necessarily minimal. Figure 2.5 shows an example of a function $f(x)$ with several local minima.

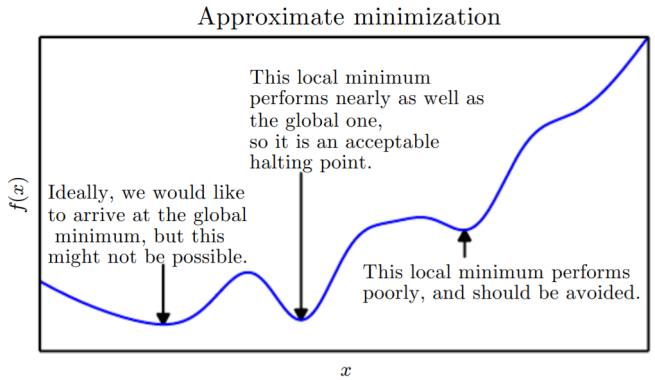


Figure 2.5: The optimization algorithms may fail to find a global minimum when there are multiple local minima. Therefore, in deep learning, a local minimum that correspond to significantly low values of the cost function is acceptable solution. Figure extracted from [18].

Functions with multiple inputs rely on the computation of partial derivatives. The concept is generalized to the gradient, which is a vector containing all the partial derivatives of a function, $\nabla_x f(x)$. In multiple dimensions, the critical points consist in points for which the gradient is zero

for every element. To minimize a multiple input function, the direction in which it decreases the fastest needs to be found. The gradient points uphill and the negative gradient points downhill, in order to decrease f by moving it in the direction of the negative gradient, which is known as **gradient descent**. Equation 2.1 presents its formulation.

$$x' = x - \varepsilon \nabla_x f(x) \tag{2.1}$$

Where ε is the **learning rate**, a positive scalar which defines the size of the step. Several approaches can be used to choose the learning rate value, such as to set ε to a small constant, or to evaluate the results of setting ε to different values and selecting the one that outputs the smallest objective function value.

2.2.2 The Learning System

A task in Machine Learning is generally described regarding how the learning system should process an example. The example consists in a collection of features in respect to an object or an event that the system is expected to process [18]. This operation is composed of three main stages:

- **Training Phase:** In this phase, the training data is used to fit the parameters of the classifier. The model learns the optimal combination of variables that generate a good predictive model [23]. The objective is to minimize the training error, in order to produce a trained model that generalizes well to new unknown data.

- **Validation Phase:** This phase consists in using a set of data different from the train set, called validation set. This allows to follow and evaluate the model performance during the train, in a set of new unobserved samples. This phase can avoid overfitting, given that the classification parameters can be adjusted during the training in order to improve the performance. A particular application is to use **Early Stopping**, which consists in stopping the model training phase when the performance stops improving. Another technique is the **Reduce LR On Plateau**, which reduces the learning rate when the results obtained computing an evaluation metric stopped improving. Usually both these methods are implemented to monitor the validation loss, in order to prevent the overfitting of the model.

- **Test Phase:** In this phase, the test set is used to assess the model. The model infers in the test samples producing the predicted classes, which are then compared with the actual true labels by computing a set of metrics.

There are many Machine Learning tasks, such as classification, regression, transcription, anomaly detection, synthesis and sampling, and denoising.

2.2.2.1 The Regression Task

The Regression task consists in understanding the relationship between dependent and independent variables. The algorithm predicts a continuous output value for a given input. There are different types of regression problems, one of the most relevant is Linear Regression.

- **Linear Regression:** This is the most used type of regression, which consists in finding the line that most closely fits the data. A linear function represented by a straight line with input x and output y has the form $y = w_1 x + w_0$, where w_0 and w_1 are weights [20]. The output value y is changed by the relative weight of one of the coefficients. Thus, w is represented by the vector $[w_0, w_1]$, defined in equation 2.2.

$$h_w(x) = w_1 x + w_0 \tag{2.2}$$

Evaluation Metrics

- **Intersection over Union (IoU):**

  The Intersection over Union (IoU) is a widely used evaluation metric in regression problems. It represents the overlap between two boxes, where as greater the region of overlap is, the higher the IoU score. This metric is mostly used for object detection applications, where a model is trained to output a bounding box that perfectly fits around an object. Equation 2.3 presents its formulation, where B_P is the predicted bounding box and B_{GT} ground truth bounding box. This consists in the area of overlap between the predicted box and the ground truth box divided by the are of union between them.

$$IoU = \frac{B_P \cap B_{GT}}{B_P \cup B_{GT}} \tag{2.3}$$

  Figure 3.6 shows a visual aid of the IoU formulation. The blue box is the predicted bounding box and the orange box is the ground truth bounding box.

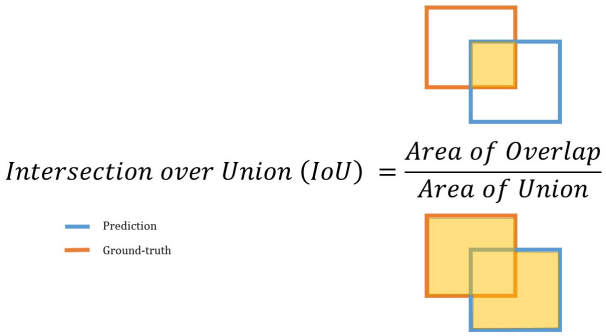


Figure 2.6: Intersection over Union (IoU) formulation, represented through a visual aid. The blue box refers to the predicted bounding box and the orange box refers to the ground truth bounding box.

- **Precision**

  The precision is a performance metric that computes the proportion of positive samples that were correctly classified in the total number of positive samples predicted. It is expressed as the number of true positives divided by the sum of the number of true positives and false positives, equation 2.4.

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

- **Recall**

  The recall, also known as sensitivity, True Positive Rate (TPR) or hit rate, is a performance metric that expresses the positive samples that were correctly classified in the total number of positive samples. It is computed by diving the number of true positives by the sum of the true positives and the false negatives, equation 2.5.

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

- **Precision-Recall Curve:** The Precision-Recall curve provides a visual evaluation of the precision in the y axis and the recall in the x axis, for a specific class at different probability thresholds. To overcome the impact caused by small variations in the ranking of detections, the interpolation precision was introduced in [24]. It is calculated at each recall level r by selecting the maximum precision measured at each r, as shown in equation 2.6.

$$P_{interp\,(r)} = max\ p(\widetilde{r}) \tag{2.6}$$

  Where $p(\widetilde{r})$ is the measured precision at recall $\widetilde{r}$.

- **Average Precision (AP):**

  The Average Precision (AP) is a widely used metric to evaluate object detectors. It returns the average precision at all possible thresholds, which is computed as the area under the Precision-Recall curve. Thus, this is a useful metric to measure the performance of the models, without considering any specific decision threshold. This metric is computed for each class separatly. Equation 2.7 presents its formulation.

$$AP = \frac{1}{N} \sum_{r=1}^{N} P_{interp\,(N)} \tag{2.7}$$

  This calculation is done by segmenting the recalls evenly to N parts and performing the interpolated precision, P_{interp}.

- **Mean Average Precision (mAP):**

Another commonly used metric to evaluate object detectors is the Mean Average Precision (mAP). It computes the mean of all Average Precisions obtained for each class. However, in some state-of-the-art proposals the Average Precision is the mean of all categories and thus, is the same computation as Mean Average Precision. Equation 2.8 shows its definition.

$$mAP = \frac{1}{C}\sum_{i}^{N} AP\,(i) \tag{2.8}$$

Where C refers to the number of classes and $AP\,(i)$ is the Average Precision of class i.

2.2.2.2 The Classification Task

Classification refers to the statistical problem of identifying to which category an observation belongs. Considering a set of categories, the observation is classified as belonging to one of the categories, based on its properties and features. In Machine Learning, the observations are known as instances and the set of possible categories are known as classes, targets or labels. The learning algorithm produces a function $y = f(x)$, where the model assigns an input x to a class y.

The implementation of a classifier is usually performed in a supervised manner, and it comprehends two types of problems:

- **Binary Classification:** This type of task consists in assigning an observation to one of two predefined classes, considering a set of attributes [25]. The classification model trained in the training phase predicts the true classes of the unknown samples, which outputs continuous or discrete values. The discrete output represents the predicted discrete class label and the continuous output represents the probability of the sample to belong to the each of the classes [26].

- **Multi-class Classification:** This type of problem refers to the classification of an instance into more than two classes. In this case, multi-label classification is used to assign a sample to one or more classes.

Evaluation Metrics

In order to evaluate the performance of a classifier, the evaluation metrics are used. They measure the performance of the model, by comparing the model's predicted classes with the actual true classes. The most widely used metrics for binary and multi-label classification problems are following described.

- **Confusion Matrix**

  The confusion matrix is a particular table layout that provides a visual representation of the model performance. The matrix is composed of columns that represent each instance of the actual classes and rows that represent each instance of the predicted classes. However

it can also be represented as the transpose of the described layout. The name confusion refers to the simplicity of observing where the model is confusing two classes. Particularly, in **binary classification problems**, the confusion matrix is composed of a 2×2 table, that denotes the number of False Positives, False Negatives, True Positives, and True Negatives.

The diagonal of a confusion matrix represents the correct predictions of each class, which is shown by the green positions in figure 2.7. In this case, a positive sample classified as positive is counted as True Positive (TP), the same sample classified as negative is considered a False Negative (FN). In turn, a negative sample classified as negative is considered True Negative (TN) and the same sample classified as positive, it is counted as False Positive (FP).

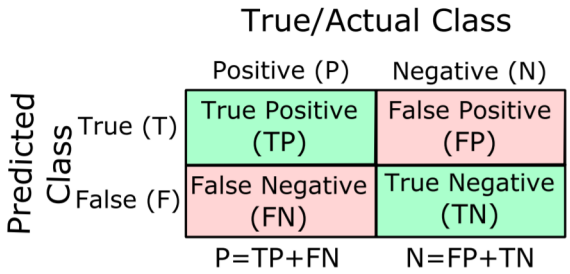


Figure 2.7: Example of a confusion matrix computed from a binary classifier, extracted from [26]. The classes Positive (P) and Negative (N) are defined and the predicted output is True (T) or False (F).

In **multi-class problems**, the confusion matrix allows a more detailed analysis of the model's misclassification. The classes that are more often misclassifed can be easily detected, which is particularly useful when dealing with **unbalanced** datasets.

In these type of problems, the confusion matrix includes more than two classes and has a slightly different representation of the TP, FP, TN and FN. Figure 2.8 presents the confusion matrix of a multi-class classification problem composed of three classes: A, B, and C. TP_A stands for the True Positive samples in class A, and E_{AB} consists in the samples belonging to class A that were misclassified as class B. The False Negatives in class A are the sum of all samples that were incorrectly classified as classes B or C, E_{AB} and E_{AC}. Therefore, the False Negatives of any class located in a column are calculated by the sum of the errors in that column. The False Positives in class A, F_{PA}, are the sum of all samples misclassified as class A: $FP_A = E_{BA} + E_{CA}$. Thus, the False Positives for a predicted class located in a row are the sum of all errors in that row.

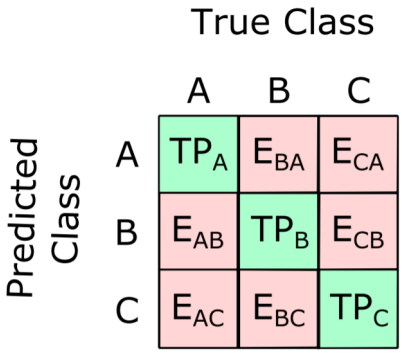


Figure 2.8: Example of the confusion matrix computed from a multi-class classifier, extracted from [26].

- **Accuracy**

  The Accuracy is a widely used evaluation metric in classification problems. It compares the correctly classified samples with the total number of samples contained in that set of data. This metric is computed through the sum of the number of true positives with the number of true negatives, divided by the sum of the number of true positives, true negatives, false positives, and false negatives, equation 2.9. Therefore, the accuracy measures the overall performance of a model.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.9}$$

  For multi-class classification problems, the accuracy can be expressed as the number of correctly predicted samples divided by the total number of samples:

$$Acc = \frac{Correctly\ Predicted\ Samples}{Total\ Number\ of\ Samples} \tag{2.10}$$

- **F1 Score**

  The F_1 score, also known as F-measure, computes the harmonic mean of precision and recall, as in equation 2.12. The output consists in a value between 0 and 1, where the high values denote high classification performance and vice-versa.

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.11}$$

  This metric has another variant, F_β-measure, which represents the weighted harmonic mean between precision and recall, equation 2.12. Moreover, it is affected by the changes in the class distribution.

$$F_\beta = (1 + \beta^2) \times \frac{Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{2.12}$$

2.2.3 Underfitting and Overfitting

The main goal of Machine Learning algorithms is to achieve a good performance in new unobserved data, which is different from the input training data. This ability is known as generalization. During the training, an error measure can be computed on the training set which is expected to be reduced, consisting in the training error. Similarly, the generalization error also called the test error, which consists in the expected value of the error on a new input, is expected to be as low as possible. The generalization error is usually estimated by measuring its performance on a set of examples collected separately from the training set, which is known as the **validation set**.

There are two main factors that determine the behaviour of a model: the ability to achieve a low training error and the capacity to obtain a small difference between the training and testing errors. These two aspects relate to the main challenge in machine learning algorithms: **underfitting** and **overfitting**. Underfitting refers to a model that is not able to achieve a sufficiently low error on the train set. In turn, overfitting consists in a too large gap between the training error and the testing error. Figure 2.9 presents three models as examples of underfitting, appropriate capacity and overfitting. Therefore, in order to control the probability of a model to underfit or overfit, its capacity can be modified. This implies that through a wide variety of functions that can be integrated in the model, its behaviour can change. In general, models with low capacity can have more difficulties to fit the training data and models with high capacity can overfit, as instead of learning general properties, they memorize specific characteristic of the samples in the training set which are not suitable in the testing set. As such, the performance of a Machine Learning model is better when its capacity is adequate for the complexity of the task to perform and for the amount of existent training data.

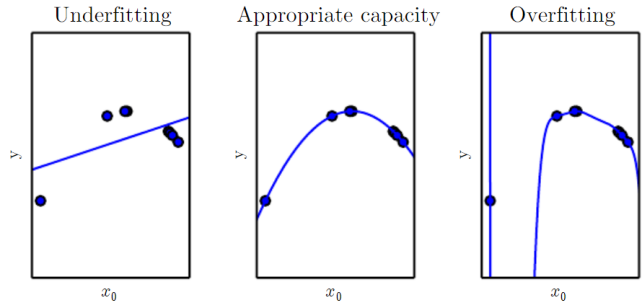


Figure 2.9: Example of the fit of three models in the training set. (Left) A linear function fit to the data experiences underfitting, once the curvature present in the data can not be captured. (Center) A quadratic function fit to the data shows to generalize well the unobserved points. (Right) A polynomial function fit to the data experiences overfitting, since the model is too specific for these data points and it will not be suitable for unobserved samples.

A model that learns a function $f(x;\theta)$ can be regularized by adding a penalty to its cost function, which is known as **regularizer**. The concept of regularization consists in any modification made to the learning algorithm that intends to reduce its generalization error but not its training error.

Convolutional Neural Network

Convolutional Neural Network (CNN) was first proposed by [27]. The term convolutional states for the mathematical operation employed, which is a particular type of linear operation. These architectures consist in neural networks that employ a convolution operation instead of a general matrix multiplication, in at least one layer.

2.3 Object Detection Architectures

The architecture of object detectors can be divided into two categories: two-stage detectors and one-stage detectors. Two-stage methods perform the detection in two consecutive stages, by first generating the region proposals and then, classifying each proposed region. Conversely, one-stage methods propose predicted boxes directly from the input image [28].

2.3.0.1 Two-Stage Object Detection

The Two-Stage Object Detectors consists in two consecutive tasks. First, these methods propose a set of Regions of Interest (RoI), by applying selective search or Region Proposal Network (RPN) and second, they classify the region candidates. One of the most used architectures for object detection is the Fast R-CNN [29]. It is a two-stage object detector, that takes two inputs: the entire image and a set of object proposals. First, the image is fed into the network to produce a feature map. Then, a Region of Interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map, for each object proposal. Finally, each feature vector is passed to a sequence of Fully Connected (FC) layers that output both probability estimation and the bounding box regression. Fast R-CNN achieves a reduction of the time cost of both training and testing, when compared to the state of the art R-CNN network [30], which is accomplished by performing the feature extraction once for the entire input image.

Faster R-CNN [31] arises as an improvement to the Fast R-CNN, by introducing a RPN as the region proposal method. Therefore, the first step consists in a RPN, which takes an image as input and outputs a set of rectangular object proposals, each assigned with an objectness score. The second step consists in the Fast R-CNN detector that uses these proposed regions. As previously described, the region proposals are merged with the feature map and a RoI pooling layer is applied to extract the feature vector and finally perform the classification. Figure 2.10 presents an overview of the Faster R-CNN.

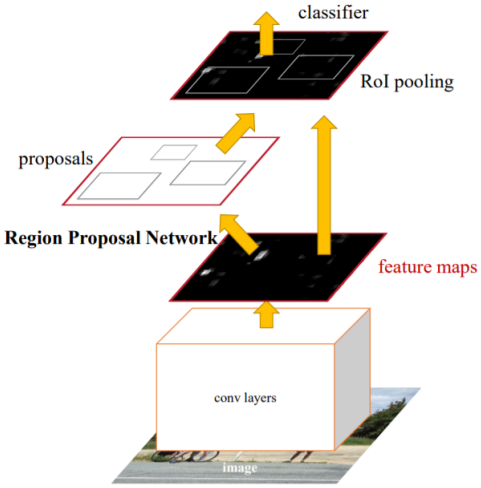


Figure 2.10: Overview of the Faster R-CNN, composed of a RPN module which serves as the 'attention' of the network.

2.3.0.2 One-Stage Object Detection

One-stage detectors consist in directly proposing predicted boxes from the input images, skipping the region proposal step. Although these methods achieve lower performance than the two-stage detectors, their execution time is faster, which makes them suitable for real-time applications.

You Only Look Once (YOLO) [32] is a novel approach for object detection, first published in 2015. It proposes using a single neural network to simultaneously predict bounding boxes and class probabilities from the whole images at once. First, the input image is divided into an S × S grid. If the center of an object is located into the grid cell, the grid cell performs object detection on itself. For each grid cell, B bounding boxes and their confidence scores across all classes are predicted, which express the model confidence regarding the box containing an object, as well as the accuracy of the box location prediction. Therefore, a cell without any object has a confidence score equal to zero. Otherwise, the confidence score consists in the Intersection over Union (IoU) between the predicted box and the ground truth. Besides that, each grid cell predicts class probabilities, if that grid cell contains an object. Finally, a class-specific confidence score is computed for each box, which consists in the multiplication of the conditional class probabilities and the individual box confidence predictions. An overview of this process is depicted in figure 2.11.

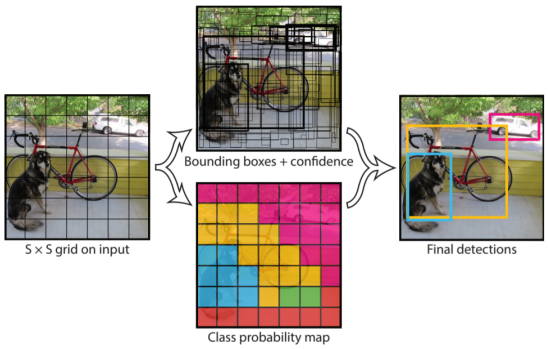


Figure 2.11: Overview of the YOLO architecture, extracted from [32]. This approach first divides the image into an *S* x *S* grid, predicting *B* bounding boxes for each grid cell, confidence for those boxes, and C class probabilities.

The Single Shot MultiBox Detector (SSD) [33], proposed in 2016, outperforms YOLO in terms of both detection accuracy and running time. This approach is based on a feed-forward convolutional network that outputs a fixed-size set of bounding boxes and their corresponding class scores. A standard VGG-16 network is used as the base network, followed by an additional auxiliary network structure and finally a Non-Maximum Suppression step to produce the final detections. More specifically, taking an input image, the model evaluates a small set of default boxes with different aspect ratios for each location in several feature maps with different scales. For each default box, the model predicts the shape offsets and the confidence scores for all object categories.

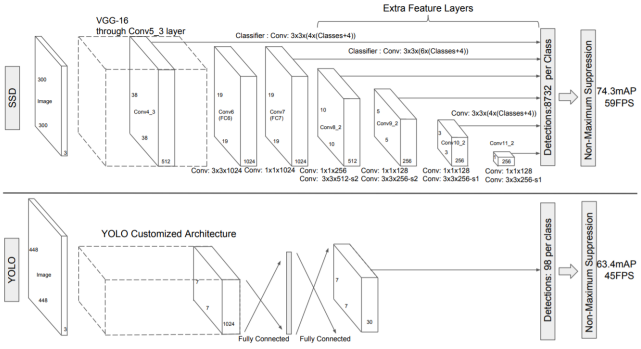


Figure 2.12: Comparison between the SSD and the YOLO approaches, extracted from [33]. The SSD approach adds a set of feature layers to the end of the base network, which predict the offsets to default boxes of different scales and aspect ratios, as well as, their confidence scores. The SSD detector significantly outperforms the YOLO approach in both accuracy and speed, when tested with the PASCAL VOC2007 dataset.

Chapter 3

Point Cloud Processing for 3D Object Detection

This chapter presents the point clouds representation in the context of 3D object detection. A contextualization of the main concepts related with point clouds is provided, the benchmark object detection datasets in the context of Autonomous Driving are introduced and their evaluation metrics described. Moreover, the most relevant state-of-the-art architectures are presented and a final overview and comparison of these methods is conducted. An overview of the sections described is hereby listed:

- **Point Cloud:** Point clouds representation in the context of 3D object detection, section 3.1.
- **Benchmark Datasets:** Most relevant object detection datasets in the context of Autonomous Driving, section 3.2.
- **3D Object Detection Task:** Main concepts related with 3D object detection, section 3.3.
- **Evaluation Metrics:** Commonly used metrics to evaluate 3D object detection methods, section 3.4.
- **State-of-the-Art Approaches:** Relevant 3D object detection architectures, section 3.5.
- **Sensor Fusion:** Types of fusion architectures and their most relevant methods in the state-of-the-art, section 3.6.
- **Final Remarks:** Analysis of the most relevant approaches in 3D object detection, section 3.7.

3.1 Point Cloud

A Point Cloud consists in a collection of data points in 3D space. It measures the shape of physical objects, where each point is composed of a set of coordinates (x, y, z). Generally, point clouds are generated using methodologies developed for photogrammetry or remote sensing. Photogrammetry is the science of obtaining reliable information about physical objects and the environment, by recording, measuring and interpreting photographic images. Also, a combination of photographs taken at different angles can be used to generate point clouds. Moreover, the Remote Sensing, which consists in collecting data of the Earth's surface by use of satellites or aircrafts relies on specific sensors to obtain the information. For that, **LiDAR** sensors are one of the most commonly used technologies. Light Detection and Ranging (LiDAR) is a methodology to determine the distance to an object, by targeting it with a laser and measuring the time that the reflected light takes to return to the receiver. This technology is widely used for 3D representations of areas on the planet surface, to generate high-resolution maps, with applications such in surveying, geodesy, archaeology, geography, geology, control and navigation in Autonomous Driving.

3.1.1 Properties

Point cloud representations can be sparse, occluded, and low resolution, which turns the identity of points ambiguous and thus, objects recognition a difficult task [34]. When using a LiDAR sensor to acquire data in a single viewpoint, which occurs in the majority of Autonomous Driving applications, the 3D point cloud automatically experiences **occlusion**. The sensor captures the surrounding scene, by measuring the time that the reflected light needs to reach the target, and thus, the reflected light beams belongs to the surface of nearest objects relative to the sensor. Therefore, the objects located behind the surrounding environment will not be fully captured by the LiDAR, as they can be visually blocked by other objects or by themselves, resulting in partial or complete invisibility of the object.

Moreover, point clouds typically produce a **sparse** representation, since the sample points are not uniformly distributed in the 3D space. A real-time LiDAR scan varies the point density produced over the range. Faraway objects are represented with sparser points than nearby objects. For this reason, resulting point clouds can not provide a detailed information regarding the 3D object shape [35]. Besides that, point clouds have a **lower resolution** when compared to the typical output of other 2D sensors. This means that there is a sparsity between points, leading to a major difficulty in identifying objects in the scene, since smaller objects, such as a pedestrian in an urban scene sometimes can not even be detected or recognizable, which usually does not occur with bigger objects, such as cars. Figure 3.1 shows an example of a point cloud.

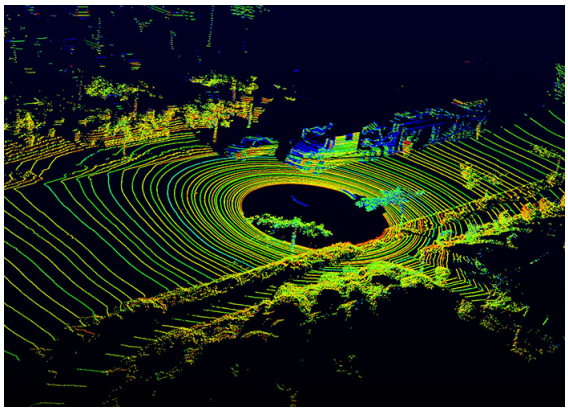


Figure 3.1: Point Cloud captured by a LiDAR scan, adapted from [36].

3.1.2 Representation

The representation of point clouds follows an unordered set of 3D points, where each point is described by four values. The first three parameters express the cartesian or spherical coordinates regarding the LiDAR coordinate frame, and the other value relates to the light pulse intensity, commonly represented by r. Therefore, this can be expressed as a set of N points: $S = \{(p_i, r_i)\}_{i=0}^{N}$, where p_i represents the 3D coordinate of a point $p_i = [x_i, y_i, z_i] \in \mathbb{R}^3$ and r_i represents the intensity [35].

- **Raw points**

  The simpler representation of a 3D point cloud consists in a matrix, where each row describes the coordinates of each point in the set S. Although this representation preserves all the information in the original set of points, it does not explore any geometric property of 3D points.

- **3D voxelization**

  This representation consists in a discretization of the 3D space into voxels to represent a 3D point cloud. An easy approach to apply 3D voxelization is to partition the 3D space into equally-spaced non overlapping voxels from each of the three dimensions, as presented in figure 3.2 (a). Then, a voxel is set to 1 if it is occupied and set to 0 otherwise. The 3D voxelization representation has several advantages, such as the hierarchical structure where the resulting voxels are inserted, the uniform spatial size of all the voxels and also being an enabler for application of standard techniques as 3D convolutions. However, it also presents some disadvantages, such as not considering specific properties of 3D point clouds, exhibiting a deeply sparse representation where most voxels are empty and the trade-off between resolution and memory consumption.

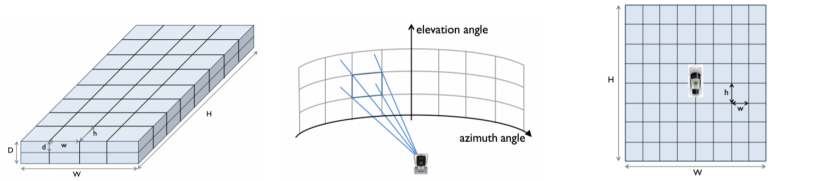
- **Range view**

  As previously explained, a real-time LiDAR scan performs a series of range measurements from a single location with certain angular field of view, as shown in figure 3.2 (b). The

Range view consists in organizing the 3D points from the LiDAR to a 2D range-view image, where the x axis refers to the azimuth angle of the LiDAR sensor and the y axis refers to the elevation angle of the laser beams. Each pixel in the 2D range-view image corresponds to a frustum in the 3D space and its value is the range from the LiDAR to the closest 3D point inside the frustum.

- **Bird's eye view**

  The Bird's eye view (BEV) consists in a particular representation of 3D voxelization, which omits the height dimension. This method projects 3D voxels to a BEV image. In other words, the height of each generated voxel cell relates to the height of the points on the 3D space that fall within that cell. As a result of the discretisation along the x and y axis, a top-view 2D image is generated, where each pixel in the BEV image represents a pillar space, as depicted in figure 3.2 (c).



(a) 3D voxel-based representation. (b) Range-view-based representation. (c) Bird's-eye-view-based representation.

Figure 3.2: Overview of the most used approaches to discretize the 3D space, extracted from [35]. (a) The 3D voxelization representation consists in discretizing the 3D space into equally-spaced non overlapping voxels from each of the three dimensions. (b) The range-view representation discretizes the 3D space along the azimuth and elevation angles. (c) The Bird's Eye-view representation consists in discretizing the 3D space along the x and y axes, ignoring the height dimension.

3.2 Benchmark Datasets

The training phase of Deep Neural Networks for perception applications in the context of Autonomous Driving rely on large datasets. There are a set of benchmark datasets available that feature multi-modal sensor data, in particular LiDAR point clouds and RGB images.

3.2.1 KITTI Vision Benchmark Suite

The KITTI Vision Benchmark Suite is a dataset projected to support the development of Autonomous Driving applications. It was first published in 2012 by [37], and became one of the most used state-of-the-art datasets for Autonomous Driving systems. The dataset was recorded while driving around a mid-size city, in rural areas and on highways, in Karlsruhe, Germany. A vehicle equipped with a recording platform was used as the setup system. It is composed of two high resolution stereo camera systems (grayscale and color), a Velodyne HDL-64E sensor with 64 layers of scanning and a localization system. Figure 3.3 shows the system setup. The objects were manually labeled in the 3D point clouds, acquired from the Velodyne system and then projected

back into the 2D image. These results in 3D bounding boxes of objects such as cars, vans, trucks, trams, pedestrians and cyclists. A difficulty is assigned to each object regarding the minimum bounding box height, the occlusion level and the maximum truncation, as shown in table 3.1.

Definition	Easy	Moderate	Hard
Min Bounding Box Height	40 Px	25 Px	25 Px
Occlusion Level	Fully visible	Partly occluded	Difficult to see
Maximum Truncation	15 %	30 %	50%

Table 3.1: Definition of the object difficulty in the KITTI dataset. It is composed of Easy, Moderate and Hard, which are defined regarding the minimum bounding box height, the occlusion level and the maximum truncation.

Figure 3.3: Recording platform with sensors, extracted from [37]. A stereo camera, a 360º Velodyne scan and a GPS module were the setup used to capture the dataset.

3.2.2 Waymo Open Dataset

The Waymo Open Dataset consists in a publicly available dataset, published in 2019 [38]. It was captured across a range of urban and suburban areas and it comprehends 1150 scenes of 20 seconds. A setup system composed of synchronized and calibrated high quality LiDAR sensor and camera was used to capture scenes across downtown and suburban areas in diverse cities, such as San Francisco, Los Angeles and Seattle. This is a robust dataset, with diverse scenarios, as daylight and night time recorded scenes, as well as varied weather conditions. Also, diverse objects were included, providing labels for four object classes: Vehicles, Pedestrians, Cyclists and Signs. The 3D ground truth bounding boxes in the LiDAR point clouds and the 2D bounding boxes in the images captured by the camera were manually and separately annotated.

3.2.3 nuScenes

The nuScenes dataset is a large-scale publicly available dataset for Autonomous Driving, published in 2019 [39]. It is the first dataset captured using a fully autonomous vehicle sensor suite composed of 6 cameras, 5 radars and 1 LiDAR, where a field of view of 360°is provided from all the sensor modalities. Figure 3.4 shows the system setup in the data collection vehicle. The dataset was recorded across two cities with different characteristics: Boston and Singapore, during both day and night. The 2D images are annotated with bounding boxes and instance masks for foreground objects, as well as semantic segmentation masks.

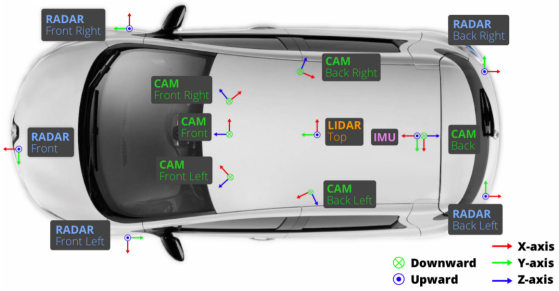


Figure 3.4: Sensor setup in the data collection vehicle, extracted from [39].

3.3 3D Object Detection Task

2D object detection is a mature field that has been a focus in the Computer Vision area. In contrast, 3D object detection still has a long way to go in development and research. Moreover, a perception of the scene is crucial to develop self-driving vehicles, as these systems need to fully understand the surrounding environment, which includes the diverse elements in the road, such as cars, cyclists and pedestrians. In this regard, the 3D object detection for Autonomous Driving takes the point cloud obtained by the LiDAR and produces oriented 3D bounding boxes for each target element in the scene. Therefore, three parameters are predicted regarding the bounding box estimation: the **Bounding Box Shape** consisting in the dimensions of width, length and height (w, l, h), in which the length grows in the direction of the vehicle, being higher than the width, the **Bounding Box Position**, which is the centre of the box $(x_c; y_c; z_c)$ represented in cartesian coordinates, regarding the LiDAR coordinate frame, and the **Bounding Box Orientation**, computed with respect to the object reference system and defined by the heading angle of the predicted bounding box, q.

3.3.1 Regression Targets with Anchor Boxes

The bounding box parameters are used to determine the regression loss. Generally, transformed versions of these 7 degrees of freedom, also known as regression targets, are predicted. This way, the dynamic range of model predictions is increased. When the network has to detect different classes of objects with different sizes, such as cars, bicycles and pedestrians, the use of

anchor boxes is required. A set of anchor boxes with different scales and aspect ratios are defined as parameters of the model, in order to best match the class objects with different dimensions. The goal is to focus each type of anchor box in a specific class of object. Therefore, after the training process, each type of anchor box is specialized in a class object that best corresponds the scale and aspect ratio of the anchor box.

The 7 regression targets corresponding to center location Δ_x, Δ_y, Δ_z, three dimensions Δ_l, Δ_w, Δ_h, and the rotation Δ_θ, are computed by the following formulas, as stated in [40]:

$$\Delta_l = log\left(\frac{l^{gt}}{l^a}\right), \Delta_w = log\left(\frac{w^{gt}}{w^a}\right), \Delta_h = log\left(\frac{h^{gt}}{h^a}\right) \tag{3.1}$$

$$\Delta_x = log\left(\frac{x_c^{gt} - x_c^a}{d^a}\right), \Delta_y = log\left(\frac{y_c^{gt} - y_c^a}{d^a}\right), \Delta_z = log\left(\frac{z_c^{gt} - z_c^a}{d^a}\right) \tag{3.2}$$

$$\Delta_\theta = \theta^{gt} - \theta^a \tag{3.3}$$

Where gt stands for the ground truth and a stands for the anchor box. $d^a = \sqrt{(l^a)^2 + (w^a)^2}$ refers to the diagonal of the base of the anchor box. These formulations are general examples of regression targets, which can suffer modifications regarding different 3D object detectors. Equation 3.1 computes the bounding box dimensions, equation 3.2 calculates the bounding box position and equation 3.3 defines the heading angle.

3.3.2 Data Augmentation

Data augmentation consists of techniques used to overcome overfitting, as it operates as a regularizer, by improving the model generalization. The amount and diversity of training data is increased by slightly modifying samples of already existing data. This can be achieved through geometric transformations applied to the sample data, which consists in operations such as flipping, color modification, cropping and rotation.

In 3D LiDAR point clouds, the augmentation methods are influenced by the original augmentation methodologies designed for the image domain. Point clouds data require augmentation methods to overcome their physical constrains regarding the propagation of the laser beams. Taking data sparsity as an example, point clouds are expected to become sparser the higher the distance to the LiDAR scan, and thus, data augmentation methods need to overcome this limitation to effectively simulate the real scenario. The most common transformations of data augmentation for point clouds consist in rotating around z axis of the LiDAR coordinate frame, flipping in the x-z and y-z planes of the LiDAR coordinate frame, applying additive Gaussian noise to LiDAR points and performing frustum dropout. In addition, a novel approach was introduced by [1], the ground truth augmentation, which intends to overcome the existence of too few ground truths for the point cloud. This technique consists in randomly selecting during training, several ground truths from all existent ground truths and including them into the training point cloud. Therefore, the number of objects contained in the point cloud can be greatly increased and also, it can simulate objects

in different environments. Figure 3.5 shows some data augmentation transformations applied to point clouds.

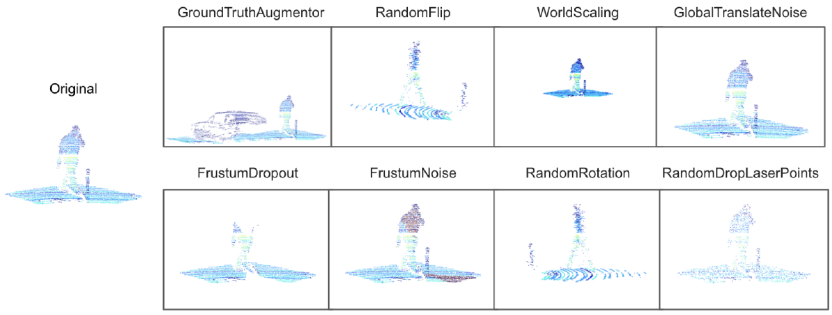


Figure 3.5: Illustration of the data augmentation transformations for LiDAR point clouds, extracted from [41]. In the top images, examples of ground truth sampling, random flips, world scaling and global translation noise are presented. In the bottom images, examples of frustum dropout and noise, random rotation and random drop of laser points are shown.

3.4 Evaluation Metrics

As explained in section 2.2.2.1, object detection models are evaluated in terms of their inherent regression and classification tasks. The metrics to evaluate 3D object detectors rely on an extension of the techniques used to evaluate 2D object detection models, as well as new metrics particularly used in this task. This section covers the commonly used techniques for this purpose.

3.4.1 3D Intersection over Union (IoU)

The 3D Intersection over Union (IoU) is applied to 3D object detection to evaluate the predicted bounding box, whether it is a True Positive (TP) or a False Positive (FP). This metric is defined as the intersection of the bounding box prediction (B_p) and the ground truth (B_{gt}), divided by the union of the prediction and the ground truth. When both predicted and ground truth boxes only have a yaw rotation angle, the 3D IoU can be formulated as the multiplication of the top-view area (A_0) and the height (h_0), divided by the sum of the predicted bounding box volume (V_p) and the (V_{gt}) ground truth box volume, subtracted by the multiplication of the top-view area (A_0) and the height (h_0). Equation 3.4 presents this calculation:

$$IoU = \frac{B_P \cap B_{GT}}{B_P \cup B_{GT}} = \frac{A_0 \times h_0}{V_{gt} + V_p - A_0 \times h_0} \tag{3.4}$$

Where V_p stands for the volume of the predicted box and V_{gt} stands for the volume of the ground truth box, A_0 refers to the top-view area and h_0 refers to the height.

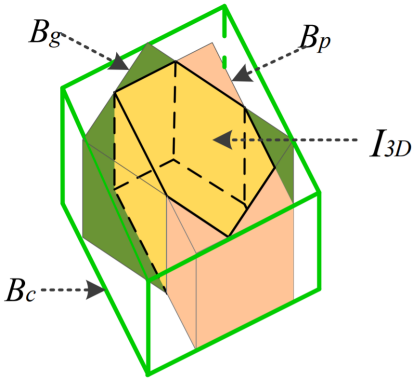


Figure 3.6: Representation of the 3D IoU, extracted from [42]. The pink and the green boxes represent the predicted bounding box B_p and the ground truth B_g, respectively, and the yellow defines the 3D intersection between them I_{3D}.

3.4.2 3D Average Precision

The Average Precision is a widely used metric for object detection. In the KITTI benchmark this metric is used to evaluate the model performance, along with the Average Orientation Similarity (AOS) [43]. The 3D Average Precision consists in the average of the maximum precision achieved s, with at least a recall value r for a defined range of recall values. These value were defined by the authors of KITTI as 40 recall positions, rather than the previously used 11 recall positions. Equation 3.5 shows this formulation:

$$AP_{3D} = \frac{1}{40} \sum_{r \in \{\frac{1}{40}, \frac{2}{40}, \ldots, 1\}} \max_{\tilde{r}:\tilde{r}>r} s(\tilde{r}) \tag{3.5}$$

3.4.3 Average Orientation Similarity (AOS)

In contrast to 3D Average Precision, the Average Orientation Similarity computes the correctness of the bounding boxes orientations. At a specific recall rate r, $D_{(r)}$ represents the set of all existing objects. $\Delta_\theta^{(i)}$ represents the difference between the estimated and the real orientation of the boxes and δ_i indicates whether the predicted box is a true positive or not.

$$AP_{AOS} = \frac{1}{40} \sum_{r \in \{\frac{1}{40}, \frac{2}{40}, \ldots, 1\}} \max_{\tilde{r}:\tilde{r}>r} \tilde{s}(\tilde{r}) \tag{3.6}$$

$$\tilde{s}(r) = \frac{1}{D_r} \sum_{i \in D_r} \frac{1 + cos\left(\Delta_\theta^{(i)}\right)}{2} \delta_i \tag{3.7}$$

3.4.4 NuScenes Detection Score (NDS)

NuScenes Detection Score is a specific metric proposed by [39] to evaluate the model's performance on nuScenes dataset. It consists in measuring the detection performance, through the computation of *mAP* in equation 3.9, and quantifying the quality of the detections in terms of box

location, size, orientation, attributes, and velocity, by computing the five mean True Positive metrics (mTP). These metrics consist in the Average Translation Error (ATE), which is the Euclidean center distance in 2D (measured in meters); the Average Scale Error (ASE), which consists in the 3D intersection over union (IOU); the Average Orientation Error (AOE), which is the smallest yaw angle difference between prediction and ground truth (measured in radians); Average Velocity Error (AVE), which consists in the absolute velocity error; Average Attribute Error (AAE), which is the difference between 1 and the classification accuracy: $1 - acc$. For each TP metric, the mean TP (mTP) over all classes is computed, as presented in equation 3.10. Equation 3.8 presents the formulation of NDS.

$$NDS = \frac{1}{10}[5\ \ mAP + \sum_{mTP \in TP} 1 - (min(1, mTP))] \tag{3.8}$$

$$mAP = \frac{1}{|C||D|} \sum_{c \in C} \sum_{d \in D} AP_{c,d} \tag{3.9}$$

$$mTP = \frac{1}{|C|} \sum_{c \in C} TP_c \tag{3.10}$$

3.5 State-of-the-Art Approaches

The employment of Deep Learning for scene understanding using point clouds is a growing research field that has reached an increased interest in the last years [44]. As previously explained, the benchmark datasets contributed to the growth of 3D point clouds in deep learning. Diverse approaches have been proposed to address several problems regarding point clouds perception, such as 3D shape classification, 3D object detection and 3D point cloud segmentation. In this regard, an extensive study of the most relevant proposed architectures for point clouds learning is following presented. The state-of-the-art proposals for 3D object detection can be categorized by different representations. Therefore, these architectures can be divided into three categories: grid-based methods, point-based methods and graph-based methods [45].

3.5.1 Point-based Methods

Point-based methods, also known as set-based methods, intend to extract features from raw 3D point clouds. **PointNet** [46], a novel approach proposed in 2016, was designed to directly process unordered 3D point sets. It consists in a Deep Neural Network, developed to address several tasks, such as object classification, part segmentation and scene semantic parsing. The previous Deep Learning algorithms developed for 2D images are not suitable for 3D point clouds applications, as they have inherent data irregularities. In this regard, PointNet emerges as a pioneering work, by directly passing point clouds as inputs, using a set of Multi-Layer Perceptrons (MLPs), global

max-pooling, and spatial transformer networks. Thus, the network learns point-wise features independently and achieves permutation invariance. Figure 3.7 presents the feature extraction module of a lightweight PointNet architecture.

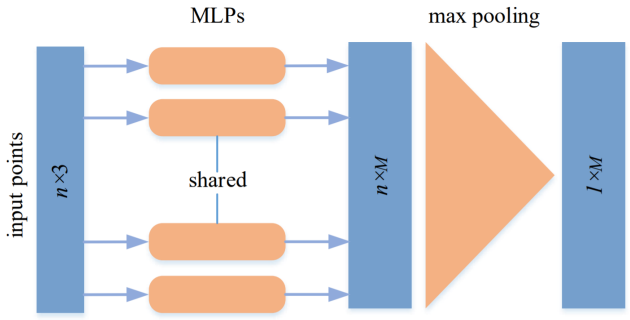


Figure 3.7: Overview of a lightweight PointNet architecture, extracted from [46]. n refers to the number of input points, M refers to the dimension of the learned features for each point.

PointNet++ [47] proposes some improvements to the PointNet architecture, by introducing a hierarchical network architecture. First, a sampling layer is employed, which applies the Farthest Point Sampling (FPS) to select a set of points from the input, where each selected point defines the centroid of local regions. Then, a grouping layer finds neighbouring points around the centroids and generates the local region sets. Finally, a PointNet layer is implemented, which encodes the region patterns into feature vectors.

3DSSD [48] is a novel proposal published in 2020, that presents a 3D single stage object detector. It achieves top-performing inference time, by removing the Feature Propagation layers and the refinement stage in PointRCNN [49], and introducing a fusion sampling strategy of Distance-FPS (D-FPS) and Feature-FPS (F-FPS). In order to completely explore the representative points, a candidate generation layer (CG) is employed. First, the representative points from F-FPS are shifted to generate candidate points. Then, each candidate point is considered a center and its surrounding points are found from the set of representative points from both F-FPS and D-FPS. Their features are extracted by employing a Multi-Layer Perceptron (MLP) network, and finally, the output is passed to an anchor-free regression head to predict 3D bounding boxes. Figure 3.8 shows an overview of the 3DSSD architecture.

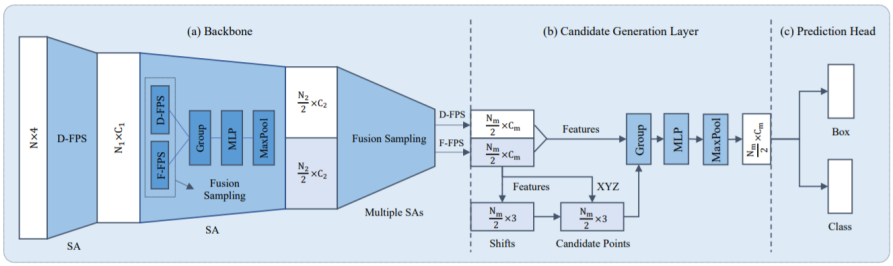


Figure 3.8: Overview of the 3DSSD architecture, extracted from [48]. It comprehends a backbone and a box prediction network, which includes a candidate generation layer and an anchor-free prediction head. (a) Backbone network, which generates global features for all representative points through several Set Abstraction (SA) layers with fusion sampling (FS) strategy, taking as input the raw point cloud. (b) Candidate generation layer (CG), which generates the features for representative points. (c) Anchor-free prediction head.

3.5.2 Grid-based Methods

The Grid-based methods, also known as 3D voxel-based methods convert a point cloud to a regular grid to be fed into a Neural Network. **PIXOR** [50] projects a 3D point cloud into a 2D Bird's Eye View (BEV) image, which is then fed into a 2D Convolutional Neural Network (CNN) for object detection. In contrast, the **VoxelNet** [40] proposes a novel approach to represent points in 3D voxels. A point cloud is divided into equally spaced 3D voxels and then, a unified feature representation is generated from a group of points within each voxel. This is done through the voxel feature encoding (VFE) layer, which is a newly introduced technique to encode the point cloud as a descriptive volumetric representation. The detections are finally performed by passing the resulting volumetric representation into a Region Proposal Network (RPN). The main disadvantage of this proposal is the high computational cost, since the resolution of the voxels grow, the memory consumption of 3D CNN grows cubically, which directly influences the inference time. Moreover, due to the point clouds sparsity, numerous empty voxels are generated, leading to unnecessary use of computational resources. **SECOND** [1] proposes some modifications to the VoxelNet architecture. It implements a 3D sparse convolution method, which significantly increases the speed of both training and inference. Also, a new form of angle loss regression is introduced to improve the orientation estimation performance. Therefore, this approach achieves lower inference speed, by removing the convolution operations in the empty voxels.

A novel approach, motivated by the previous works is **PointPillars** [51]. It was proposed in 2019 and it intends to achieve fast processing time, by transforming a 3D point cloud into 2D pillars. This is addressed through a new encoder which employs PointNets to learn a representation of point clouds structured in vertical columns (pillars) that enables the use of 2D convolutions. Therefore, instead of feeding voxels into 3D convolutions, this methods applies 2D convolutions to pillars, which enables the computation time reduction. The point cloud is first converted into a 2D pseudo-image, where its height is equal to the point cloud's height. The resulting features are passed to a simplified version of PointNet that outputs a tensor of size (C,P), where P is the

number of non-empty pillars per sample. After encoded, the features are scattered back to the pillar locations to create a pseudo-image of size (C,H,W) where H represents the height and W the width of the canvas. Following this methodology, a 2D CNN backbone is employed, which applies 2D convolutions. A detection head is finally employed to perform object detection, by using the Single Shot Detector (SSD), which outputs 3D bounding boxes. Figure 3.9 shows an illustration of the PointPillars approach.

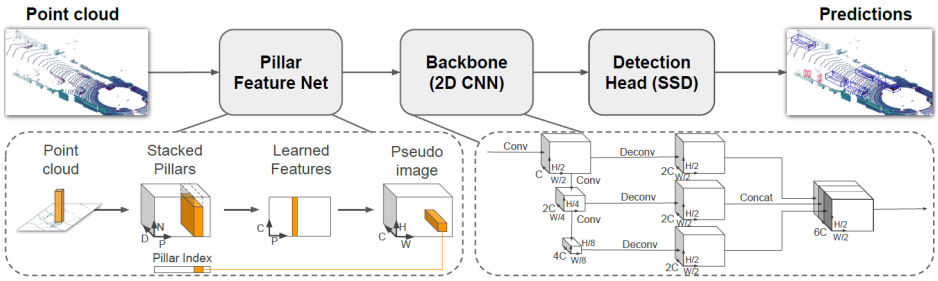


Figure 3.9: Overview of the PointPillars approach, extracted from [51]. The point cloud is transformed into a stacked pillar tensor and pillar index tensor. The stacked pillars are used by the encoder to learn a set of features that can be projected back into a 2D pseudo-image for a CNN. Finally, the detection head predicts 3D bounding boxes for objects.

Point-Voxel Feature Set Abstraction for 3D Object Detection (**PV-RCNN**) [52] takes advantage from the integration of both 3D voxel CNN and PointNet abstraction, by learning more discriminative point cloud features. It consists in a two-stage 3D detection framework, which achieves an accurate 3D object detection from point clouds. The previous works in 3D detection are based on either 3D voxel CNN with sparse convolution, which generate high-quality 3D object proposals, or PointNet-based, which capture more accurate contextual information. This proposal benefits from both methods, by applying a 3D voxel CNN with sparse convolution as the backbone. Two novel operations are implemented to perform the correspondence of the 3D object proposal features from the scene. First, the voxels of the overall scene feature volumes are transformed into a small number of feature keypoints, by employing the voxel-to-keypoint scene encoding. Then, the scene keypoint features are aggregated into RoI grids to perform the confidence prediction and location refinement.

3.5.3 Graph-based Methods

Graph-based Methods consist in generating edges for the graph based on the neighbors of each point in a point cloud, by considering each of the points a vertex of a graph [44]. Some researches have focused on the use of graph neural networks for point cloud classification and semantic segmentation. However, few studies propose a graph neural network for 3D object detection in point clouds. One of them is the **Point-GNN** [45], which introduces in 2020 a graph neural network to process point clouds as graph structures for 3D object detection. The point cloud is efficiently encoded in a fixed radius near-neighbors graph. For each vertex in the graph, this

method predicts the category and shape of the object that it belongs to. It proposes an auto-registration method that reduces translation variance, and also efficiently combines detections from multiple vertices, by implementing a box merging and scoring operation. Figure 3.10 presents an illustration of the Point-GNN architecture.

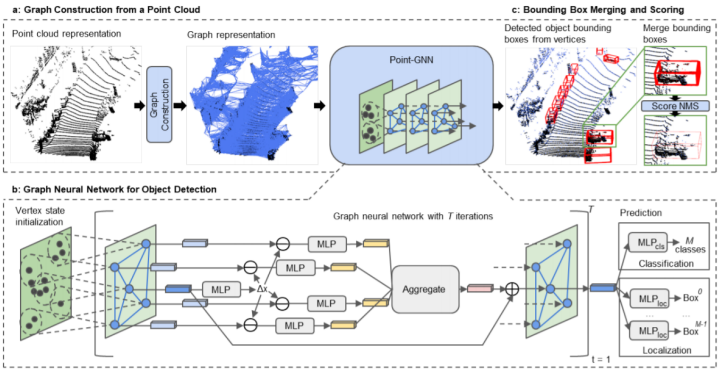


Figure 3.10: Architecture of the Point-GNN, extracted from [45]. (a) Graph generation from the point cloud; (b) Graph neural network for object detection; (c) Bounding box merging and scoring.

3.6 Sensor Fusion

The data acquired through a LiDAR sensor provides accurate depth information of the surrounding environment in the form of 3D points [53]. These sensors are robust to illumination changes and less affected by variation of weather conditions than visual cameras. However, current LiDAR technologies produce low resolution point clouds and cannot capture the textures of the objects, contrarily to visual cameras where it is provided as detailed texture information. Moreover, as previously explained, the generated sample points are not uniformly distributed in the 3D space, with points becoming more sparse for distant objects. For these reasons, object detection would benefit from a combination of the two types of data, providing robustness to a perception system. Therefore, the information acquired by the different data modalities are merged, following one of the sensor fusion strategies.

There are three main data fusion methods: the early fusion, which consists in merging raw data from both sensors and then feeding that to a network; late fusion, having two different networks for each type of data and then merging both outputs; and deep fusion, which is a more complex strategy that consists in joining early fusion and late fusion approaches. Figure 3.11 provides an overview of each of these fusion methods.

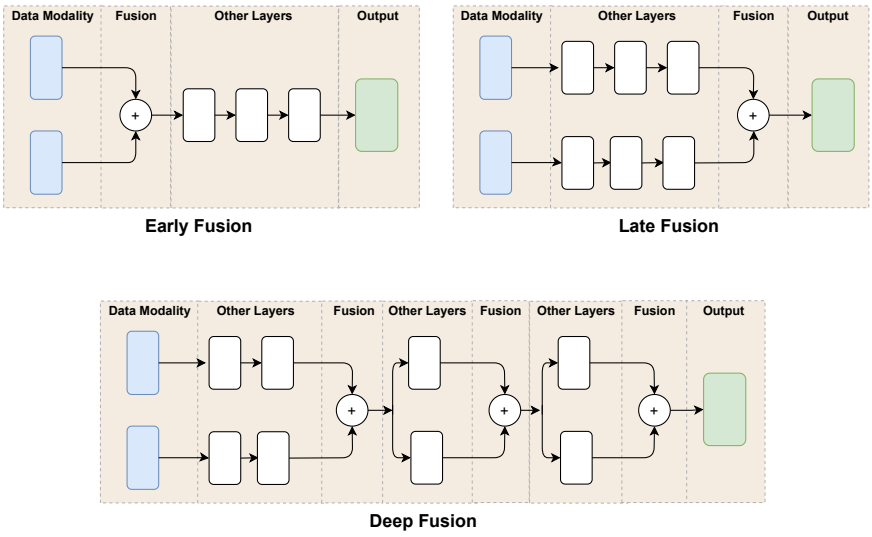


Figure 3.11: Overview of the different types of data fusion methods, adapted from [53]. It includes early fusion, late fusion and deep fusion.

3.6.1 Early Fusion

Early Fusion methods merge the raw or preprocessed data obtained from the different sensors before feeding them into a neural network. This strategy benefits from low memory cost, since it jointly processes the multiple sensor modalities and thus, only one neural network is employed. The network learns the combination of features from both modalities at an early stage, which allows to fully exploring the information from the raw data. However, this technique yields limitations regarding model inflexibility. Replacing an input sensor modality or changing input channels sizes, implies the complete retraining of the network. Moreover, early fusion suffers from spatial-temporal data misalignment, which signifies that when calibration errors occurs, data alignment is necessary. Also, different sensor modalities can have different sampling rates, however to pass the data into the network, they are required to be synchronized.

MVX-Net is a single-shot based architecture, published in 2019 by [54]. It combines the RGB and LiDAR data by implementing two early fusion based approaches: PointFusion and VoxelFusion. The PointFusion method consists in implementing a 2D object detection network to extract the high-level image features which encode semantic information. Then, by projecting each 3D point onto the image and aggregating the point with the projected location index, this algorithm correlates information about the object pose from 2D images to every 3D point. On the other hand, VoxelFusion aggregates features from the RGB image at the voxel level. First, the 3D space is divided into a set of equally spaced voxels and the points are grouped into these voxels. By projecting each non-empty voxel onto the image plane, a 2D region of interest (ROI) is produced. A pre-trained detector network is employed to output feature maps, and by pooling the features within the ROI, a feature vector is produced. Therefore, prior information from the 2D image is encoded at every voxel.

3.6.1.1 Sequential Fusion

Sequential Fusion is a particular type of early fusion technique, which merges the data from a sensor modality with the extracted information from the other sensor modality. The Frustum PointNets [55] architecture implements sequential fusion, using as input the RGB-D data. The 3D bounding frustum of the objects are extracted by translating the 2D bounding boxes to a frustum creating a 3D space for the detected object. Then, in the 3D space it performs object instance segmentation followed by 3D bounding box regression, through a PointNet [56] implementation. In contrast, Frustum ConvNet proposes a sequential fusion approach, which generates a set of frustums for each region proposal. By applying a PointNet implementation, it aggregates point-wise features, given the sequence of frustums and point association in feature vectors of individual frustums. These feature vectors are assembled as 2D feature maps. A Fully Convolutional Network (FCN) is then employed to fully fuse the features across the frustum axis. A more recent proposal, published in 2020, that relies on sequential fusion is PointPainting [57]. This approach takes the output of a 2D image semantic segmentation network to project it to the LiDAR points. By using semantic information, it enables the detection of small objects, otherwise easily misclassified. The class scores are assigned to each point and then, the resulting point cloud is passed to a LiDAR based algorithm. The experiments are done using the KITTI and the nuScenes datasets. For the KITTI experiments, the DeepLabV3+ network is applied to generate the segmentation scores. Figure 3.12 presents a scheme of this methodology.

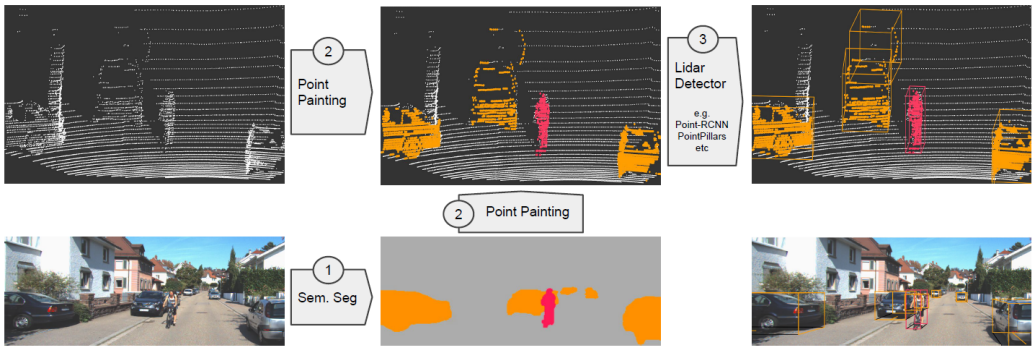


Figure 3.12: Overview of the MVX-Net PointFusion architecture, extracted from [57].

3.6.2 Late Fusion

Late Fusion strategy consists in having separated networks to learn relevant features for each data modality, and then, combine the outputs into a single parameter. As it takes a specific network for each type of data, this technique allows to include a new sensor modality without interfering with the other networks, with only the respective network needing to be trained. However, the fact of having multiple networks leads to higher computational cost.

3.6.3 Deep Fusion

Deep Fusion technique combines both early fusion and late fusion methods. It consists in fusing data or relevant features from the different modalities multiple times across the network. The network learns joint modalities with different features along the network depth. One approach that implements deep fusion is the Multi-View 3D Object Detection (MV3D) [58], which consists in a sensory-fusion framework that uses LiDAR point clouds and RGB images as input and predicts oriented 3D bounding boxes. First, 3D object proposals are generated from the Bird's Eye View (BEV) map, and then, multi-view features are deeply fused through region-based representation. Another approach is the Aggregate View Object Detection (AVOD) [59], which proposes a novel method of deep fusion. This proposal relies on feature extractors to generate feature maps from the Bird's Eye View (BEV) map and from the RGB image. A Region Proposal Network (RPN) is employed to generate non-oriented region proposals, taking as input the feature maps. Finally, the region proposals are fed into the detection network for the final classification.

3.7 Final Remarks

A general overview of the methodologies previously presented was conducted. The 3D object detection architectures using exclusively LiDAR input data were divided into point-based, grid-based and graph-based methods. A brief comparison between these algorithms that rely on one single data modality and some relevant multi-modal algorithms that combine LiDAR and image sensor data are presented. Table 3.2 shows the results reported by [44], obtained using the test set of the KITTI 3D detection benchmark. It is organized into different rows for the point-based, grid-based and graph-based methods and a last row for the multi-modal based architectures. The best performing results are highlighted.

As observed in the table 3.2, 3DSSD and PV-RCNN achieved the best performing results for all the three classes, except for the Pedestrians in Easy and Moderate difficulties, where IPOD outperforms these methods. This demonstrates that processing raw point clouds as well as the combination of 3D voxelization with point-based methods are suitable approaches for the 3D object detection task. Point-GNN also obtained good results, suggesting that the use of graphs to represent point clouds is an adequate strategy. Besides that, multi-modal architectures did not outperform some LiDAR based algorithms, although achieving notable results.

Regarding the multi-modal based methods, IPOD presents the best results for the pedestrian class. The pedestrians are one of the main focus when developing Autonomous Driving systems, once they are the most vulnerable element integrating the scene. Although vehicles, cars and cyclists, are important elements, the pedestrians imply a different focus, since they do not have a vehicle to protect them in case of an accident. In this regard, when selecting a methodology for object detection in Autonomous Driving applications, the system has to focus in the detection of pedestrians. PointPainting method also achieves good performance for the three classes, however, its low inference time makes impossible to run in a real-time environment.

Method	**Architecture**	**Modality**	**Speed**	**Cars**			**Pedestrians**			**Cyclists**		
			(fps)	E	M	H	E	M	H	E	M	H
3DSSD [48]	Single Shot	L	25.0	88.36	79.57	74.55	**54.64**	**44.27**	40.23	**82.48**	64.10	56.90
PointRCNN [49]	Region Proposal	L	10.0	86.96	75.64	70.70	47.98	39.37	36.01	74.96	58.82	52.53
VoxelNet [40]	Single Shot	L	2.0	77.47	65.11	57.73	39.48	33.69	31.51	61.22	48.36	44.37
SECOND [1]	Single Shot	L	26.3	83.34	72.55	65.82	48.96	38.78	34.91	71.33	52.08	45.83
PointPillars [51]	Single Shot	L	62.0	82.58	74.31	68.99	51.45	41.42	38.89	77.10	58.65	51.92
PV-RCNN [52]	Region Proposal	L	12.5	**90.25**	**81.43**	**76.82**	52.17	43.29	**40.29**	78.60	63.71	**57.65**
PointGNN [45]	-	L	1.7	88.33	79.47	72.29	51.92	43.77	40.14	78.60	63.48	57.08
MVX-Net [54]	Single Shot	L & I	16.7	84.99	71.95	64.88	-	-	-	-	-	-
F-PointNets [55]	Region Proposal	L & I	5.9	82.19	69.79	60.59	50.53	42.15	38.08	72.27	56.12	49.01
PointPainting [57]	Region Proposal	L & I	2.5	82.11	71.70	**77.08**	50.32	40.97	37.87	77.63	63.78	55.89
MV3D [58]	Region Proposal	L & I	2.8	74.97	63.63	54.00	-	-	-	-	-	-
AVOD [59]	Region Proposal	L & I	12.5	76.39	66.47	60.23	36.10	27.86	25.76	57.19	42.08	38.29
F-ConvNet [60]	Region Proposal	L & I	2.1	**87.36**	**73.39**	66.69	52.16	43.38	38.80	**81.98**	**65.07**	**56.54**
IPOD [61]	Region Proposal	L & I	5.0	80.30	73.04	68.73	**55.07**	**44.37**	**40.05**	71.99	52.23	46.50

Table 3.2: Comparison of the results obtained for 3D object detection on the KITTI test 3D detection benchmark, reported by [44]. The 3D bounding box IoU threshold is 0.7 for cars and 0.5 for pedestrians and cyclists. In the modality column, L refers to the LiDAR and I refers to the Image. Also, the letters 'E', 'M' and 'H' state for easy, moderate and hard difficulties, respectively. The results are presented in percentage (%).

Results for the MVX-Net method were only obtained for the car class, which achieves the second best result compared with the other multi-modal approaches. Moreover, this architecture achieves an inference rate of 16.7 fps [44], which is the fastest fusion method, leading to conclude that this could be an adequate approach for a real-time perception system. The trade-off between its performance, although only tested for the car class, and its inference time implies that this is the most suitable fusion strategy for object detection in the context of autonomous vehicles.

Chapter 4

Methodology

This chapter describes the methodology of the system. First, a brief contextualization regarding the methodology proposal is provided. Then, the framework implementation is introduced, detailing the architecture definition. The datasets used to train and validate the solution are described. Finally, the performed experiences are presented.

As explained in the previous chapter, an appropriate point cloud representation allows the generation of accurate range view images, which organizes the 3D points from the LiDAR to a 2D range-view image. However, the low resolution and texture makes the point clouds less precise for detection of smaller objects, such as pedestrians. These objects are clearly visible in 2D images, but difficult to detect and distinguish from other objects in point clouds representation. On the other hand, the image does not provide depth information, but yields fine-grained texture and colour information. In this sense, fusion architectures overcome the failures of each data modality, by combining information of LiDAR point clouds with image [57].

The top-performing state-of-the-art methods are based on single-modality data. Since the literature regarding fusion architectures has not been as deeply studied as the LiDAR based 3D object detectors, their performing results do not surpass the more recent single-modality architectures [5]. In this work, an in-depth comparison between the best methods for both single-modality and multi-modality object detection is depicted, considering the trade-off between performance and inference time. In this sense, the 3DSSD architecture is used as the benchmark results for LiDAR based object detector and the MVX-Net is taken as the best fusion approach. Therefore, this approach proposes to use both data modalities, by implementing a fusion architecture based on MVX-Net.

A literature survey was conducted, considering the most relevant detectors to integrate in the proposed solution. MVX-Net enables the mix and match of different object detection networks for both image and LiDAR modalities. In this regard, figure 4.1 presents the most relevant 2D and 3D detectors available to integrate this framework.

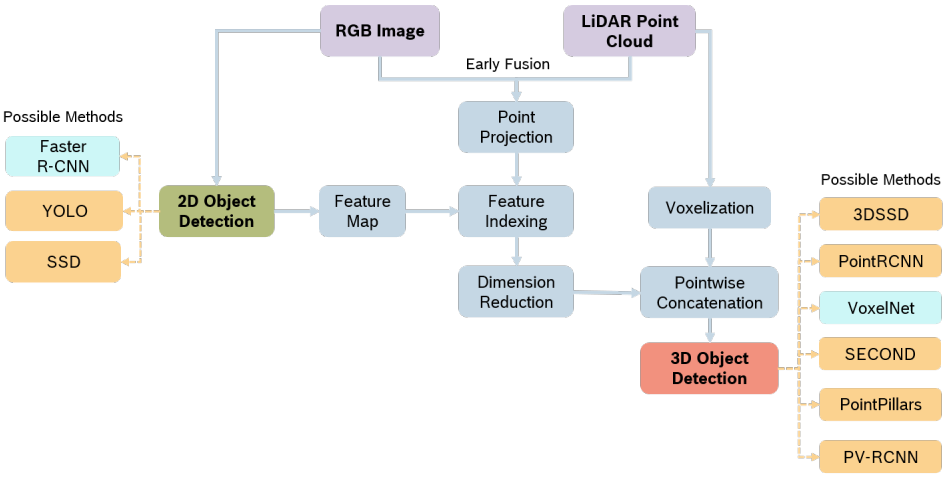


Figure 4.1: Overview of the MVX-Net architecture with the most relevant 2D and 3D object detectors available to integrate this approach. The networks used in the original MVX-Net proposal are highlighted, which consist in Faster R-CNN and VoxelNet.

An initial architecture was proposed, which implements SSD [33] for 2D detection and 3DSSD for 3D detection. This is a novel approach, since it would make use of both single-shot detectors in a fusion strategy, which was not explored in the state-of-the-art yet. Moreover, implementing a faster network to retain the image features, combined with a robust 3D solution to perform the final detection using the fused data would enable achieving a trade-off between performance and inference rate. Therefore, the first proposal was to implement SSD, as it is a single-shot based method and achieves fast inference speed. However, an SSD implementation which considers KITTI dataset for training is not provided in a publicly available repository and thus, an implementation was developed to train the model with this dataset. The first trained models performance did not achieve results comparable with the state-of-the-art, and therefore, the network implementation could need to be optimized. Since this does not addresses the scope of this dissertation and it would imply large amount of time focusing on the problem, it is not a viable solution.

Currently, 3DSSD is one of the top-performing methods in the state-of-the-art, which achieves remarkable results. On the other hand, MVX-Net extracts image features from the 2D object detector, which are fused with the point cloud to be passed as inputs to the 3D detector. In this sense, the 3D detector is the core component integrating this architecture. Having this into consideration and taking the 3DSSD as benchmark, this work will focus on the use of 3D detectors. MVX-Net is a voxel-based architecture, and thus, it performs the voxelization of the point cloud for further detection. As such, in order to implement 3DSSD on top of it, modifications in the architecture would be necessary. 3DSSD is a point-based method and thus, its straightforward integration with MVX-Net is not possible. Regarding the image modality, in this architecture the 2D detector is not as relevant as the 3D component. Nevertheless, better performing networks for 2D object detection can be found on the state-of-the-art and their implementation on MVX-Net architecture could improve future results. Moreover, in the future, it would be interesting to focus in this data modality to enable retain relevant features that could improve the fusion performance. Therefore,

a detailed study focusing in 2D detectors for a fusion architecture would need to be developed.

4.1 Framework Implementation

The image modality aims to complement the information obtained by the LiDAR point clouds, performing as an auxiliary module for the object detection task. For this reason, this framework makes use of a fast 2D object detector which does not need to be the top-performing method in the state-of-the-art. Instead, a fast running approach is used, which obtains good inference speed. Therefore, the Faster R-CNN [31] was considered as the 2D object detection approach.

On the other hand, the 3D object detector is required to be a voxel-based method and should rely on a more robust algorithm, ideally one of the top performing state-of-the-art methods. Therefore, the PointPillars [51] architecture was implemented. This network was proposed in 2019, right after the publication of MVX-Net, and thus, the inclusion of this 3D object detector in the MVX-Net fusion architecture was not explored yet. Additionally, another architecture was proposed, consisting in integrating SECOND network as the 3D detector. Thus, under this work, three of the more relevant voxel-based methods are implemented on the top of MVX-Net fusion approach. Figure 4.2 shows an overview of the proposed framework. The Faster R-CNN and the VoxelNet are the detectors implemented in the original MVX-Net proposal, as mentioned previously.

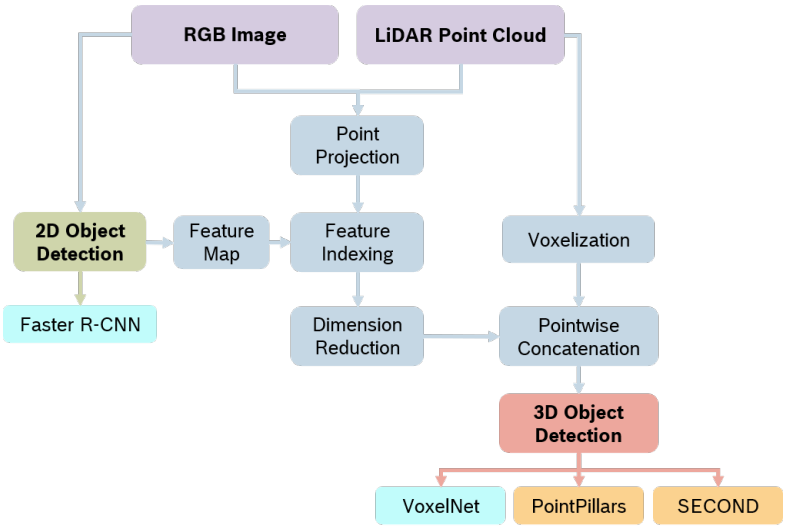


Figure 4.2: Overview of the proposed implementation of MVX-Net architecture with VoxelNet, PointFusion and SECOND detectors. The Faster R-CNN and the Voxel-Net are the detectors proposed in the paper [54], which are further referred as the baseline architecture. Additionally, the PointPillars and the SECOND networks are implemented in this framework.

4.1.1 MVX-Net Architecture

MVX-Net makes use of multi-modal data to perform object detection in a 3D scene. Under this work, the PointFusion strategy is considered, as it achieves better results than the VoxelFusion. First, the image is passed to a 2D object detector that extracts a high level feature map. Each

3D point is then projected onto the image, appending the point with the feature of the projected location index. The features extracted are reduced in dimension by implementing a set of fully-connected layers. Then, these features are concatenated with the point feature and passed to a set of Voxel Feature Encoding (VFE) layers. Figure 4.3 shows an overview of this architecture.

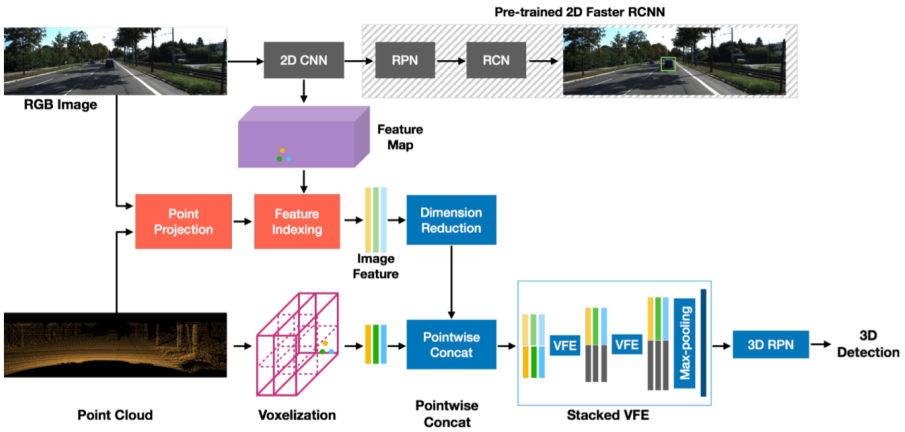


Figure 4.3: Overview of the MVX-Net PointFusion architecture, extracted from [54].

Table 4.1 shows the results obtained for MVX-Net on the KITTI validation set. The VoxelNet architecture benefits from the use of image based data, outperforming the baseline network. Moreover, MVX-Net with PointFusion reported benchmark results in the KITTI test set, achieving top-performing results when compared with the other fusion methodologies [54].

Method	AP_{BEV}(IoU=0.7)			AP_{3D} (IoU=0.7)		
	Easy	Moderate	Hard	Easy	Moderate	Hard
VoxelNet	87.6	83.7	78.4	79.5	65.7	64.6
MVX-Net (PF)	89.5	84.9	79.0	85.5	73.3	67.4
Delta	**+ 1.9**	**+ 1.2**	**+ 0.6**	**+ 6.0**	**+ 5.1**	**+ 2.8**

Table 4.1: Results of MVX-Net on the KITTI validation set, using mean Average Precision (in %) with IoU=0.7, adapted from [54]. MVX-Net (PF) stands for PointFusion strategy.

4.2 Data Processing

To train and test the proposed framework, two datasets were used. The KITTI 3D object detection, which is the most widely used dataset in the literature for 3D object detection in Autonomous Driving, and also, nuScenes, a large and challenging dataset, which was not as explored as KITTI.

4.2.1 KITTI 3D Object Detection

The KITTI 3D Object Detection was used to train and validate the framework. The state-of-the-art methods typically present their benchmark results using this dataset and, therefore, the employment of KITTI becomes an obvious choice to perform a direct comparison with the published results. However, the KITTI test set is used exclusively for publication results and the bureaucratic constrains to access it are unjustified for the purpose of this work. As such, the evaluation of the architecture is performed with the validation set. The dataset was split into train and validation sets, using a standard division available in the literature. The 7481 training examples were divided into 3712 samples for the training set and 3769 samples for the validation set, as presented in Table 4.2.

Set	Number of Samples
Train	3712
Validation	3769

Table 4.2: Split of KITTI 3D Object Detection dataset.

Figure 4.4 presents the classes distribution in the KITTI dataset. Considering the three classes used: car, pedestrian and cyclist, the class cyclist is the most underrepresented, in detriment of cars, which is largely the most overrepresented.

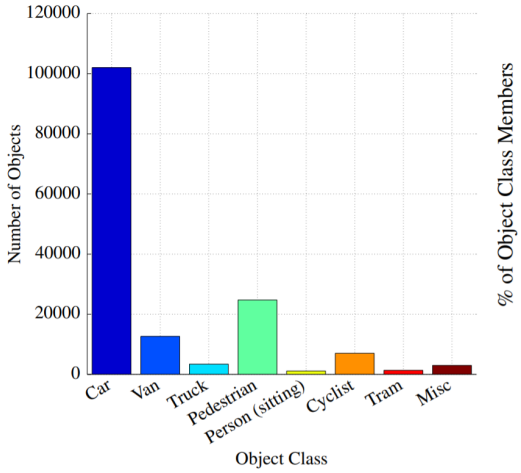


Figure 4.4: Class distribution of the KITTI dataset, extracted from [37].

4.2.2 nuScenes

In order to train and validate the proposed architecture in a large and different scenario, the LiDAR and camera modalities of nuScenes dataset were used. This dataset consists of 1000 scenes of 20 second length selected from the raw data collected, which comprehends diverse

traffic situations and behaviours. The objects were manually labeled with accurate 3D bounding boxes at 2Hz over the entire dataset. Therefore, it is composed of 700 for train, 150 for validation and 150 for test, as depicted in Table 4.3.

Set	**Number of Scenes**
Train	700
Validation	150

Table 4.3: Organization of the nuScenes dataset.

4.3 Experimental Setup

The architectures for 3DSSD, PointPillars and MVX-Net were the first to be trained, serving as baseline for further results comparison. As a starting point for the implementation, the repository from open-mmlab, MMDetection3D [62] was used for the fusion and LiDAR based architectures. However, it does not provide a proper implementation of 3DSSD as described in [48] and thus, OpenPCDet [63] repository was used for this purpose.

Data augmentation techniques were implemented to train 3DSSD, MVX-Net and PointPillars, which rely on artificially creating new training data from the existing training samples. This strategy is widely used to increase the diversity of data, which leads to a better generalization of the model. The transformations are applied to the raw point clouds. They consist in random world flips along the x-axis, random world rotations around the z-axis with a random angle sampled from $[-\frac{\pi}{4}, \frac{\pi}{4}]$ radians, a random world scaling with a random scaling factor sampled from $[0.95, 1.05]$ and a global translation with (x, y, z) drawn from a $[0, 0.2]$ interval. To the input images, no type of augmentation is applied, other than simple resize and normalization procedures. Both 3DSSD and PointPillars apply a sampling strategy, the ground truth augmentation, proposed in [1], by randomly including different ground truth objects into the point clouds during the training. Additionally, experiments were performed applying this sampling strategy with different configuration parameters to the proposed framework: MVX-Net with PointPillars.

3DSSD was trained end-to-end using the ADAM optimizer, with an initial learning rate of 0.001. The step schedule for the learning rate started at 0.01 and decayed by a factor of 10 at 40 epochs. The model was trained with a batch of size 16 for 120 epochs. PointPillars was trained end-to-end using the ADAM optimizer, with a one cycle strategy, defining the initial learning rate at 0.0018. The model was trained using a batch of size 6 for 160 epochs. MVX-Net with VoxelNet (baseline) was trained end-to-end using the ADAM optimizer with a step schedule for the learning rate starting at 0.003. A weight decay of 0.01 was used to reduce the learning rate during the

training, setting a minimum value of $1e^{-5}$. A batch of size 2 was used during 120 epochs. MVX-Net with PointPillars was trained end-to-end using the ADAM optimizer with one cycle strategy, setting the initial learning rate at 0.0018. A batch size of 2 was used during 120 epochs.

Moreover, an additional architecture was implemented under the fusion approach. It consisted in the use of SECOND as the 3D detector in MVX-Net. This approach proposes novel improvements to VoxelNet, by implementing Sparse Convolutions in the middle layers. Therefore, the computationally heavy 3D Convolutions are replaced by sparse convolutional layers. MVX-Net with SECOND was trained end-to-end with the same setup as the original MVX-Net architecture. Table 4.4 summarizes the hyperparameters set for the training of each architecture.

Hyperparameters	**3DSSD**	**PP**	**MVX-Net Baseline**	**MVX-Net + PP**	**MVX-Net + S**
Epochs	120	160	120	120	120
Batch Size	16	6	2	2	2
Optimizer	ADAM	ADAM	ADAM	ADAM	ADAM
Schedule	Step	One Cycle	Step	One Cycle	Step
Initial Learning Rate	0.001	0.0018	0.003	0.0018	0.003

Table 4.4: Hyperparameters setup used for train each of the architectures. PP refers to PointPillars network; MVX-Net Baseline refers to the original architecture, which uses VoxelNet as the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars; MVX-Net + S refers to the MVX-Net with SECOND.

4.3.1 Hardware

An infrastructure was provided to perform the training of the proposed framework, as well as the state-of-the-art architectures used for comparison and analysis. The hardware resources consisted in a Xeon Gold 6150 CPU and a Tesla V100-SXM2 32GB GPU.

4.3.2 Model Generalization

In order to evaluate the models performance given a new set of data, additional experiments were conducted. 3DSSD and MVX-Net with PointPillars were trained using the nuScenes dataset. 3DSSD was trained end-to-end using the ADAM optimizer, with a one cycle strategy, defining the initial learning rate at 0.01. The model was trained with a batch of size 16 for 120 epochs. MVX-Net with PointPillars was trained end-to-end using the ADAM optimizer with a step schedule for the learning rate starting at 0.001. A weight decay of 0.01 was used to reduce the learning rate during the training. A batch of size 4 was used during 120 epochs.

Chapter 5

Results and Analysis

This section presents the results obtained in the experiments, and are assessed regarding performance and inference rate. A benchmark comparison between the proposed architectures is developed, as well as a trade-off definition of the best approach in this context. Additionally, an in-depth analysis of the multi-modality performance is assessed, considering the relevant aspects of the detection.

The initial MVX-Net with PointPillars uses the data augmentation transformations described in the Chapter 4, without any sampling strategy employed to increase the amount of objects in the scene. However, in order to optimize the model and achieve higher results, a study was conducted by performing a set of experiments regarding sample strategy. The authors in [64] developed an extensive study focused on data augmentation in PointPillars. In particular, several configurations are used employing a sample strategy, defined by two different parameters: filter by points and oversampling. Filter by points consists in excluding the annotations that contain less than a certain amount of points. Oversampling is the operation that samples the additional objects into the current scene. The annotations and their corresponding points in the train set are saved, by iterating once through all the annotations. During the training, the model checks if the additional objects do not collide with any of the original objects in the scene. For that, a value is assigned corresponding to the number of annotations to be oversampled. This paper only carries out experiments to assess the impact of data augmentation for the class Car, which is the most overrepresented class in the KITTI dataset. Under this work, a less detailed study is developed, applied for all the three classes in the dataset, particularly the cyclists class, which is the most underrepresented class in the KITTI dataset.

Table 5.1 presents the results of the experiments, as well as the MVX-Net with PointPillars baseline. The configuration parameters were the same for all the difficulties, as defined in table 3.1. Model SS1 applies the same values to the parameters for sampling pedestrians and cyclists. The results show that the cyclists are highly improved, however, the performance on pedestrians decreases. In Model SS2 the same sampling strategy is applied for cyclists, but not employing any sampling to the other objects. This strategy shows a remarkable improvement for the pedestrians class, by only oversampling the cyclists in the scene. An in-depth analysis of this behaviour is

further developed in this section. Model SS3 applies more oversampling to all the three object classes and decreases the number of minimal points required for each sample object. The results show improvements in both car and cyclist classes, however lower results are obtained in the pedestrian class. Finally, Model SS4 applies the values for minimal points and oversampling that achieved the best results in the previous experiments, namely in Model SS3. However, it does not employ any sampling to the pedestrians since it has been shown that oversampling this class deteriorates the performance. Overall, the results show significant improvements in pedestrians and cyclists classes.

Model	**Sampling Strategy**	**Car**			**Pedestrian**			**Cyclist**		
		E	M	H	E	M	H	E	M	H
MVX-Net + PP	Results	90.18	87.42	86.22	67.29	61.34	57.92	78.56	56.70	52.97
Model SS1	Filter by Points	5			10			10		
	Oversampling	15			10			10		
	Results	89.78	87.21	84.96	62.61	56.27	51.65	83.01	67.25	63.49
Model SS2	Filter by Points	0			0			10		
	Oversampling	0			0			10		
	Results	88.68	86.39	79.29	**69.06**	**62.68**	**60.32**	79.43	61.98	57.99
Model SS3	Filter by Points	5			5			5		
	Oversampling	20			20			15		
	Results	90.82	90.02	89.41	61.84	55.10	50.73	83.10	66.99	63.59
Model SS4	Filter by Points	5			0			5		
	Oversampling	20			0			15		
	Results	89.22	86.68	84.59	66.67	60.40	58.24	83.61	64.78	61.10

Table 5.1: Comparison of the results obtained on the KITTI validation set, applying the sampling strategy proposed in [1]. The values assigned to each configuration parameter are detailed, which are the same for all the difficulties in each class. The AP_{BEV} with IoU threshold of 0.7 for cars and 0.5 for pedestrians and cyclists was used for evaluation, presented in percentage (%).

From the results presented in table 5.1, the models SS2 and SS4 achieve the top-performing results. Table 5.2 shows the results obtained for each model architecture. Additionally, the more relevant experiments conducted applying oversampling are also presented. The models were evaluated using the KITTI validation set. The AP_{BEV} metric is used with IoU thresholds of 0.7 for cars and 0.5 for pedestrians and cyclists.

Method	Cars			Pedestrians			Cyclists		
	E	M	H	E	M	H	E	M	H
3DSSD	90.31	88.15	86.42	62.50	56.37	53.92	88.95	72.39	67.79
PointPillars	90.04	86.99	79.74	57.33	52.60	47.33	81.46	62.90	60.96
MVX-Net Baseline	89.67	80.03	71.26	66.24	58.67	55.92	53.30	41.77	40.72
MVX-Net + PP	90.18	87.42	86.22	67.29	61.34	57.92	78.56	56.70	52.97
Mvx-Net + PP (SS2)	**88.68**	**86.39**	**79.29**	**69.06**	**62.68**	**60.32**	**79.43**	**61.98**	**57.99**
Mvx-Net + PP (SS4)	89.22	86.68	84.59	66.67	60.40	58.24	83.61	64.78	61.10
MVX-Net + S	89.42	86.86	85.07	70.91	64.04	61.89	74.92	60.26	55.94

Table 5.2: Comparison of the results obtained on the KITTI validation set. The AP_{BEV} IoU threshold is 0.7 for cars and 0.5 for pedestrians and cyclists. The results are presented in percentage (%). MVX-Net baseline refers to the original architecture, which uses VoxelNet as the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars as the 3D detector; MVX-Net + S refers to the MVX-Net with SECOND network. The SS2 and SS4 refers to the Sampling Strategy applied to the data.

3DSSD achieves top-performing results for both car and cyclist classes, however, the performance in the pedestrian class is lower. MVX-Net Baseline obtained the worst performance in the cyclist class, but higher in the pedestrian class when compared with 3DSSD. Cyclists are the less represented class in the KITTI dataset, and thus, as the baseline architecture does not apply any sampling strategy, the performance degrades. Moreover, the results demonstrate that PointPillars benefits from multi-modality, achieving better results for the three classes when implemented with MVX-Net. In the fusion approaches of MVX-Net with PointPillars, as well as MVX-Net with SECOND, the results achieved for the pedestrian class have minimal differences between the Moderate to Hard difficulties, varying less than 5%. This can be justified by the improvement that fusion strategies have regarding small objects and at longer distances.

MVX-Net + PP (SS1) results improved the performance for the cyclist class by a large margin in all difficulties, however decreasing the performance in pedestrians. Although the results obtained by MVX-Net + S are comparable with MVX-Net + PP (SS2), the model shows decreased performance for the cyclist class. Overall, the MVX-Net + PP (SS2) achieves the best results, obtaining top-performing results in the pedestrian class.

Looking now at the loss evolution for these models, figure 5.1 shows the loss when training 3DSSD during the 120 epochs. The model achieves a loss of 1.46 when the training process finishes. Figure 5.2 presents the loss progression while training of MVX-Net with PointPillars

(SS2) and MVX-Net with SECOND during 120 epochs. MVX-NET + PP (SS2) finishes the training with a loss value of 0.30 and MVX-Net + S with a loss of 0.41. The loss obtained for both are relatively similar, being however clearly lower when comparing with 3DSSD.

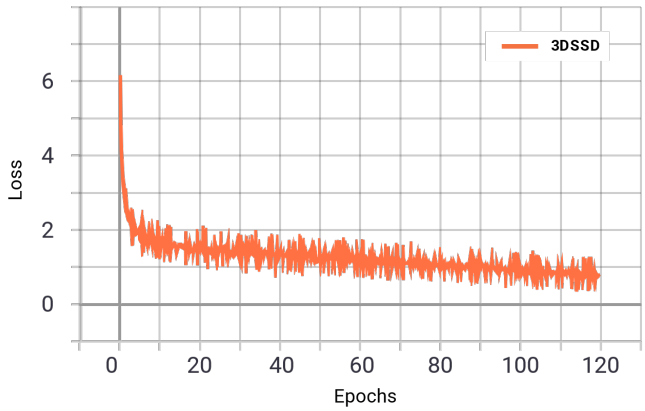


Figure 5.1: Loss obtained when training the 3DSSD during 120 epochs.

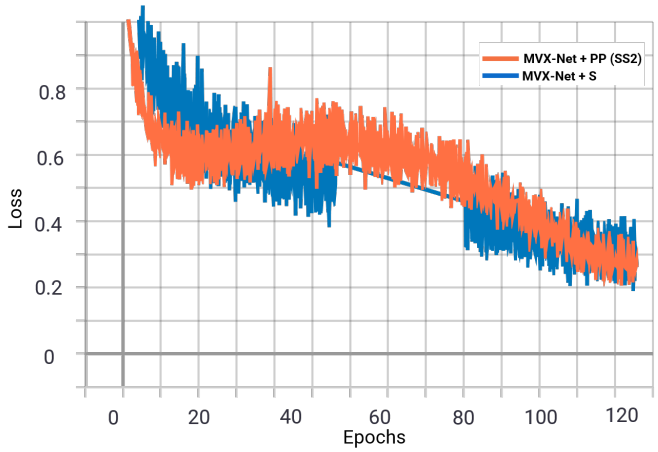


Figure 5.2: Loss obtained when training the MXV-Net + PP and MVX-Net + S during 120 epochs. MVX-Net + PP (SS2) refers to the fusion architecture implemented with PointPillars applying a sample strategy and MVX-Net + S refers to the fusion architecture implemented with SECOND.

5.1 Inference Rate

As previous explained, to develop an effective application for object detection in the context of Autonomous Driving, inference rate is a core feature for running in real-time. Therefore, the inference time was computed for each architecture. Table 5.3 presents the average inference (in Hertz) of the 10 runs computed.

Model	Inference (Hz)
3DSSD	3.1
PointPillars	31.4
MVXNet Baseline	2.4
MVX-Net + PP	15.2
MVXNet + SECOND	11.4

Table 5.3: Comparison of the inference rate results obtained with each architecture. MVX-Net baseline refers to the original architecture, which uses VoxelNet the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars as the 3D detector; MVX-Net + S refers to the MVX-Net with SECOND network.

The inference rate reported in [48] for 3DSSD could not be reproduced, as preliminary investigation revealed a run-time of only 3.10 Hz. An in-depth study would be necessary to improve the architecture run-time, which was not conducted, as it is not within the scope of this dissertation.

MVX-Net with PointPillars shows to be the fastest approach, running at 15.2 Hz. MVX-Net with SECOND decreases 3.8 Hz when compared with PointPillars on the top of MVX-Net. In a real-world context, the developed product or application has to accomplish a set of requirements depending on the target hardware used. The resources to run the model would be much lower than the ones used to perform these tests, and thus, the model inference would automatically drop. Having this into consideration, the results obtained for MVX-Net with PointPillars compared with MVX-Net with SECOND present a relevant difference for real-time inference. Therefore, PointPillars proves to improve MVX-Net architecture in terms of inference, showing to be the most suitable approach for real-time applications.

Figure 5.3 provides a visual aid of the trade-off between inference rate and performance. The horizontal axis represents the inference rate (in Hz) and the vertical axis represents the performance (in percentage) for the moderate difficulty of pedestrians class, using the metric AP_{BEV}. The graphic shows the results obtained with each of the models.

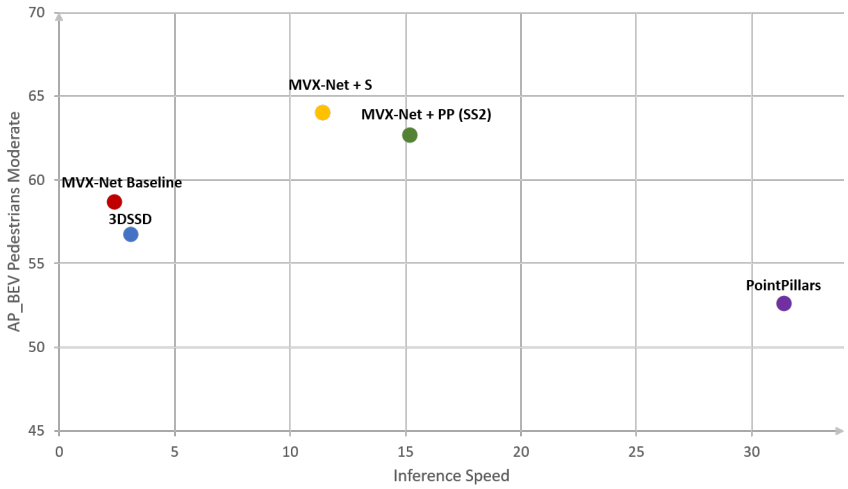


Figure 5.3: Trade-off between performance and inference rate. The horizontal axis presents the inference rate (in Hz) and the vertical axis presents the performance (in %) in moderate difficulty of pedestrians class, using the metric AP_{BEV}. MVX-Net Baseline refers to MVX-Net with VoxelNet; MVX-Net + PP (SS2) refers to MVX-Net with PointPillars applying a sampling strategy; MVX-Net + S refers to MVX-Net with SECOND.

By observing the graphic, it is possible to conclude that MVX-Net with PointPillars and MVX-Net with SECOND achieve the best results, reaching a balance between performance and inference rate. MVX-Net with SECOND shows a slightly higher performance, however, the gain in the inference rate with PointPillars on the top of MVX-Net compensates. Overall, MVX-Net with PointPillars is the best approach, and therefore it will be used for the next experiments.

5.2 Model Generalization

In addition to the results obtained with the KITTI dataset, more experiments were conducted with nuScenes. This is a challenging dataset, which has not been as widely explored in the literature as KITTI. In this regard, the use of this dataset aims to assess the capacity of the architectures to generalize given new information. Moreover, as KITTI is the benchmark dataset to evaluate state-of-the-art methodologies, the architectures can be biased regarding this data and therefore, they may not be suitable for real-world applications, having as input new and unpredictable information. Table 5.4 shows the results obtained with nuScenes.

Method	NDS (%)	mAP (%)	mATE (m)	mASE (1-iou)	mAOE (rad)	mAVE (m/s)	mAAE (1-acc)
MVX-Net + PP	36.46	19.32	0.5085	0.3069	0.7250	0.5627	0.2165
3DSSD	24.97	13.37	0.5550	0.4043	0.5991	1.4815	0.6131

Table 5.4: Comparison of the results obtained with the nuScenes validation set. The metrics used to evaluate consist in the NDS (NuScenes Detection Score), mAP and five error metrics regarding translation, scale, orientation, velocity and attribute, corresponding to mATE, mASE, mAOE, mAVE and mAAE, respectively. MVX-Net + PP refers to the MVX-Net with PointPillars.

It is important to note that the models were not optimized for this dataset, which explains the lower results obtained. By comparing the values for both *mAP* and *NDS*, it is clear that 3DSSD performs worst with nuScenes. In [48] the authors propose the use of a specific backbone in 3DSSD with nuScenes dataset, different from the original backbone used with KITTI. This backbone focuses in the specific dataset. Thus, the architecture is not scalable to different input resolutions and therefore, given a more challenging dataset such as nuScenes, the performance results are lower than expected. In this sense, this network translates an unrealistic behaviour for a real-world context and would need architecture re-adjustments to fit the new target data.

5.3 Results Analysis

An in-depth analysis of the chosen model trained and evaluated on the KITTI dataset is conducted. The classification performance for each object integrating the scenes is assessed, computing the confusion matrix. Figure 5.9 shows the confusion matrix obtained with MVX-Net with PointPillars (SS2). The columns represent the true classes and the rows represent the predicted classes. The TP values for each class are shown in green.

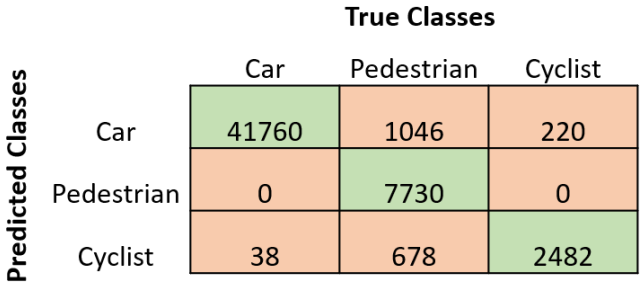


Figure 5.4: Confusion matrix obtained with MVX-Net with PointPillars (SS2). The columns refer to the true classes and the rows to the predicted classes. The green cells are the TP values.

The model shows to be effective in the pedestrians classification, not misclassifying any other object as pedestrian. This behaviour is translated by the $FP_{pedestrian} = 0$, which are remarkable results. The detection of pedestrians is a very delicate and challenging task for autonomous vehicles and therefore, not confusing other objects with them is a relevant achievement. On the other hand, the model misclassifies some pedestrians as cyclists. As shown in table 5.1, applying a sample strategy to the cyclists, improves the performance in the pedestrians class. This behaviour is observable by the amount of pedestrians confused with cyclists. Including more cyclist ground-truths in the scene enables the model to learn from more sample objects regarding both cyclists and pedestrians. Since a cyclist is composed of a person and a bicycle, the misclassification of pedestrians as cyclists is an expected behaviour. Besides that, some cyclists are confused with cars, however none of them are misclassified as pedestrians. The class cars achieves top-performing classification results compared with the other classes, as only approximately 9% of cars are misclassified as cyclists. Therefore, the $FN_{pedestrian} = 1724$ translates the difficulty to correctly classify pedestrians, which can be a critical behaviour in a real-world context.

Figure 5.5 shows the sample image 2206, which contains four labeled pedestrians in the scene. Two of them are misclassifed by the model as cyclists, which are presented within the selected area, zoomed in the image. As observed, these pedestrians are overlapped by the bicycles in the background and thus, their correct classification is difficult, being therefore confused by the model with cyclists.

Figure 5.5: Example of sample image 2206. The two pedestrians within the selected area are misclassified as cyclists.

Another example is presented in figure 5.6, which corresponds to sample 4954. The scene includes six labeled pedestrians, two of them were misclassified as cyclists by the model. Zooming into these objects in the image, it is possible to visualize that the person is carrying a bicycle. Although the person is not on the top of the bicycle, it is difficult for the model to make this distinction and therefore, classifies these pedestrians as cyclists.

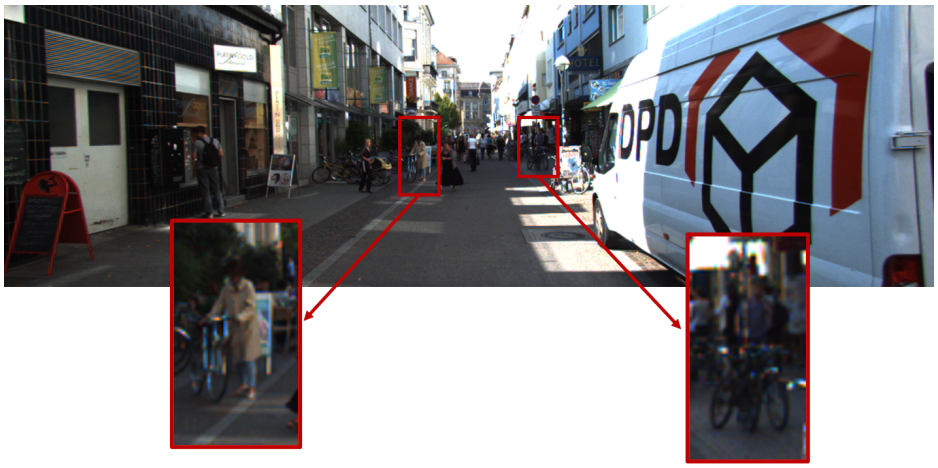


Figure 5.6: Example of sample image 4954, in which two pedestrians are misclassified as cyclists. Presented within the selected area, these objects are zoomed in for better visualization.

In figure 5.7 sample 4928 is shown. This sample is composed of several labeled pedestrians, in which two were misclassified as cyclists. The image provides a zoom into these objects, where it is possible to observe their occlusion by the parked bicycles. As such, the model assumes this composition of a person obstructed by a bicycle as a cyclist, misclassifying these sample objects as cyclists.

Figure 5.7: Example of sample image 4928, which includes a set of labeled pedestrians. The pedestrians within the selected area are misclassified as cyclists, zoomed in for better visualization.

The model confusion between pedestrians and cyclists reveals a lack of distinction between a person on a bicycle and a person with a bicycle. This is an ambiguous behaviour, since a person with a bicycle can also be seen as a cyclist int the rules of the road. KITTI labels only assume a pedestrian as a person when walking or standing and a cyclist as a person on top of the bicycle. Moreover, the composition of a cyclist being a set of both a person and a bicycle is a problem solvable with instance segmentation, where a different mask is assigned to the person and the bicycle, therefore resulting in two separated segmentations. However, by applying combination

rules to the masks, the segmentation of a person on the top of a bicycle can be classified as a cyclist.

Figure 5.8 shows sample image 2206, which contains one labeled pedestrian and two labeled cars in the scene. The pedestrian is misclassifed by the model as a car, which is presented within the selected area, zoomed in the image. It is clear that the object is partially occluded by the wall, and only the upper body is visible. In this sense, as there is a car on the background, within a small distance to the person, the pedestrian can be interpreted as an occupant of the vehicle and therefore, the model misclassifies it as a car.

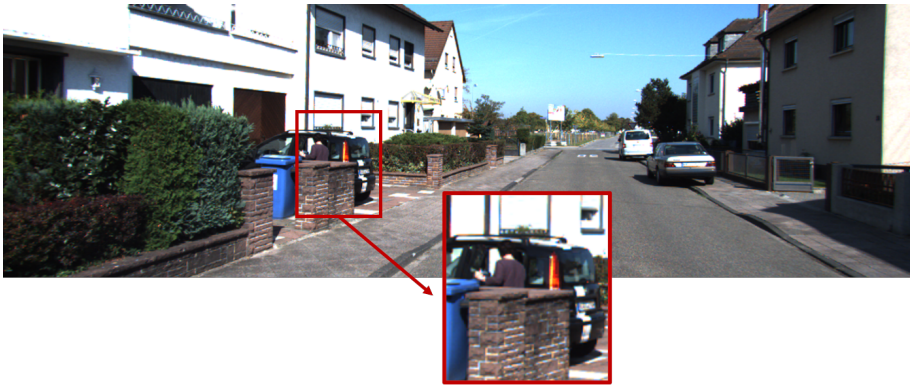


Figure 5.8: Example of sample image 63, which includes a set of labeled pedestrians. The pedestrian within the selected area is misclassified as car, zoomed in for better visualization.

Additionally, the baseline model of MVX-Net with PointPillars without any sampling strategy is evaluated. An analysis regarding the employment of oversampling is conducted, by comparing the classification performance of both models, with and without oversampling. In this sense, the confusion matrix for MVX-Net + PP was obtained, as shown in figure 5.9.

		True Classes		
		Car	Pedestrian	Cyclist
Predicted Classes	Car	41272	1890	660
	Pedestrian	0	5382	0
	Cyclist	32	248	1550

Figure 5.9: Confusion matrix obtained with MVX-Net with PointPillars. The columns refer to the true classes and the rows to the predicted classes. The green cells are the TP values.

As in the model applying sampling, this approach shows to accurately predict pedestrians, in the sense that no object is misclassified as pedestrian. Therefore, $FP_{pedestrian} = 0$, which reveals certainty of the model to classify pedestrians. For the class car, the model achieves high performance, quite similar to the model with oversampling. On the other hand, the cyclists are highly

confused with cars, much more than in the model with oversampling. This behaviour is related to the lack of cyclists samples in the KITTI dataset, and thus, by oversampling this class, the model can learn from more samples, as well as combining different cyclists in the same scene, turning into a more challenging scenario. On the opposite, the model without sampling only learns from a limited amount of cyclist samples, which is the most underrepresented class. Moreover, this model shows to confuse more pedestrians with cars and less with cyclists. This behaviour is not realistic since the confusion of a pedestrian with a cyclist is more obvious and justifiable, as a cyclist is a person on a bicycle, while a person and a car do not have anything in common.

An example corresponding to sample 2091 is presented in figure 5.10. Only one pedestrian integrates this scene and it is misclassified by the model as a car. This confusion is related with the car behind the person, however, it is visible that the pedestrian is distanced from the car and thus, this behaviour reveals the lower performance this model has to classify pedestrians.

Figure 5.10: Example of sample image 2091, which includes one labeled pedestrian in the scene. The pedestrians within the selected area is misclassified as car, zoomed in for better visualization.

Figure 5.11 shows the sample image 7080. There is one labeled pedestrian integrating this scene, which was misclassified as cyclist. By zooming in the pedestrian, it is possible to observe that there is no similarity of this object with a cyclist, other than being composed of a person. In this sense, the model classification for this sample is completely wrong.

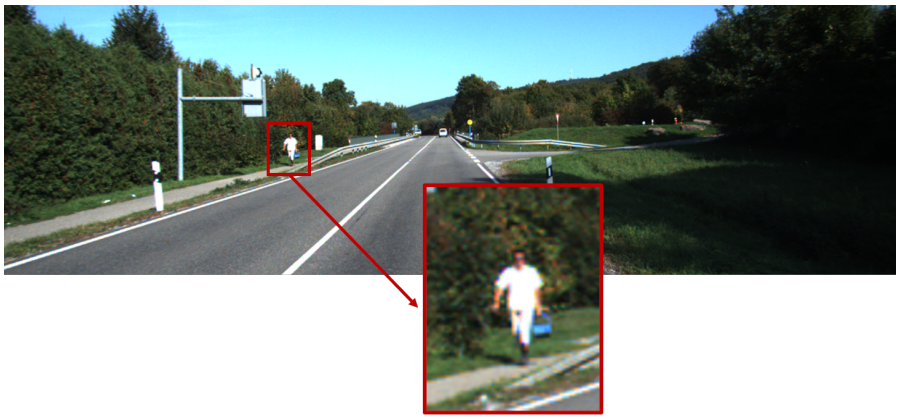


Figure 5.11: Example of sample image 7080. In this scene, there is one labeled pedestrian, shown within the selected area. This pedestrian is misclassified as cyclist.

In figure 5.12 the sample 6785 is presented. This sample is composed of three pedestrians and a cyclist. This cyclist is shown within the selected area in the image. By zooming into this area, it is possible to observe that the three pedestrians and the cyclist are partially occluded by each other, however, the cyclist is the most visible. The model misclassifies it as car, which in this case is hard to explain, since there is no car behind or near the cyclist. In this sense, it would be justifiable that the model confuses the cyclist with a pedestrian, but not with a car.

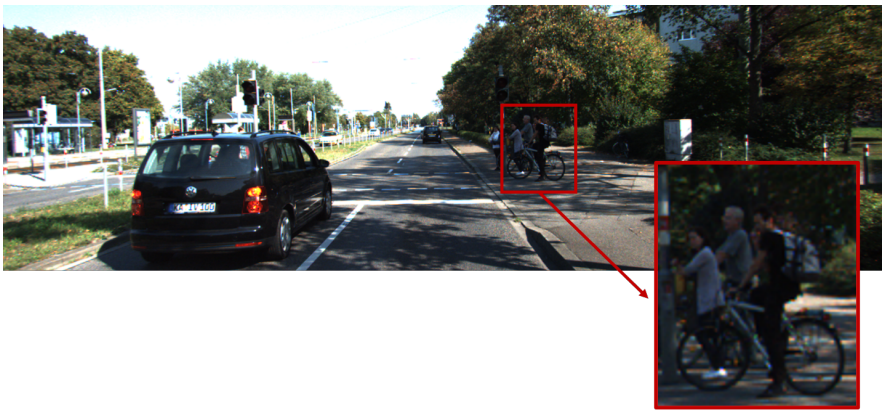


Figure 5.12: Example of sample image 6785. In this scene, there is one labeled pedestrian, shown within the selected area. This pedestrian is misclassified as car.

5.4 Final Remarks

PointPillars proved to benefit from the integration of image-based data. MVX-Net enables to extract relevant image features from the 2D detector applied. This architecture allows to effectively process image and point cloud information, allying the advantages of image with LiDAR modalities in an early fusion strategy. The results obtained with PointPillars on top of MVX-Net show the improvement to both performance and inference rate, when compared with the baseline. On the other hand 3DSSD, stated as one of the top-performing state-of-the-art architectures,

presented some relevant weaknesses when assessed. The network achieves low inference rate, which is not comparable with the other methodologies tested. Although this implementation can be optimized to improve the inference rate, the results lead to questioning whether the network architecture can be that much improved. Moreover, 3DSSD architecture appears to not be scalable to new input data. The experiments performed with nuScenes dataset obtained lower results when compared with the results obtained with PointPillars on the top of MVX-Net. To overcome this behaviour, the authors propose the implementation of a new backbone to use with nuScenes dataset. In this sense, 3DSSD architecture is unable to generalize for this data, which translates into an unrealistic scenario, since real-world systems are always receiving new input data and thus, using this network, the behaviour would be unpredictable. However, more experiments would be necessary to effectively assess this method, by using a new and more challenging dataset.

Additionally, experiments were conducted applying a sampling strategy. This technique proved to largely improve the performance in cyclists, by adding these objects to the scenes and therefore oversampling this class, the most underrepresented in KITTI dataset. Moreover, by oversampling the class cyclists, the performance in pedestrians is improved, which is related with the composition of a cyclist as a person and a bicycle, and therefore, increasing the amount of cyclists in the scene automatically increases the amount of people. On the other hand, this sampling also leads to more confusion between pedestrians and cyclists, translating in more pedestrians being misclassified as cyclists, when compared with the model without any sampling.

Overall, the MVX-Net with PointPillars applying a sampling strategy (SS2) shows to be the best approach, achieving good results for both performance and inference rate. Through the analysis of the confusion matrix, it is clear that this model sometimes confuses cyclists with pedestrians, which is an expected behaviour, since a cyclist also includes a person. In this sense, the misclassification of a pedestrian as a cyclist can be ambiguous. As previous depicted in some sample data, KITTI labels a cyclist as a person on top of a bicycle and therefore, if the person is walking alongside a bicycle, it is labeled as pedestrian. As such, this behaviour is not critical, since some rules of the road apply the same way to pedestrians and cyclists. However, to accurately distinguish a person, a bicycle and a cyclist, semantic rules could be employed. By segmenting the person and the cyclist separately, a rule could be defined to perceive the combination of a person on a bicycle as a cyclist.

Chapter 6

Conclusions and Future Work

The advances in Artificial Intelligence and consequently, the emergence of Deep Learning technologies led to an increased interest by the scientific community in Autonomous Driving. Self-driving cars need to accurately perceive the surrounding environment and therefore understand and distinguish the diverse elements integrating the scene. In this sense, perception systems need to be developed, which take the input data from the integrated sensors in the vehicle and which can inference and reason to perform several tasks. In particular, 3D object detection can be based on two types of data: image and LiDAR. LiDAR-based methods were more explored in the literature in detriment of multi-modal based methods. Therefore, they achieve top-performing results when compared with fusion architectures. However, by analysing the performance in smaller objects, such as pedestrians, multi-modal based approaches outperform LiDAR-based methods. Image data can retrieve texture information, as well as high resolution data, which enables the detection of smaller objects at higher distances. Having this into consideration, this work centers in the study of single and multi-modal architectures, by comparing two top-performing methods in the state-of-the-art. Moreover, a fusion based architecture is proposed and developed, and its performance is evaluated on two benchmark datasets.

In section 2, the relevant concepts for object detection were explained. The approaches to develop systems able to detect and classify objects in 2D images were described, as well as their evaluation metrics for the two tasks integrating object detection: regression and classification. Section 3 presents the main concepts to understand and interpret point clouds. The benchmark datasets in the context of Autonomous Driving are introduced, as well as the 3D evaluation metrics. Moreover, in section 4 a study is conducted regarding 3D object detection state-of-the-art proposals, drawing an overview of the most relevant architectures.

Section 5 covers the results obtained with the proposed framework, establishing a comparison with the top-performing LiDAR-based architecture 3DSSD. Several experiments were conducted regarding data augmentation, by employing a sampling strategy proposed in [1]. The results show that applying new ground-truth objects in the class cyclist leads to improvements for both cyclists and pedestrians. This behaviour can be justified by the underrepresentation of cyclists in the KITTI dataset. An analysis of the generalization ability was conducted, by training the models with the

challenging nuScenes dataset. 3DSSD results revealed that the architecture is not scalable to different input resolutions. In this regard, the authors propose a different backbone architecture specific for nuScenes dataset, which focuses in this particular data. Therefore, the architecture would need to be adjusted to fit the new target data. On the other hand, the fusion strategy is shown to be a suitable approach for real-world applications, obtaining promising results regarding the trade-off between performance and inference speed.

6.1 Future Work

In terms of future work, a more extensive study regarding data generalization should be conducted. The experiments performed using nuScenes dataset showed a decrease in 3DSSD performance, revealing a lack of ability by the model to generalize. However, in order to accurately assess the models, the use of a more compelling and real-world dataset would be necessary. In this sense, a Bosch Internal Sub-set could be used, consisting in large collection of real-world data acquired under diverse weather conditions and between urban and rural scenes.

Besides that, a new fusion architecture would be developed, consisting in integrating the 3DSSD in the MVX-Net fusion strategy. 3DSSD achieves top-performing results in state-of-the-art, however its performance decreases for the pedestrian class. In this sense, the combination of one of the top-performing detectors based on LiDAR inputs with one of the top-performing fusion strategies would be an interesting and promising study. The MVX-Net architecture implements a voxel-based approach and, therefore, it allows the integration of voxel-based detectors on the top. As such, some modifications would be necessary in the fusion architecture to enable its implementation with a point-based network, such as 3DSSD. In this sense, the voxelization of the point cloud would be removed, and the 3D detector would be integrated after the feature indexing. The input for the 3DSSD would consist in a tensor with the processed point cloud concatenated with the features extracted from the 2D detector outputs. However, as the name indicates, MVX-Net is a multi-modal voxel network and therefore, excluding the voxelization could potentially not make sense, as it would change the main aspects of this architecture. As such, a new fusion architecture could be considered to integrate the 3DSSD detector.

Moreover, very recently, a novel fusion method, Voxel-Pixel Fusion Network (VPFNet) was proposed by [65]. It achieves top-performing results on the KITTI dataset, outperforming the existent methods for multi-modal data. In this sense, a detailed analysis and comparison with the approach proposed under this dissertation, would be necessary in order to evaluate the feasibility to pursue with this fusion architecture.

Appendix A

Training Results

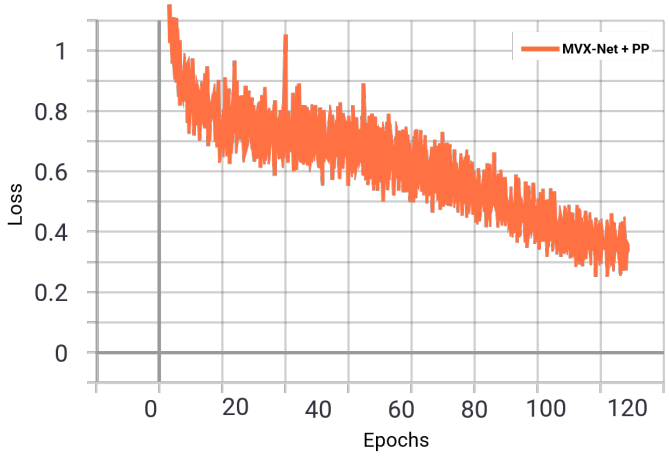


Figure A.1: Loss obtained when training the MXV-Net + PP during 120 epochs. MVX-Net + PP refers to the fusion architecture implemented with PointPillars without any sampling strategy.

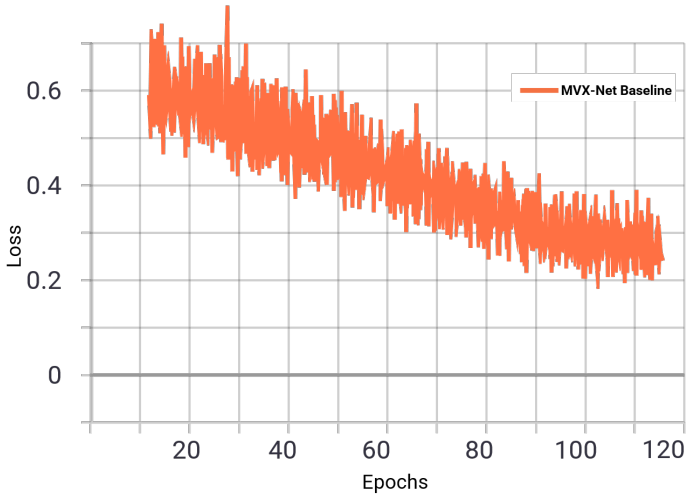


Figure A.2: Loss obtained when training the MXV-Net Baseline during 120 epochs.

Method	Car			Pedestrian			Cyclist		
	E	M	H	E	M	H	E	M	H
3DSSD	88.98	78.72	77.08	56.85	51.97	47.97	87.75	67.57	66.18
MVX-Net Baseline	87.33	76.88	68.60	63.96	56.80	49.49	52.53	40.73	35.12
MVX-Net + PP	84.85	75.65	73.97	57.20	51.62	49.39	74.63	52.23	49.29
MVX-Net + PP (SS2)	84.01	75.27	73.61	56.78	54.57	50.26	75.64	56.61	53.89
MVX-Net + S	87.15	76.96	74.54	66.75	60.94	57.89	73.70	56.47	54.47

Table A.1: Comparison of the results obtained on the KITTI validation set. The AP_{3D} IoU threshold is 0.7 for cars and 0.5 for pedestrians and cyclists. The results are presented in percentage (%). MVX-Net baseline refers to the original architecture, which uses VoxelNet the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars as the 3D detector; MVX-Net + S refers to the MVX-Net with SECOND network. The SS2 refers to the Sampling Strategy applied to the data.

Method	Car			Pedestrian			Cyclist		
	E	M	H	E	M	H	E	M	H
3DSSD	96.35	90.00	89.08	66.31	63.32	58.47	89.67	75.42	73.66
MVX-Net Baseline	90.47,	89.05	71.84	43.38	39.71	35.69	51.06	40.33	38.26
MVX-Net + PP	90.58	88.94	87.60	29.68	30.26	28.12	66.87	49.46	28.12
MVX-Net + PP (SS2)	90.16	88.59	87.04	35.07	37.25	34.54	82.31	62.39	59.17
MVX-Net + S	90.71	89.32	88.22	65.72	60.89	58.19	76.85	60.53	58.30

Table A.2: Comparison of the results obtained on the KITTI validation set. The AP_{AOS} IoU threshold is 0.7 for cars and 0.5 for pedestrians and cyclists. The results are presented in percentage (%). MVX-Net baseline refers to the original architecture, which uses VoxelNet the 3D detector; MVX-Net + PP refers to the proposed architecture of MVX-Net with PointPillars as the 3D detector; MVX-Net + S refers to the MVX-Net with SECOND network. The SS2 refers to the Sampling Strategy applied to the data.

Appendix B

Visual Results

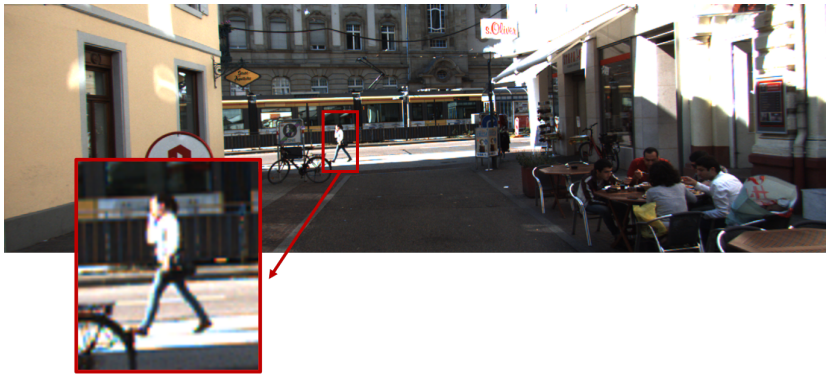


Figure B.1: Example of sample image 5812. The labeled pedestrian in the scene, shown within the selected area, is correctly classified by the model MVX-Net PointPillars (SS2).

Figure B.2: Example of sample image 76. The labeled cyclist in the scene is correctly classified by the model MVX-Net PointPillars (SS2).

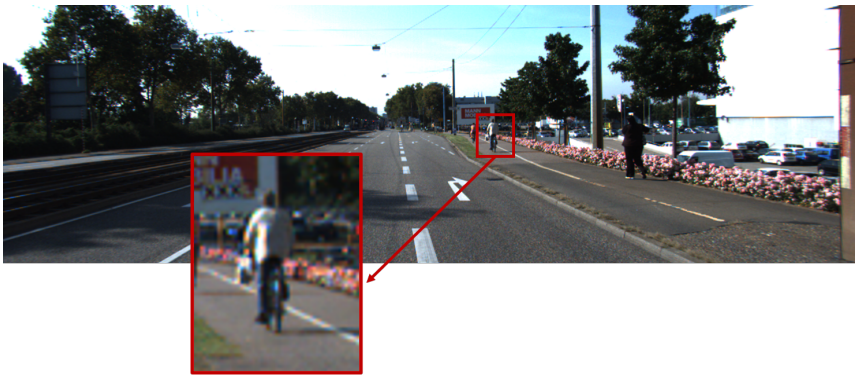


Figure B.3: Example of sample image 811. The labeled cyclist in the scene is correctly classified by the model MVX-Net PointPillars (SS2).

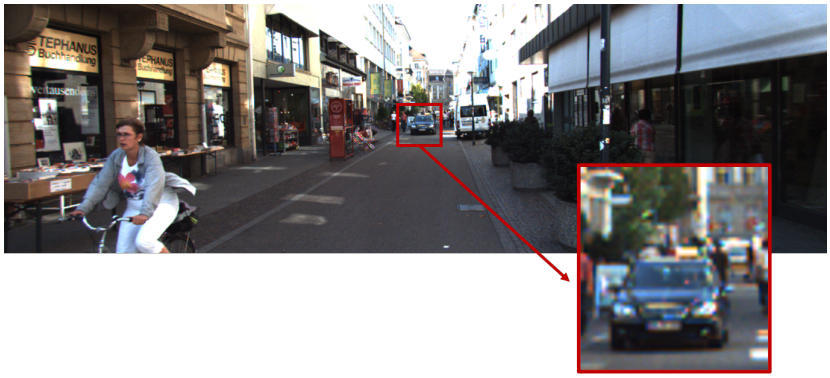


Figure B.4: Example of sample image 1008. There is a labeled car in the scene, which is correctly classified by the model MVX-Net PointPillars (SS2).

References

[1] Yan Yan, Yuxing Mao, and Bo Li. "SECOND: Sparsely Embedded Convolutional Detection". In: *Sensors* 18 (2018). DOI: 10.3390/s18103337.

[2] Santokh Singh. "Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey". In: Oct. 2017.

[3] Mingyang Hao and Toshiyuki Yamamoto. "Shared Autonomous Vehicles: A Review Considering Car Sharing and Autonomous Vehicles". In: *Asian Transport Studies* 5.1 (2018), pp. 47–63. ISSN: 2185-5560. DOI: 10.11175/eastsats.5.47.

[4] Ryan Harrington et al. "The Role of Infrastructure in an Automated Vehicle Future". In: *Bridge* 40 (June 2018), pp. 48–55.

[5] Yanming Chen et al. "Three-Dimensional Urban Land Cover Classification by Prior-Level Fusion of LiDAR Point Cloud and Optical Imagery". In: *Remote Sensing* 13 (Dec. 2021), p. 4928. DOI: 10.3390/rs13234928.

[6] Stamatia Dasiopoulou et al. "Knowledge-assisted semantic video object detection". In: *Circuits and Systems for Video Technology, IEEE Transactions on* 15 (Nov. 2005), pp. 1210–1224. DOI: 10.1109/TCSVT.2005.854238.

[7] Zhengxia Zou et al. "Object Detection in 20 Years: A Survey". In: *ArXiv* (May 2019). DOI: arXiv:1905.05055.

[8] Li Liu et al. "Deep Learning for Generic Object Detection: A Survey". In: Sept. 2018. DOI: arXiv:1809.02165.

[9] Sreedhar Achari. *Practical Implementation of Object Detection On Video with OpenCV and Yolo v3 pre-trained weights on coco data*. May 2019. URL: https://medium.com/@vsreedharachari/practical-implementation-of-object-detection-on-video-with-opencv-and-yolo-v3-pre-trained-weights-a2d2995aac41 (visited on 10/30/2021).

[10] Richard Szeliski. *Computer Vision : Algorithms and Applications*. 2011. URL: http://dx.doi.org/10.1007/978-1-84882-935-0.

[11] Zhong-Qiu Zhao et al. "Object Detection With Deep Learning: A Review". In: *IEEE Transactions on Neural Networks and Learning Systems* PP (Jan. 2019), pp. 1–21. DOI: 10.1109/TNNLS.2018.2876865.

[12] N. Dalal and B. Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[13] D.G. Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[14] Pedro Felzenszwalb, David Mcallester, and Deva Ramanan. “A Discriminatively Trained, Multiscale, Deformable Part Model”. In: vol. 8: June 2008. DOI: 10.1109/CVPR.2008.4587597.

[15] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673. URL: https://books.google.de/books?id=EoYBngEACAAJ.

[16] Xian Da Zhang. *A matrix algebra approach to artificial intelligence*. 2020, pp. 1–805. ISBN: 9789811527708. DOI: 10.1007/978-981-15-2770-8.

[17] Issam El Naqa, Ruijiang Li, and Martin Murphy. *Machine Learning in Radiation Oncology: Theory and Applications*. Jan. 2015, pp. 3–11. ISBN: 978-3-319-18304-6. DOI: 10.1007/978-3-319-18305-3.

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

[19] G.E. Hinton and R.R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science (New York, N.Y.)* 313 (Aug. 2006), pp. 504–7. DOI: 10.1126/science.1127647.

[20] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[21] Steven Walczak and Narciso Cerpa. “Artificial Neural Networks”. In: Dec. 2003, pp. 631–645. ISBN: 9780122274107. DOI: 10.1016/B0-12-227410-5/00837-1.

[22] D. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536. DOI: https://doi.org/10.1038/323533a0.

[23] Daniel T. Larose and Chantal D. Larose. “Discovering Knowledge in Data”. In: *Discovering Knowledge in Data* (2014). DOI: 10.1002/9781118874059.

[24] M. Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88 (2009), pp. 303–338. DOI: 10.1007/s11263-014-0733-5.

[25] G. Parmigiani. “Decision Theory: Bayesian”. In: *International Encyclopedia of the Social & Behavioral Sciences*. Ed. by Neil J. Smelser and Paul B. Baltes. Oxford: Pergamon, 2001, pp. 3327–3334. ISBN: 978-0-08-043076-8. DOI: https://doi.org/10.1016/B0-08-043076-7/00403-4.

[26] Alaa Tharwat. "Classification Assessment Methods: a detailed tutorial". In: (Aug. 2018). DOI: 10.1016/j.aci.2018.08.003.

[27] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[28] Licheng Jiao et al. "A Survey of Deep Learning-Based Object Detection". In: *IEEE Access* PP (Sept. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2939201.

[29] Ross Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[30] Ross Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[31] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.

[32] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: June 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[33] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: vol. 9905. Oct. 2016, pp. 21–37. ISBN: 978-3-319-46447-3. DOI: 10.1007/978-3-319-46448-0_2.

[34] Hanchen Wang et al. "Unsupervised Point Cloud Pre-Training via View-Point Occlusion, Completion". In: *arXiv preprint arXiv:2010.01089* (2021). DOI: arXiv:2010.01089.

[35] Siheng Chen et al. "3D Point Cloud Processing and Learning for Autonomous Driving: Impacting Map Creation, Localization, and Perception". In: *IEEE Signal Processing Magazine* 38 (Jan. 2021), pp. 68–86. DOI: 10.1109/MSP.2020.2984780.

[36] *Velodyne HDL-64E | AutonomouStuff*. URL: https://autonomoustuff.com/products/velodyne-hdl-64e (visited on 08/25/2021).

[37] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? The KITTI Vision Benchmark Suite". In: May 2012, pp. 3354–3361. ISBN: 978-1-4673-1226-4. DOI: 10.1109/CVPR.2012.6248074.

[38] Pei Sun et al. "Scalability in Perception for Autonomous Driving: Waymo Open Dataset". In: June 2020, pp. 2443–2451. DOI: 10.1109/CVPR42600.2020.00252.

[39] Holger Caesar et al. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: June 2020, pp. 11618–11628. DOI: 10.1109/CVPR42600.2020.01164.

[40] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". In: June 2018, pp. 4490–4499. DOI: 10.1109/CVPR.2018.00472.

[41] Shuyang Cheng et al. "Improving 3D Object Detection Through Progressive Population Based Augmentation". In: Nov. 2020, pp. 279–294. ISBN: 978-3-030-58588-4. DOI: 10.1007/978-3-030-58589-1_17.

[42] Jun Xu et al. “3D-GIoU: 3D Generalized Intersection over Union for Object Detection in Point Cloud”. In: *Sensors* 19 (Sept. 2019), p. 4093. DOI: 10.3390/s19194093.

[43] Joram R. Van Der Sluis, Ewoud A.I. Pool, and Dariu M. Gavrila. “An Experimental Study on 3D Person Localization in Traffic Scenes”. In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. 2020, pp. 1813–1818. DOI: 10.1109/IV47402.2020.9304607.

[44] Yulan Guo et al. “Deep Learning for 3D Point Clouds: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.12 (2021), pp. 4338–4364. DOI: 10.1109/TPAMI.2020.3005434.

[45] Weijing Shi and Raj Rajkumar. “Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 1708–1716. DOI: 10.1109/CVPR42600.2020.00178.

[46] R. Qi Charles et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: (2017), pp. 77–85. DOI: 10.1109/CVPR.2017.16.

[47] Charles R Qi et al. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *arXiv preprint arXiv:1706.02413* (2017). DOI: arXiv:1706.02413.

[48] Zetong Yang et al. “3DSSD: Point-Based 3D Single Stage Object Detector”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11037–11045. DOI: 10.1109/CVPR42600.2020.01105.

[49] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. “PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud”. In: June 2019, pp. 770–779. DOI: 10.1109/CVPR.2019.00086.

[50] Bin Yang, Wenjie Luo, and Raquel Urtasun. “PIXOR: Real-time 3D Object Detection from Point Clouds”. In: June 2018, pp. 7652–7660. DOI: 10.1109/CVPR.2018.00798.

[51] Alex Lang et al. “PointPillars: Fast Encoders for Object Detection From Point Clouds”. In: June 2019, pp. 12689–12697. DOI: 10.1109/CVPR.2019.01298.

[52] Shaoshuai Shi et al. “PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection”. In: June 2020, pp. 10526–10535. DOI: 10.1109/CVPR42600.2020.01054.

[53] Di Feng et al. *Deep Multi-modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges*. Feb. 2019. DOI: arXiv:1902.07830.

[54] 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)* (Apr. 2019). DOI: arXiv:1904.01649.

[55] Charles R. Qi et al. “Frustum PointNets for 3D Object Detection from RGB-D Data”. In: (2018), pp. 918–927. DOI: 10.1109/CVPR.2018.00102.

[56] Charles Ruizhongtai Qi et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: (Dec. 2016). DOI: arXiv:1612.00593.

[57] Sourabh Vora et al. “PointPainting: Sequential Fusion for 3D Object Detection”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 4603–4611. DOI: 10.1109/CVPR42600.2020.00466.

[58] Xiaozhi Chen et al. “Multi-View 3D Object Detection Network for Autonomous Driving”. In: (Nov. 2016). DOI: arXiv:1611.07759.

[59] Jason Ku et al. “Joint 3D Proposal Generation and Object Detection from View Aggregation”. In: Oct. 2018, pp. 1–8. DOI: 10.1109/IROS.2018.8594049.

[60] Zhixin Wang and Kui Jia. “Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 1742–1749. DOI: 10.1109/IROS40897.2019.8968513.

[61] Zetong Yang et al. “IPOD: Intensive Point-based Object Detector for Point Cloud”. In: (Dec. 2018). DOI: arXiv:1812.05276v1.

[62] MMDetection3D Contributors. *MMDetection3D: OpenMMLab next-generation platform for general 3D object detection*. 2020. URL: https://github.com/open-mmlab/mmdetection3d (visited on 11/01/2021).

[63] OpenPCDet Development Team. *OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds*. 2020. URL: https://github.com/open-mmlab/OpenPCDet (visited on 11/01/2021).

[64] Martin Hahner et al. “Quantifying Data Augmentation for LiDAR based 3D Object Detection”. In: *ArXiv* (Apr. 2020). DOI: arXiv:2004.01643.

[65] Chia-Hung Wang, Hsueh-Wei Chen, and Li-Chen Fu. “VPFNet: Voxel-Pixel Fusion Network for Multi-class 3D Object Detection”. In: (Nov. 2021). DOI: arXiv:2111.00966.