

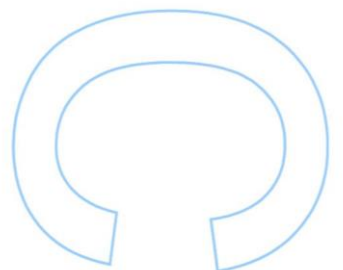
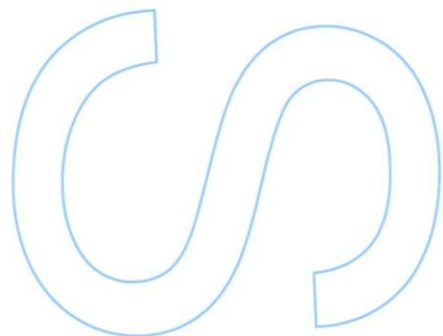
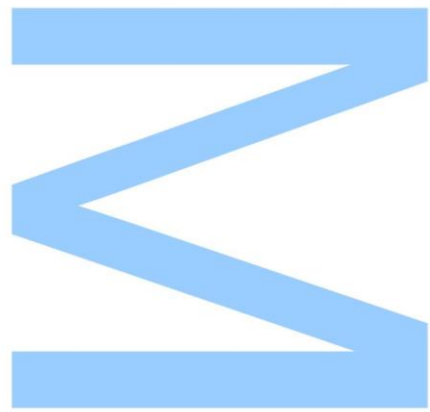
Object Detection in Autonomous Driving based on LiDAR

David Fernandes da Costa

Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2023

Orientador

Inês de Castro Dutra
Professora Auxiliar
Faculdade de Ciências da Universidade do Porto



Abstract

Autonomous driving is characterised by the ability of a vehicle to navigate or at least control some critical aspects of driving without direct human intervention. For this to be possible, a complete understanding of the environment is required to enable these systems to make safe decisions. For this understanding, 3D object detection is an essential aspect, allowing the identification of 3D structures present in the environment, as well as LiDAR sensors, which produce highly detailed point clouds that serve as accurate representations of real-world scenarios.

In Portugal, statistics show that behaviours that reduce human reaction time - alcohol, drugs, fatigue and distraction - contribute to more than 30% of road accidents. This highlights the key role that inference speed plays in the safety of autonomous driving systems. Although this aspect has been recognised as important by the research community, we find that there is a noticeable lack of research dedicated to the inference speed of these systems. To fill this gap, in this dissertation we will investigate the impact of architectural variations, implementations and hardware configurations on the inference speed of 3D object detection models. We have found that the architecture of the models, followed by the hardware used, has the greatest impact on inference speed, while training configurations, such as batch size, have no impact on inference performance. In environments where the data received by the models comes in batches, this can also lead to improvements in inference speed, especially in multi-GPU environments. Our aim is to contribute to the development of standardised evaluation metrics and benchmarking tasks. The ultimate goal of this research is to facilitate the future development of autonomous driving systems that are able to strike a balance between speed and precision.

Keywords: 3D Object Detection; Autonomous Driving; Inference Speed; LiDAR; Machine Learning

Resumo

A condução autónoma caracteriza-se pela capacidade de um veículo navegar ou, pelo menos, controlar alguns aspectos críticos da condução sem intervenção humana direta. Para que tal seja possível, é necessária uma compreensão completa do ambiente para que estes sistemas possam tomar decisões seguras. Para esta compreensão, a deteção de objectos 3D é um aspeto essencial, permitindo a identificação de estruturas 3D presentes no ambiente, bem como os sensores LiDAR, que produzem nuvens de pontos altamente detalhadas que servem como representações precisas de cenários do mundo real.

Em Portugal, as estatísticas mostram que os comportamentos que reduzem o tempo de reação humana - álcool, drogas, fadiga e distração - contribuem para mais de 30% dos acidentes rodoviários. Isto realça o papel fundamental que a velocidade de inferência desempenha na segurança dos sistemas de condução autónoma. Embora este aspeto tenha sido reconhecido como importante pela comunidade de investigação, verificamos que existe uma notória falta de investigação dedicada à velocidade de inferência destes sistemas. Para colmatar esta lacuna, nesta dissertação iremos investigar o impacto de variações arquitectónicas, implementações e configurações de hardware na velocidade de inferência de modelos de deteção de objectos 3D. Verificámos que a arquitetura dos modelos, seguida do *hardware* utilizado, tem o maior impacto na velocidade de inferência, enquanto as configurações de treino, como o *batch size*, não têm impacto no desempenho da inferência. Em ambientes em que os dados recebidos pelos modelos vêm em batches, isso também pode levar a melhorias na velocidade de inferência, especialmente em ambientes multi-GPU. O nosso objetivo é contribuir para o desenvolvimento de métricas de avaliação normalizadas e de tarefas de *benchmark*. O objetivo final desta investigação é facilitar o desenvolvimento futuro de sistemas de condução autónoma que sejam capazes de encontrar um equilíbrio entre velocidade e precisão.

Palavras-chave: Condução Autónoma; Deteção de Objectos 3D; LiDAR; Velocidade de Inferência; Aprendizagem Automática

Agradecimentos

Gostaria de agradecer à minha orientadora, a professora Inês Dutra, por toda a orientação e suporte prestado ao longo da elaboração desta dissertação.

Agradeço à Faculdade de Ciências da Universidade do Porto por consistentemente se preocupar em adequar as condições de ensino para garantir que as minhas incapacidades não fossem um entrave na minha aprendizagem.

Também quero agradecer à minha família pelo apoio continuado que me deu força para nunca desistir, assim como aos meus amigos por estarem sempre ao meu lado.

Por fim, quero dar um especial obrigado ao Sr. Aurélio por sempre acreditar em mim.

Este trabalho é apoiado pelos Fundos Estruturais e de Investimento Europeus na componente FEDER, através do Programa Operacional Competitividade e Internacionalização (COMPETE 2020) [Projeto nº 047264 - THEIA; Referência de financiamento: POCI-01-0247-FEDER-047264].

Dedico este trabalho à minha mãe.

Contents

Abstract	i
Resumo	iii
Agradecimentos	v
Contents	ix
List of Tables	xi
List of Figures	xiv
Listings	xv
Acronyms	xvii
1 Introduction	1
1.1 Context	1
1.2 Motivation and Objectives	1
1.3 Document Structure	2
2 Background	3
2.1 Machine Learning	3
2.2 Object Detection	5
2.3 Point Clouds	6
2.4 Autonomous driving	6

2.5	3D Sensing Technologies	8
2.5.1	Radar	8
2.5.2	Sonar	9
2.5.3	RGB-D	9
2.6	Summary	10
3	3D Object Detection: sensors, datasets and models	11
3.1	LiDAR Sensors	12
3.2	Datasets	13
3.3	Models	15
3.3.1	Point Clouds Representations	15
3.3.2	Single/Two Stages	17
3.3.3	Data Augmentation	18
3.3.4	Evaluation Metrics	19
3.3.5	Benchmarks	21
3.4	Summary	23
4	Exploring Inference Speed	25
4.1	Motivational Exploration	26
4.2	The KITTI Dataset	26
4.3	PointPillars	29
4.4	GD-MAE	31
4.5	Graph R-CNN	33
4.6	Experimental Setup	35
4.6.1	Equipment	35
4.6.2	Dataset	35
4.6.3	Model config	35
4.6.4	Loss functions	36
4.6.5	Experiments	37

4.7	Summary	38
5	Results and Discussion	39
5.1	Training models	39
5.2	Comparing models	41
5.3	Evaluate inference times	43
5.3.1	Exploring the impact of training batch size on inference speed	43
5.3.2	Exploring test batch size variations in inference speed	45
5.3.3	Exploring inference speed in multi-GPU settings	47
5.3.4	Summary	49
5.4	Discussion	51
6	Conclusions	53
6.1	Limitations	54
6.2	Future Work	54
A	Datasets Description	55
A.1	The nuScenes Dataset	55
A.2	The Waymo Open Dataset	56
A.3	The Pandaset Dataset	58
	Bibliography	61

List of Tables

5.1	AP Results for KITTI Car Class Benchmark Tasks.	41
5.2	AP Results for KITTI Pedestrian Class Benchmark Tasks.	42
5.3	AP Results for KITTI Cyclist Class Benchmark Tasks.	43

List of Figures

2.1	Object detection examples	5
2.2	Six levels of automation	7
3.1	Key Players in LiDAR Technology Landscape	12
3.2	Number of papers mentioning these datasets in the last three years	14
3.3	Data Augmentation techniques [12].	18
3.4	Representation of the Intersection and Union for 3D IoU and 2D IoU.	20
4.1	Top View of the KITTI Sensor Setup	27
4.2	Total number of object occurrences.	28
4.3	Number of object labels per class and image for car, pedestrian and cyclist and Class distribution.	28
4.4	PointPillars complete network overview	29
4.5	Pillar Feature Network overview	30
4.6	Backbone overview.	30
4.7	GD-MAE architecture overview.	31
4.8	Sparse Pyramid Transformer overview.	31
4.9	Proposed masking strategies.	32
4.10	Overview of the generative decoder pipeline	32
4.11	Graph R-CNN complete architecture overview.	33
4.12	Dynamic Point Aggregation overview.	34
4.13	Multi-level attentive fusion overview.	34

5.1	Representation of the Train Loss and Overall mAP of the three models on the 3D Object Detetion of the Car, Pedestrian and Cyclist classes	40
5.2	Inference Times and GPU memory usage of the trained models for different batch sizes on the Tesla V100-SXM2 GPU	44
5.3	Inference Times and GPU memory usage of the trained models for different batch sizes on the GeForce RTX3090 GPU	44
5.4	Inference Times and GPU memory usage of the trained models for different batch sizes on the Tesla V100-SXM2 GPU	45
5.5	Inference Times and GPU memory usage of the trained models for different batch sizes on the GeForce RTX3090 GPU	46
5.6	Inference Speed of the trained models with a test batch size 1.	47
5.7	Inference Speed of the trained models with a test batch size 8.	48
5.8	Unified View: GPU Setup, Inference speed, and Car AP_3D(%) analysis (with test batch size 8).	49
5.9	Performance evaluation of the trained models versus best-ranked models	49
A.1	Top View of the nuScenes Sensor Setup	56
A.2	Top View of the Waymo Open Dataset Sensor Setup	57
A.3	Top View of the Pandaset Sensor Setup	59

Listings

Acronyms

AOS	Average Orientation Similarity	JPEG	Joint Photographic Experts Group
AP	Average Precision	RTK	Real-Time Kinematic
AR	Average Recall	mAP	mean Average Precision
BEV	Bird’s Eye View	MAE	Masked Autoencoder
BGR	Blue Green Red	MEMS	Micro Electro Mechanical Systems
FPS	Frames Per Second	RoI	Region of Interest
GNSS	Global Navigation Satellite Systems	RPN	Region Proposal Network
GPS	Global Positioning System	RS	Region Shift
IMU	Inertial Measurement Unit	SRA	Sparse Regional Attention
IoU	Intersection over Union		

Chapter 1

Introduction

1.1 Context

Autonomous driving is defined by the automotive industry as the ability of a car to drive partially or completely by itself, with limited or no human intervention. A crucial aspect of autonomous driving systems is their ability to understand the environment, which is essential for safe decision-making. With this in mind, 3D object detection has emerged as an important part of these systems as it offers the ability to recognize and perceive three-dimensional structures present in the surroundings. In order to create an accurate way of representing real-world scenarios, one of the most prominent sensor technologies used is LiDAR sensors, which enable the creation of highly accurate and dense point clouds of the surrounding environment.

In Portugal, 34.275 accidents were recorded by the ANSR (Autoridade Nacional de Segurança Rodoviária) [3] in 2022. The PRP (Prevenção Rodoviária Portuguesa) [54] reports that 4.5% were caused by alcohol, 4.6% by drugs and up to 20% by fatigue or distraction. This means that 30% of accidents are caused by behavior that reduces reaction time. Therefore, there is a general consensus that the inference times of autonomous driving systems and 3D object detection models are one of the most important components to consider in order to maintain the safety of not only the driver, but also passengers and other road users.

1.2 Motivation and Objectives

In this field, all researchers agree that the most important aspect of 3D object detection models is their precision and robustness of prediction. This is even more important in autonomous driving tasks, where an error in detecting other vehicles or pedestrians can lead to loss of life. However, another important aspect of these models is their inference speed, and although this is recognised in the field, little work has been done to analyse and evaluate the impact of different characteristics on inference speed, as well as a lack of standardised evaluation metrics and benchmarks. With this in mind, the aim of this work is to analyse the impact of changes in

architectures, implementations and hardware on the inference speed of 3D object detection models. Ultimately, we aim to gain a better understanding of how much some of these characteristics affect the overall inference, contributing to the creation of future metrics and benchmark tasks that provide a quantitative measure of the inference speed of models and thus help finding a better balance between speed and precision.

1.3 Document Structure

The remainder of this document is organized into five additional chapters:

- **Chapter 2 - Background:** This chapter provides an overview of key concepts and themes in machine learning, object detection, LiDAR data, autonomous driving, and various 3D sensing technologies and their applications in object detection.
- **Chapter 3 - 3D Object Detection: sensors, datasets and models:** In this chapter, we provide a comprehensive review of the literature on 3D object detection, covering LiDAR sensors, autonomous driving datasets, and various aspects of object detection models, including point cloud representations, approaches, data augmentation, evaluation metrics, and benchmark tasks.
- **Chapter 4 - Exploring Inference Speed:** This chapter serves to outline the motivations, technical aspects of the dataset, model architectures, and experimental procedures for the analysis that we carried out.
- **Chapter 5 - Results and Discussion:** In this chapter, we summarize our research findings by discussing model training, performance comparisons, real-time capabilities, and the balance between precision and inference speed.
- **Chapter 6 - Conclusion:** In this chapter we draw conclusions based on the work's main findings. Providing insights into some of its difficulties, limitations and important topics for future work.

Chapter 2

Background

This chapter aims to explain some important concepts and themes for a better understanding of this work. In the context of machine learning, the learning system, some of its categories and deep learning are explained. After that, a definition of object detection and some of its applications are given. In the context of LiDAR data, a brief explanation is given about point clouds, how they are captured and their types. An explanation is provided about what is autonomous driving, the classification system and the architecture behind automated driving systems. Lastly, there is a discussion about other 3D sensing technologies, including Radar, Sonar, and RGB-D sensors, and their applications in object detection, highlighting their advantages and limitations.

The list below gives an overview of the sections described above:

- **Machine Learning:** explain the learning system and categories, Section 2.1.
- **Object Detection:** an overview of object detection and their models, Section 2.2.
- **Point Clouds:** define and explain what are point clouds, Section 2.3.
- **Autonomous driving:** a brief summary about automated driving and classification systems, Section 2.4.
- **3D Sensing Technologies:** overview of existing 3D sensing technologies, Section 2.5.

2.1 Machine Learning

Machine learning is a field dedicated to understanding and building data analysis methods that are capable of building automated analytical models. It is a sub field of artificial intelligence that operates on the fundamental principle that systems are capable of learning from data, distinguishing patterns and arriving at certain informed decisions while reducing the need for human intervention. Learning algorithms are based on the idea that strategies that worked in the past are likely to continue working in the future. By leveraging data mining techniques as a

basis for the learning system, machine learning algorithms can typically be divided into three components [32]:

1. **A decision process:** At this stage the algorithm is trying, from a set of input data, to estimate some kind of pattern that can solve the task at hand.
2. **An Error Function:** After a prediction is given, an error function based on known examples compares the precision and evaluates the quality of the prediction made by the model.
3. **Optimization Process:** Finally, the weights that affect the decision making of the model are adjusted to reduce the difference between what is estimated by the model and the ground truth.

This learning system of decision, evaluation and optimization is then repeated and its weights are consecutively changed until a certain level of performance is reached. And now knowing the process behind machine learning algorithms, the way these models are trained traditionally fall into these three categories [49]:

- **Supervised machine learning:** In this category, the way learning is done consists of using labeled datasets to train the algorithms and then make predictions for unseen cases. This is the most common learning model because it benefits from readily available labeled data, being the preferred option for regression and classification tasks. Some instances where these methods have been employed are Neural networks, Naïve Bayes, Random forests, among others.
- **Unsupervised machine learning:** Here, the way the model learns is different since only unlabeled datasets are used and then predictions are made for the unseen cases. Since there are no labels it is more difficult to evaluate their performances but with these algorithms it is possible to find hidden patterns on the data, making it a good option for exploratory data analysis. Some algorithms that use these methods are K-means clustering, PCA (Principal component analysis) and K-nearest neighbors.
- **Semi-supervised learning:** The way these algorithms learn is in a middle ground between the two previous categories since they use datasets with labeled and unlabeled data and then make predictions of unseen cases. With this, an attempt is made to take advantage of the unlabeled data to increase the overall performance. This is used when it is easy to get the data but difficult or expensive to label it.

In practice there are other categories such as Active learning [58], Reinforcement learning [34], On-line learning [7], Transfer Learning [71] and many more intermediate categories. Depending on the objectives of the model and the type of data being used, a specific category may be more suitable for the task at hand and yield better results, but even so, we cannot consider one better than the others.

2.2 Object Detection

Object detection is one of the most important branches of computer vision, related with the detection of objects and identification of their classes (like car, pedestrian, cyclists, among others) from images and videos. This technology is of growing interest in a wide range of fields like automotive, medical and security industries by helping in multiple computer vision tasks such as autonomous driving, face detection, medical imaging, robotics and many more. Figure 2.1 shows some of the object detection uses and problem solving capabilities.



Figure 2.1: Use cases for object detection [6].

The goal of object detection is to determine the class of a detected object and the bounding box coordinates that delineates its location. In the past, traditional models relied on manually engineered features and classifiers to accomplish that task. Nowadays, with the rise of deep learning, object detection has seen significant advancements and higher performances. The underlying architecture of most of those deep learning-based object detection models employ convolutional neural networks, since they demonstrated to be extremely effective in the detection of complex patterns and objects from visual data. The training process typically involves the use of annotated datasets where each object corresponds to a labeled bounding box which is a rectangular or cuboidal shape that encloses and defines the spacial extent of the objects. During training, loss functions such as Intersection over Union (IoU) and classification loss are used to evaluate the performance of models and optimize their parameters to accurately classify objects and predict their bounding boxes.

Despite the advancements object detection has made in terms of accuracy and inference times there are some challenges that still need to be addressed, including handling of occlusions and overlapping objects, detect objects at different scales, meeting computational demands of real time applications, among others. Everyday innovative architectures and techniques are created and explored by researchers to overcome those challenges.

2.3 Point Clouds

A point cloud is a discrete set of data points in a 3D coordinate system where each point has a set of Cartesian coordinates that represent a single spatial measurement of a physical object or space. As a result, it is possible to accurately generate a virtual representation of a real object. The two main methods used to capture a point cloud are LiDAR and Photogrammetry.

LiDAR is an acronym for Laser imaging, Detection And Ranging. This method consists of emitting laser pulses that are split into two beams where one beam is directed to the receiving laser detector and the other beam is directed to the object. By comparing the difference between the return times of both reflected beams it is possible to determine the distance of the object from the laser. The multiple return distances to the objects are used to generate a three-dimensional map out of the scanned area [66].

Photogrammetry is the process of creating 3D models from pictures. This method involves taking multiple photographic images from different positions, followed by identifying common points on each image and then with the help of triangulation determine the three-dimensional location of those points [31].

After the capture phase ends, it is necessary to store this data in a way that allows its observation and analysis. There are two possible formats that can be used for this purpose [67]:

- **Organized point clouds:** the data is divided into rows and columns resembling an image like structure. Since the division is based on the spatial relationship between the points, the memory layout is related to the spatial layout that the XYZ coordinates of the points represent.
- **Unorganized point clouds:** the data is stored in a single stream of 3D coordinates where each coordinate represents a point. As a result of the varying size, resolution, density and point ordering, the spatial relationship between points is nonexistent.

In the context of this work, organized point clouds are the basis of the training, evaluation and inference of all the models that will be presented later on.

2.4 Autonomous driving

In recent years, great advancements have been achieved in the field of autonomous driving and they have been revolutionizing the way transportation is perceived and experienced. A vehicle has autonomous driving capabilities if it is able to navigate or at least operate some critical driving aspects without direct human intervention. This is possible with the help of sensors to perceive the vehicle's surroundings, control systems to interpret that information and models to form driving strategies and decisions. These mechanisms can help keeping the vehicle within its

lane with small automatic adjustments at the steering wheel, adjusting the vehicle's speed to stay within the speed limit when it changes or even sending a warning when the tire pressure is too low.

Due to the impact that autonomous driving is expected to have on the automotive industry and adjacent fields, it is becoming increasingly important to create regulations, legislations and standards to safeguard its users. To this end, SAE International (formerly the Society of Automotive Engineers) has developed a classification system that divides into six levels of automation depending on the amount of driver intervention required ranging from non-automation (level 0) to full automation (level 5) as we can see in Figure 2.2.

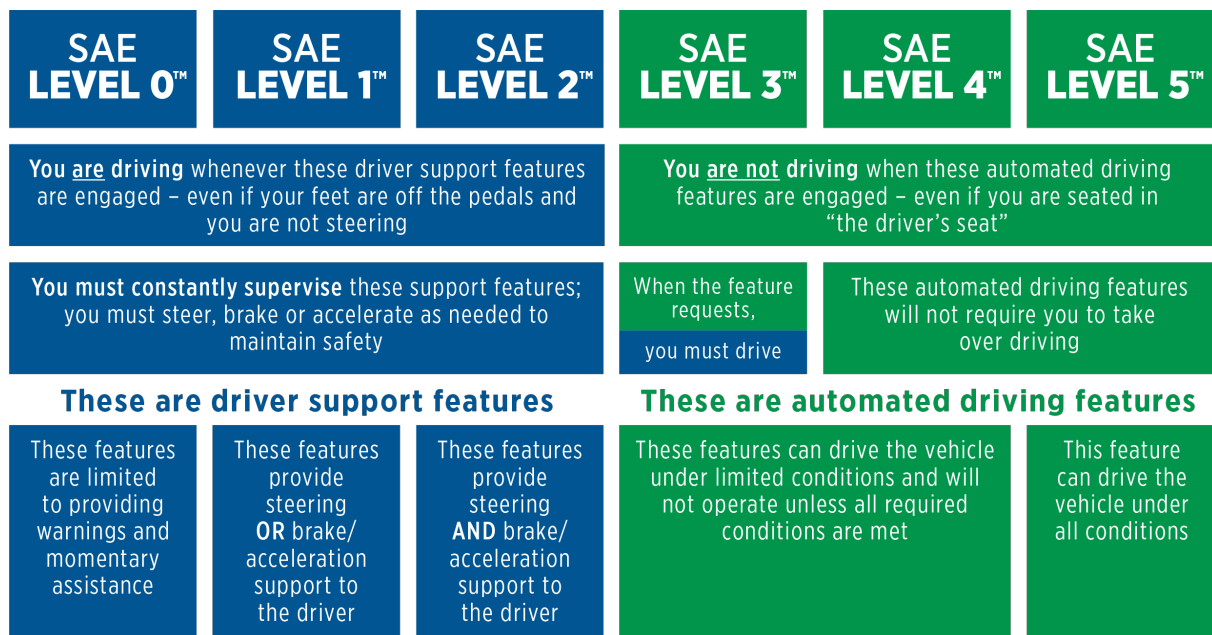


Figure 2.2: The six levels of automated driving [1].

For autonomous vehicles to operate in real-world scenarios, it is necessary to develop automated driving systems. These systems are constituted by the hardware that consists of the car's sensors and controllers such as brakes, lights, steering wheel, among others and the software. Multiple software architectures have been developed and can be roughly divided into 5 components [21]:

- **Perception:** This component is where sensor data is analysed and interpreted to create a sense of the environment around the vehicle. Typically, in this phase, models such as object detectors and semantic image segmentation are implemented.
- **Localization and Mapping:** From the sensor data and output of the perception phase, the local and global location of the vehicle is calculated and a map of the environment is constructed.
- **Prediction:** At this stage, predictions are made about the future trajectories of both the self-driving vehicle and other traffic related entities. Models like motion and overtaking

predictors can be used in this component.

- **Planning:** Based on the information given by the previous components, possible routes that are safe for the vehicle to take are calculated. In this phase techniques like path and maneuver planning are usually implemented.
- **Control:** Finally, after planning a trajectory, this component is responsible for controlling the vehicle in such a way that the difference between the calculated trajectory and the actual trajectory being followed is minimized. It is also responsible for switching control to the human and/or back to the car.

2.5 3D Sensing Technologies

Traditional object detection methods are able to identify objects present in images but due to the lack of depth information, they present difficulties to understand the true 3D structure of those objects. In order to capture a 3D representation of the surroundings most researchers resort to the use of LiDAR sensors, but depending on the purpose under study, other alternatives such as Radar, Sonar or RGB-D sensors can yield better results. These technologies are very important in other applications such as underwater research, robotics, medical imaging and augmented reality where accurate 3D object detection is essential for safe and efficient interaction with the surrounding environment.

2.5.1 Radar

The Radar technology uses radio waves to detect, locate and track objects in the environment. These systems emit radio frequency waves, where by analysing the time it takes for the waves to reach the object and bounce back, together with the frequency shift caused by the Doppler effect, are able to determine the distance, speed, direction, size and shape of objects. Because of this it is widely used for mapping large areas and detecting objects even in adverse weather and low-visibility conditions [46].

This technology offers multiple advantages across a wide range of applications. Its ability to detect objects at long distances allows for early identification of incoming aircrafts, ships, or weather changes. The capability to operate in adverse conditions, such as rain, fog, and darkness, makes it reliable for monitoring and surveillance. Given its precise measurement of velocities and its ability to penetrate materials like foliage and thin walls, its a robust way of tracking object movements. Furthermore, its non-intrusive nature, versatility, and reliability enhance its usefulness.

Despite these advantages, this technology also comes with limitations and certain considerations to be taken into account. Its limited resolution makes it challenging to distinguish closely spaced objects or provide detailed information about their shapes and sizes. The presence

of unwanted signals, such as terrain or buildings, and their susceptibility to interference from other devices or signals can lead to inaccurate readings and reduced precision. The development, installation, and maintenance of these systems are expensive, and require a significant amount of energy to operate. The implementation of these technologies should heed privacy concerns and the finite frequency spectrum available for radar usage. Lastly, it is not suitable for underwater applications because of the limitations that radio waves have in propagating through water.

2.5.2 Sonar

Sonar is a technology that utilizes sound waves that is used for navigation, communication, and detection of underwater objects. These sensors work by emitting sound waves into the water. When these waves encounter an object and bounce back, the time it takes for the wave to come and go is then analysed together with the frequency shift caused by the Doppler effect. This combined analysis allows to capture information about the object's distance, size, and composition [35].

A multitude of advantages can be found in the use of this technology. For underwater navigation, multiple aquatic vehicles including boats and submarines rely on these sensors due to their underwater effectiveness to determine their position and evade obstacles. They are capable of generating detailed seafloor maps, making them indispensable for geological, topographical, and marine habitat studies. The ability to locate objects has also made this technology very important for behavioral and population studies of marine life as well as for military applications. Additionally, it is used for underwater communication between research stations and ships.

Every technology has its pros and cons and this one is no exception. The use of a high-intensity sonar has a negative environmental impact as it affects marine life leading to harm not only of the ecosystems, but also damage and even lead to the death of marine animals. Despite its capacity of providing detailed information about the underwater environment, its resolution remains limited when compared to other sensors. Furthermore, maintaining these sensors can be expensive, as well as complex, since specialized training is required to operate these sensors.

2.5.3 RGB-D

RGB-D refers to the type of sensors that combine color information from traditional image capture sensors with depth data. Depth data can be obtained in several ways, such as time-of-flight, sonar, radar, or even by projecting a pattern into the environment and analysing its deformation. When the depth data is synchronized with the RGB color data, we obtain a more comprehensive representation of the surrounding area. This fusion between camera and depth data offers a balanced compromise between 3D sensing and color information, making this technology useful for a variety of applications [5].

This technology presents a wide variety of compelling advantages. The combination of color

and depth information helps to improve the understanding of the environment with greater precision, proving useful for tasks such as object recognition and segmentation. By utilizing the fused data, it can easily generate intricate 3D models of the environment and people. This capability is a significant advantage for fields like virtual reality, gaming, and archaeology. Other applications include medical imaging, where 3D scans can aid in surgical planning and prosthesis. It also enables safe and efficient navigation through the environment, as well as estimation of poses and gestures. These aspects are crucial for tasks like motion capture and sign language recognition.

Even with these advantages, it also has several limitations. The accuracy of the data obtained by these sensors only works in small ranges, where an increase in distances greatly reduces its accuracy. This limits its usefulness in tasks that require accurate long-range measurements. Susceptibility to environmental interference, such as reflectivity, transparency, and light, leads to errors in depth measurements and limits its outdoor use. Calibration between the RGB and depth sensors is crucial for proper data fusion. The complexity to maintain and achieve adequate calibration requires greater processing power, which makes its use for real-time tasks expensive and limits its application in low-budget scenarios.

2.6 Summary

In this chapter we gain important knowledge into the realm of machine learning, object detection, point clouds, autonomous driving and 3D sensing technologies. Machine learning is understood as a branch of artificial intelligence focused on creating automated analytical models capable of learning from data, which can be categorized as supervised, unsupervised and semi-supervised learning. Object detection is a crucial task in computer vision where we emphasize its role in identifying and classifying objects using deep learning methods. Point clouds are a set of 3D data points representing the spatial positions of objects or environments, which can be captured through LiDAR and photogrammetry, and can be stored in an organized or unorganized fashion. We underscore the significance of autonomous driving along with the SAE International classification systems and main components of automated driving systems. Lastly, diverse 3D sensing technologies such as radar, sonar and RGB-D sensors were explored for their characteristics and important role in applications such as underwater research, robotics, medical imaging and augmented reality.

These insights serve as an important knowledge base, laying the groundwork for a comprehensive understanding of the work that will be described later. In the next chapter we review works related with 3D object detection for autonomous driving based on LiDAR with a focus on the sensors, datasets and models that play a pivotal role in the field.

Chapter 3

3D Object Detection: sensors, datasets and models

In this Chapter, we review works in the literature related to 3D object detection. We begin by diving into LiDAR sensors to give insight into their history, the types of sensors available and their limitations. Following this, we introduce the main datasets used for autonomous driving scenarios, discussing their characteristics and development difficulties. Finally, on object detection models based on LiDAR, we shed light on the point clouds representations, single-stage and two-stage approaches, data augmentation techniques, most common metrics used to assess the performance of the models and available benchmark tasks.

The list below gives an overview of the sections described above:

- **LiDAR Sensors:** insight into available LiDAR sensors, Section 3.1.
- **Datasets:** prominent datasets for autonomous driving scenarios, Section 3.2.
- **Models:**
 - **Point Clouds Representations:** main approaches for point cloud representations, Section 3.3.1.
 - **Single/Two Stages:** overview of single-stage and two-stage approaches, Section 3.3.2.
 - **Data Augmentation:** description of commonly used data augmentation techniques, Section 3.3.3.
 - **Evaluation Metrics:** summary of existing evaluation metrics, Section 3.3.4.
 - **Benchmarks:** overview of available evaluation benchmark tasks, Section 3.3.5.

3.1 LiDAR Sensors

LiDAR is a technology that has revolutionized several industries which, thanks to its use of lasers, has become instrumental in creating high-resolution 3D maps of real-world spaces, impacting fields such as environmental, archaeological, autonomous driving, among others. This technology began to be developed at the beginning of the 20th century, but it was only in the 1960s with the invention of lasers that the development of LiDAR sensors as we know them today began. In the following decades, scientists advanced and developed several new LiDAR systems gaining prominence in the 90s in environmental sciences with the invention of airborne LiDAR systems. With recent advances such as solid-state LiDAR and 360° mechanical scanning LiDAR, its use has expanded to fields such as robotics and autonomous vehicles [50].

Currently, there are several types of LiDAR sensors suited for different applications. Mechanical LiDAR sensors [56] that feature high levels of precision and solid-state LiDAR sensors [37] that are more compact and cost-effective, have been integrated into autonomous vehicles, drones and industrial automation for tasks such as navigation and object detection. Flash LiDAR sensors that, thanks to their ability to capture an entire scene in an instant, are used for real time 3D modeling and cultural preservation [45]. MEMS-based LiDAR sensors which, due to their low power consumption, are preferred for small drones and augmented reality [28, 87]. Multi-line LiDAR sensors that, with the ability to capture multiple laser lines, are used for mapping agricultural, forestry and low visibility areas. Choosing a type of LiDAR sensor depends on factors such as range, size, accuracy and cost, making it a versatile solution across a wide range of industries [15, 57]. For a better perception of the LiDAR sensors market, Figure 3.1 shows the groups of companies that are currently involved in the development, production and commercialization of LiDAR technologies [17].

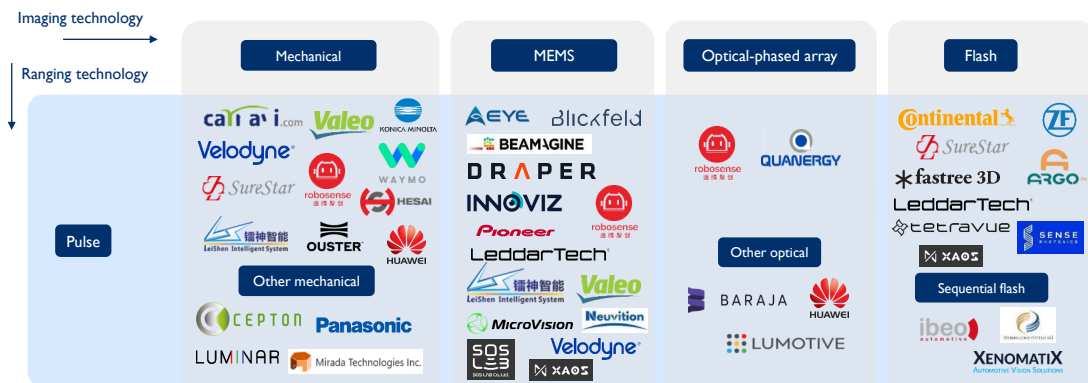


Figure 3.1: Key Players in LiDAR Technology Landscape [17].

Even with this variety of LiDAR sensors, their use presents some challenges and limitations. The biggest challenge is their cost, where a high-quality LiDAR sensor can cost up to tens of thousands of dollars, making its adoption in certain industries unprofitable. LiDAR sensors are very sensitive to environmental conditions where rain, fog or dust affect their performance and reliability. The high amount of data generated by these sensors requires the use of systems

capable of processing and storing large scale data, which incurs additional expenses. Certain materials with high levels of reflectivity, such as metal and glass, can lead to difficulties in the sensor's ability to accurately detect the position of objects. Finally, there remains a significant difficulty in reducing the size of LiDAR sensors while maintaining their performance [57].

In summary, the use of Lidar technologies has impacted several industries thanks to its use of laser-based systems for high-resolution 3D mapping. Originating in the early 20th century and evolving with the invention of lasers, LiDAR sensors have found their place in applications such as environmental sciences, autonomous driving, robotics, and more. There are a plethora of different LiDAR sensors but challenges with their cost, sensitivity and data management remain present. Nevertheless, ongoing research and innovation maintain the growth potential of LiDAR systems, enhancing their understanding and utilization across a multitude of industries.

3.2 Datasets

The use of datasets is quite important in the areas of data science and machine learning as they are the basis for developing models that allow researchers to extract relevant patterns, make accurate predictions and gain insight in various fields. A dataset can be defined as a structured collection of data that is organized and stored in a format that allows its analysis and processing. Depending on the objectives that are being pursued, datasets can be found with different types, sizes and formats. They can be obtained using different methods such as surveys, observations or data streams with diverse data sources such as numerical, textual or even multimedia data.

For the field of 3D object detection the use of quality datasets is crucial for the advancement of research and development as they provide the tools for reseachers to be able to train, develop and evaluate their models and algorithms. Over the years, several prominent 3D object detection datasets have appeared that create new challenges and opportunities that allow researchers to work towards safer and more reliable autonomous systems. The KITTI [23] dataset is one of the oldest datasets renowned for its high-quality 3D bounding box annotations for objects such as car, pedestrian and cyclist that were captured in real-world driving scenarios. The Waymo Open Dataset [64] provides high-resolution sensor data and 3D object annotations with a primary focus on autonomous driving research. The nuScenes [8] is known for its large and comprehensive coverage of multiple complex driving scenarios, weather conditions and time of day. Figure 3.2 shows some of the most used datasets in recent years by researchers [8, 10, 23, 24, 44, 64, 76].

These datasets exhibit characteristics that make them pivotal tools in the development of 3D object detection models. They provide annotations that encompass a set of object labels characterized by their 3D bounding box coordinates, orientation and class name. In some cases, additional information such as speed, acceleration and steering angle are also provided. This information allows models to precisely distinguish objects in 3D space. Furthermore, as these datasets present a diverse set of objects captured in real-world scenarios full of variations in traffic, environment, weather and time of day, they allow the models to maintain accurate detection

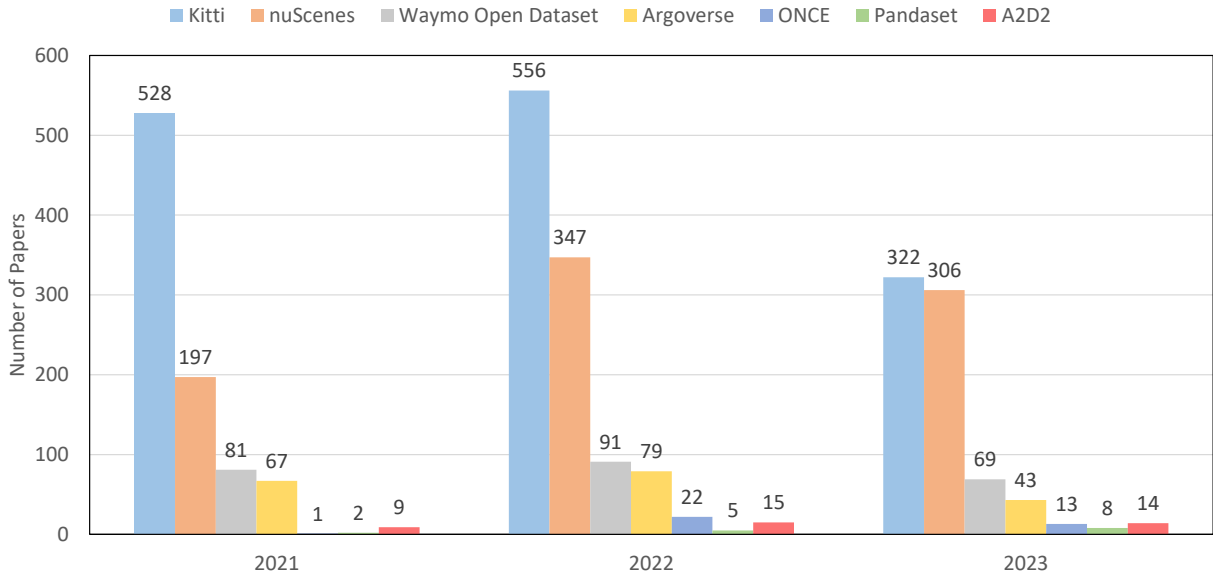


Figure 3.2: Number of papers mentioning these datasets in the last three years. Created from <https://paperswithcode.com/datasets?mod=3d&task=object-detection>.

even in adverse conditions. Typically both LiDAR and camera data are provided which enable sensor data fusion that are important in addressing the problem of object occlusions. Finally, as these datasets come to provide millions of annotated objects, they encourage models to learn meaningful patterns by reducing the effects of overfitting as well as adding enough statistical meaning to provide greater confidence in their performance.

Due to the characteristics of 3D object detection datasets, there are some challenges that arise in the process of creating them [68]. Data annotation is a process that takes a long time and, as it demands the use of specialized tools and expertise, is labor intensive. This becomes even more problematic due to the high number of frames that need to be annotated and can pose a hindrance to the development of new high-quality datasets [41]. Data capture is done from real-world scenarios that are inherently unbalanced and cluttered, thus leading to an imbalanced distribution of object classes and many of them being occluded. Because of this, it is necessary to create methods capable of carefully dealing with possible minorities of labeled objects as well as occluded objects [42]. The use of several different sensors, each with its own characteristics, can lead to synchronization problems that are not always easy to resolve. Finally, it is also necessary to have responsible practices when capturing data because of the sensitive information that real-world scenarios often present.

From all this, it is possible to see that 3D object detection datasets are essential in enhancing the field of computer vision, especially in the context of autonomous driving systems. They have characteristics that allow them to present a very complete representation of the real world. Even so, it is necessary to respond effectively to the various challenges in their creation so that the development of new models can effectively, accurately and robustly detect objects in diverse and complex environments.

3.3 Models

In the ever-evolving area of computer vision, 3D object detection models have seen increasing research interest due to the need for machines to perform increasingly complex tasks. This detection capability allows machines to perceive, identify and understand three-dimensional spaces. Thanks to sensors such as LiDAR that allow the construction of 3D representations of the real world, machines practically gained the human like capability of understanding real world surroundings. This ability becomes even more important in autonomous driving systems that need to have a high perception of the surrounding environment to guarantee safe driving [22, 27, 39]. In this section we will shed a light on some of the main approaches of point cloud representations for efficient model operations, the differences between a single stage and a two stage approach, explain some data augmentation techniques and their impact on model training, and finally a description of the most common metrics used to assess the performance of the models.

3.3.1 Point Clouds Representations

Due to the effect that external factors such as reflectivity, visibility and quality of LiDAR sensors have on the captured raw data, the vast majority of captured point clouds are unstructured, sparse and have high dimensionality. For this reason it is necessary to have point cloud representations with a structure that allows 3D object detection models to efficiently process the point cloud input. Some of the most used approaches present in the literature are point-based, voxel-based, projection-based and graph-based:

Point-based approach refers to all methods that primarily rely on processing, feature extracting and analysing individual points from a point cloud to detect, classify and localize objects. A point-based 3D object detection model normally operates by receiving a point cloud as primary input and applying some preprocessing techniques to improve its quality, then feature extraction techniques are applied to each individual point to allow the capture of relevant information about the environment and finally the model analyses the extracted features to be able to identify objects present within the point cloud. There are some problems with this approach such as dealing with occlusions, varying point densities and noise. Various models and techniques have been developed that have brought advancements to point-based approaches with models like PointNet [11] and PointNet++ [55] which, even though not designed for 3D object detection, introduced several concepts and methods for processing point cloud data, becoming one of the pillars in the development of 3D object detection models. This approach is used by several works, such as PointRCNN [59], 3DSSD [85], IPOD [83] and STD [84].

Voxel-Based Approach in order to be able to detect, classify and locate objects within a point cloud, this method instead of using individual points, discretizes the point cloud into regularly spaced 3D grid cells called voxels, where processing and analysis takes place. 3D object detection models that follow this approach begin by first converting the point cloud into a voxel

grid where each point is assigned to the voxel that contains it. Then, instead of applying feature extraction techniques to each point, they are applied to each voxel and in the end the model analyses the extracted features to identify clusters of voxels corresponding to objects and classify them. One of the great advantages of this approach is its ability to efficiently represent 3D data in a structured format that facilitates the application of convolutional neural networks. Of course this method has its disadvantages, where one of the biggest challenges is the grid resolution in which if the voxel sizes are too small they lead to a high consumption of computational resources but if the voxel sizes are too big there is a loss of detail. Some works that use this approach are PV-RCNN [60], PV-RCNN++ [61], VoxelNet [90], Voxel R-CNN [14], SECOND [78] and VoxSeT [29].

Projection-Based Approach the main idea is to project the 3D point cloud data into a 2D representation so that techniques already developed for 2D object detection can be applied to the projected data. Some projection proposals are Bird’s Eye View where the data is projected onto a horizontal plane from a top-down perspective, Front View where the data is projected along the z-axis corresponding to the sensor viewing angle, Range View where a range image is generated corresponding to a 2D 360° panoramic view representation of the data and Spherical View that maps the data onto a spherical surface to compensate for the sensor’s lens distortion and thus maintain its original perspective. The projection-based 3D object detectors mostly consist of first projecting the point cloud into a 2D representation with the chosen projection scheme, after which traditional 2D object detection models and algorithms are applied to analyse the images, then the depth information of the 3D data is fused to the 2D object detection so that it is possible to determine the equivalent 3D position of the object. The benefit of this approach comes from the extensive methods, architectures and models developed for 2D object detection. Some limitations of projecting the 3D data onto 2D images is the risk of occlusions, loss of information or even errors from the sensors calibrations that affect their accuracy. Some works using this approach are LaserNet [47], PIXOR [79], BirdNet [4] and PointPillars [36].

Graph-Based Approach is a method where point clouds are represented in a graph structure, making it possible to leverage the already known knowledge about graph theory and allowing a better understanding of the scene. Graph-based 3D object detection begins by first transforming a point cloud into a graph representation where the points become the nodes, and for the edges, algorithms are applied to determine which nodes should be connected based on different criteria like neighborhood relationships, spatial proximity or semantical relationships. After this, feature extraction techniques are applied to then be associated with the graph nodes and edges. Next it is applied deep learning architectures designed to process and analyse graph data called Graph Neural Networks [75]. Finally, object detection is carried out where not only the points are considered but also the relationships between points. This approach is a good alternative as it is able to capture contextual information that approaches like point-based and voxel-based would not be able to find since they treat each point and voxel independently. Some problems with this approach are that adding information to point clouds leads to higher computational consumption and finding the best graph construction method is challenging. This approach is used in some works such as Graph R-CNN [80], Point-GNN [62], DCGNN [77],

SVGA-Net [86] and PointGAT [89],

Choosing the correct data representation has a significant impact on the performance of 3D object detection models. This choice depends on several factors such as computational resources, characteristics of the LiDAR sensors and purpose of the specific application. To do this, it is necessary to carry out a lot of experimentation and fine tuning to determine the best suiting representation for the specific task.

3.3.2 Single/Two Stages

The object detection models mostly fall between single-stage and two-stage where choosing between one of these paradigms is a decisive factor that has a substantial impact on the model's performance. These two paradigms use different methodologies and strategies for the detection, classification and location of objects, each with its strengths and drawbacks. Some characteristics and key aspects that differentiate these two paradigms are:

- **Single-Stage:** 3D object detection models that follow this paradigm are developed to be able to detect, classify and locate objects in a single pass through the network. Typically these models are divided into three steps. In the first step, the data is received as input to which, if needed, transformations are applied and augmentation techniques are used to increase the diversity of the data. Then, in the second step, a backbone is used where important features are extracted from the data that improves the speed and performance of the models. Finally, in the last step, a detection head responsible for predicting the bounding boxes and class labels is applied. Single-stage models often offer better inference speeds at the expense of precision. Models like Pointpillars [36], PIXOR [79], VoxelNet [90] and SECOND [78] are some examples of a single-stage model.
- **Two-Stage:** For this paradigm, a more elaborate strategy is used for detection, classification and localization where the models networks are divided into two distinct stages. In the first stage, a RPN is used where, by using proposal generating techniques, the model tries to identify regions in the point clouds that may contain objects to then be proposed as candidates for the next stage. In this second phase, the model takes these object region candidates, applies some fine-tuning for more precise results and then, for each candidate, determines the object contained in that region and the coordinates of the bounding box. Two-stage models often show greater precision and robustness. However, as they have greater computational consumption than single-stage models, they also have longer inference times. Models like GD-MAE [81], Graph R-CNN [80], PointRCNN [59] and BirdNet [4] are some examples of two-stage models.

The process of choosing between single-stage and two-stage must be based on the requirements, objectives and constraints of the tasks at hand. Typically for real-time tasks such as autonomous driving and robotics, single stage models are preferred due to their higher inference speeds,

while for tasks such as medical imaging and facial recognition where there is a greater focus on precision, two-stage models are preferable.

3.3.3 Data Augmentation

In the field of 3D object detection, data augmentation techniques are used to improve the performance and development of reliable models, becoming a crucial component in the preprocessing stage. This is a process where, by applying a set of transformations and operations, greater variety and diversity is introduced to the original data and in doing so improve the learning capabilities of the models. Some commonly used data augmentation techniques are [91]:

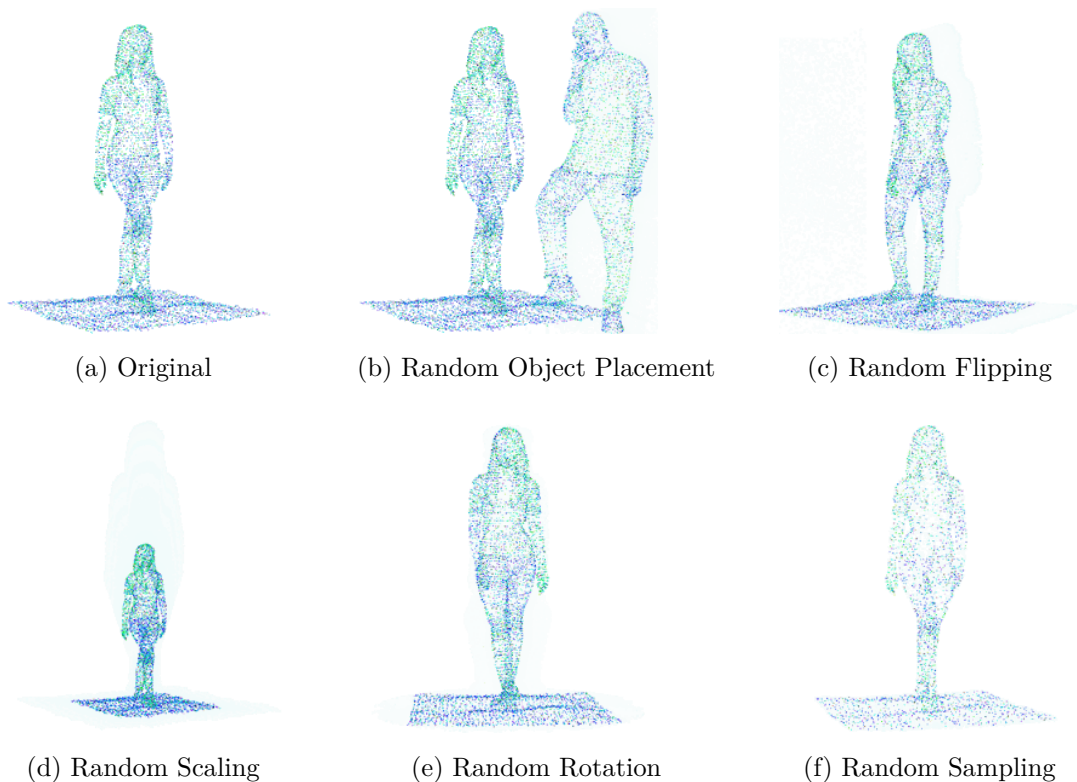


Figure 3.3: Data Augmentation techniques [12].

- **Random Object Placement:** In this technique 3D ground truth boxes and respective point clouds of other frames are randomly added to the original point clouds in the dataset, as shown in Figure 3.3b. The goal of this technique is to increase the range of possible object instances, the amount of data and its diversity.
- **Random Flipping:** This technique, shown in Figure 3.3c, consists of randomly mirroring the point clouds along the x , y or z axis with a probability P . Its objective is to help the model in recognizing objects even if they have different orientations.
- **Random Scaling:** From Figure 3.3d, we can see this technique involves resizing the point cloud data along the x , y , z axis, to have representations of point clouds and objects at

different sizes and distances. Since objects can appear at multiple distances from the sensor, this technique is important to the ability of models to handle objects at different scales.

- **Random Rotation:** This technique consists of applying random rotations to the point clouds around their center or specific pivot point, as shown in Figure 3.3e. This not only helps the model deal with different orientations but also its ability to handle changes in observation angles.
- **Random Sampling:** In this augmentation, points are randomly subsampled from the original point clouds to simulate the capture of point clouds with multiple point densities, as shown in Figure 3.3f. This makes the models more resilient to variations in quality and capture densities of the sensors.

3.3.4 Evaluation Metrics

The use of evaluation metrics is crucial for an effective analysis of the performance and precision of object detection models. Evaluation metrics are important since providing a quantitative measure for the performance of models allows researchers to develop and refine algorithms, optimize and gain insight into the weaknesses and strengths of models, compare different ones and assess their safety and reliability. Metrics such as Intersection over Union (IoU), Average Orientation Similarity (AOS), Average Precision (AP) and Average Recall (AR) are some of the most commonly used:

Intersection over Union: This metric is used to measure the degree of intersection between the object’s ground truth and the predicted region. The IoU is calculated by dividing the volume or area of intersection between the predicted box and respective ground truth box, with the union of the two bounding boxes representing the total space they occupy. This allows for the quantifying of the models predicted bounding boxes quality [65, 88]:

$$IoU_{2D} = \frac{A_{intersection}}{A_{union}} = \frac{A_{intersection}}{A_{pred} + A_{truth} - A_{intersection}} \quad (3.1)$$

$$IoU_{3D} = \frac{V_{intersection}}{V_{union}} = \frac{V_{intersection}}{V_{pred} + V_{truth} - V_{intersection}} \quad (3.2)$$

For the 2D IoU the areas of the predicted and ground truth bounding boxes are represented by A_{pred} and A_{truth} , respectively. In a similar fashion, for the 3D IoU the same thing happens but, instead of using the area, we use the volume where V_{pred} represents the volume of the predicted bounding box and V_{truth} represents the volume of the ground truth bounding box. Both $A_{intersection}$ and $V_{intersection}$ refer to the overlapping of the areas and volumes, respectively, of the bounding boxes. Calculations are made based on the equation 3.1 for 2D IoU and equation 3.2 for 3D IoU. Figure 3.4 gives an illustration exemplifying the intersection and union of 3D and 2D bounding boxes.

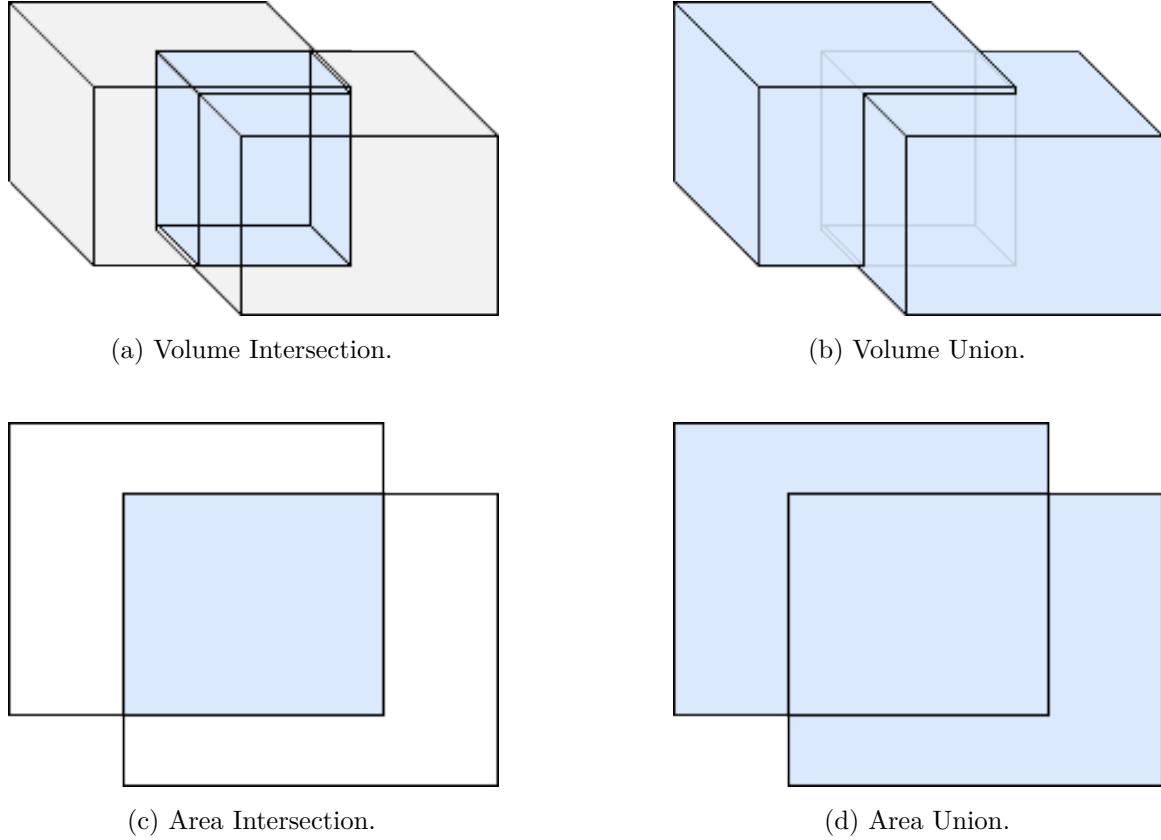


Figure 3.4: Representation of the Intersection and Union for 3D IoU (a), (b) and 2D IoU (c), (d). Adapted from [65].

This gives an IoU score with a value between 0 and 1, where 0 indicates a complete mismatch and 1 indicates a perfect match between the ground truth and the predicted bounding box. With this, it is possible to determine whether a prediction is correct or not where, by defining a value as threshold, if the IoU score is greater than the threshold it is considered as true positive otherwise it is considered false positive.

Orientation Similarity: This metric is used to gauge the quality of the model's orientation estimations by measuring how closely the object's predicted orientation aligns with its ground truth orientation. This value comes from the cosine of the difference between the predicted and ground truth orientation angles:

$$\text{Orientation similarity} = \cos(\theta_{pred} - \theta_{truth}) \quad (3.3)$$

In equation 3.3 we see how this calculation is made where the predicted orientation is represented by θ_{pred} and the ground truth orientation by θ_{truth} . The value will be between -1 and 1 where 1 means that the angle of both orientations is the same indicating high similarity and -1 means that the angle of both differs exactly by 180 degrees indicating low similarity. This metric is important for applications such as autonomous driving and robotics as it provides a quantitative measure of the performance of models estimating the orientations angles of objects.

Average Precision (AP) and Recall (AR): Based on the values obtained in the IoU at a particular threshold, these two metrics are valuable for a deeper understanding of the models performances. The average precision serves to evaluate the trade-offs between precision and recall while the average recall evaluates how good the model is at identifying objects. To this end there is a need for precision which measures the ratio of true positives (objects that were correctly predicted) to the total of predictions made, and recall which measures the ratio of true positives to the total number of objects (objects detected correctly + existing objects failed to be detected). Defined by:

$$Precision = \frac{TP}{TP + FP} = \frac{True\ Positives}{All\ Detections} \quad (3.4)$$

$$Recall = \frac{TP}{TP + FN} = \frac{True\ Positives}{All\ Ground\ Truths} \quad (3.5)$$

The AP calculation begins by first creating a precision-recall curve that plots the precision on the y-axis and recall on the x-axis for different IoU thresholds, which gives a representation of the variation in the model's precision for the different thresholds. After constructing this curve, following a common practice in computer vision, an 11-point interpolated precision (equation 3.6) [43] is applied, which consists of measuring the precision at 11 recall levels, adding them together and dividing this total by 11, ending up with the Average Precision (equation 3.7). The average recall is calculated from the average of the recall values across all classes (equation 3.8).

$$P_{interp}(r) = \max_{r' \geq r} p(r') \quad (3.6)$$

r is the recall level and $p(r')$ is the precision for the respective recall,

$$AP = \frac{1}{N} \sum_{r=1}^N P_{interp}(r) \quad (3.7)$$

where $N = 11$,

$$AR = \frac{1}{n} \sum_{i=1}^n Recall_i \quad (3.8)$$

where n represents the total number of classes.

3.3.5 Benchmarks

A crucial aspect for the development of new models and algorithms is the ability to evaluate and compare their performance with other existing models. For this purpose, different benchmark tasks were developed that provide a standardized way of comparing and evaluating their performances

for different scenarios, thus promoting innovation and guaranteeing the usefulness of the developed models for real-world applications. For 3D object detection models, the tasks most commonly used as benchmarks are:

- **2D Bounding Box Detection:** In this task, the primary objective is to evaluate the model's ability to detect and locate a specific class from 2D images. For the case of models that only use LiDAR data as input, the predicted 3D bounding boxes are projected into the 2D images to then be compared to the respective 2D ground truth bounding boxes. To be able to make this assessment, the AP metric is used with a particular 2D IoU threshold. Some benchmarks use the mean AP (mAP) which consists of calculating the mean of the APs of the various difficulties or classes.
- **Bird's Eye View Detection (BEV):** This task aims to evaluate the ability that models have in detecting a specific class in a top-down projection (BEV) of the scene's point cloud representation. For this, as in the bounding box task, the AP metric with a particular 2D IoU threshold is used, but before applying this metric the 3D data is transformed into a 2D grid representation of the point cloud in a top-down view.
- **3D Bounding Box Detection:** As with Bounding Box Detection, this task aims to evaluate the model's performance in detecting and locating a specific class from the 3D data. As the LiDAR data is already the 3D representation of the scenes, no changes are applied. For this assessment once again the AP metric for a 3D IoU with a particular threshold is used.
- **Average Orientation Similarity (AOS):** The purpose of this task is to evaluate how accurate the models are in estimating the orientation of the objects. This is possible to determine because ground truth labels also contain information about the orientation angle of objects called heading. To do this, the Orientation Similarity metric is used on all objects and then for the specific category, an average of all these scores is made, ending up with an AOS score.

Depending on the research objectives of the benchmarks, only some tasks or other tasks that apply similar concepts may be used. The task of 2D bounding box detection is a fundamental component of several benchmarks such as the COCO [38] and Pascal VOC [18] datasets that focus on 2D object detection. In the case of the 3D Bounding box detection task, in addition to KITTI, it is also used in other benchmarks such as the Waymo Open Dataset and nuScenes datasets, which focus on evaluating 3D object detection and tracking capabilities for autonomous driving scenarios. The last two tasks are more specific to KITTI, but other benchmarks adopt similar methodologies adapted to specific applications or datasets. The principles and challenges addressed by these tasks have a broad impact on the design and evaluation of models, which are fundamental concepts that foster the development of new applications and architectures.

3.4 Summary

With this chapter we gain important insights into the pivotal role that different types of LiDAR sensors play across diverse industries, from environmental sciences to robotics and autonomous driving, as well as their historical development and some characteristics and limitations they present. We emphasize the importance of quality datasets as the foundation for the development and evaluation of 3D object detection models by showcasing datasets known for their comprehensive 3D annotations and real-world driving scenarios, as well as some of the challenges in their creation. Finally, we explore the most used representations of point clouds present in the literature, such as point-based, voxel-based, projection-based and graph-based approaches, and delve into the differences of single-stage and two-stage model paradigms with focus in the trade-offs between speed and accuracy. We also learn about some of the most prevalent data augmentation techniques, discuss crucial evaluation metrics like IoU, AOS, AP and AR, and some well-known benchmark tasks like AOS, 2D bounding box, BEV and 3D bounding box detection.

Most works on object detection in the context of autonomous driving focus on prediction precision. However, there is also a need to analyse their inference speeds, which is another relevant aspect as vehicles need to have a fast enough reaction time to react before something bad happens. Within the field of autonomous driving and computer vision there is a general consensus that inference speed is important which is palpable since all works in the literature were concerned with providing information about their models inference speeds. But this is nothing more than that, as few studies are carried out on the effects of different characteristics of the equipment, as well as an absence of standardized metrics and benchmark tasks that allow their comparison and evaluation. Due to this lack, in the next chapter we will describe the models, dataset and the set of experiences that will later be used to assess and evaluate the effects architectures, implementations and hardware have in the inference speeds of some models.

Chapter 4

Exploring Inference Speed

This chapter aims to describe and explain the methods and procedures employed for the analysis that will be carried out in the next chapter, namely motivation, technical details about the dataset, model architectures and experiments carried out. We start by explaining the motives and reasoning behind all our choices. A description of the data capturing setup, the sensors data formats, the provided annotations and statistics about the KITTI dataset are provided. For each model, the multiple stages of the entire network architecture and proposed methods are described. Lastly, information about the equipment, initial configurations, experiments and loss function are given.

The list below gives an overview of the sections described above:

- **Motivational Exploration:** reasoning behind the choices made, Section 4.1.
- **The KITTI Dataset:** description of the data provided by the dataset, Section 4.2.
- **PointPillars:** overview of the model's network architecture, Section 4.3.
- **GD-MAE:** description of the proposed components and architecture of the model, Section 4.4.
- **Graph R-CNN:** explanation of the created modules and model's architecture, Section 4.5.
- **Experimental Setup:** description of the equipment, initial configurations and overall setup, Section 4.6.

4.1 Motivational Exploration

For the training process, it is necessary to use a high quality dataset so that the models are capable of accurately making predictions on unseen data. Due to the high number of different datasets available, their large size and the cost of training, it is not feasible to work with multiple datasets, so we decided to use only one dataset. After a brief analysis of some of the existing datasets (a brief description of these datasets is available in Appendix A), we chose the KITTI Dataset not only because it is the most popular dataset but also because of its compatibility with the vast majority of available models.

After determining the dataset with which we are going to work, it is now necessary to choose the models for our analysis. For our choice, the models had to have available code and be compatible with the KITTI dataset. Furthermore, the models would have to be embedded in a standard code so that the inference speeds would not be influenced by the code implementation and thus be able to make a real comparison of the performance of the models. Taking this into account, all experiments were made using a Github repository [82] with several models implemented. The models we chose for our analysis are PointPillars, GD-MAE and Graph R-CNN.

4.2 The KITTI Dataset

The KITTI dataset [23] was developed by researchers from the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago with the aim of boosting the development of robotics and computer vision algorithms focused on autonomous driving. It consists of different driving scenarios captured in and around the city of Karlsruhe with diverse sensor data including cameras, 3D LiDAR, GPS and IMU (Inertial Measurement Unit). The annotations of this dataset allow to perform object detection, tracking, semantic segmentation and depth estimation tasks, thus becoming an important benchmark for training and evaluating autonomous driving algorithms.

The sensors setup consists of several components that allow the capture of data from the environment, as we can see in Figure 4.1. It consists of four PointGray Flea2 cameras with a 1.4 megapixels resolution, a frame rate of 15 fps, 1/2" Sony ICX267 CCD and global shutter, where two of them are gray scale (model FL2-14S3M-C) and the other two colored (model FL2-14S3C-C). Each camera uses an Edmund varifocal lens with a focal length of 4mm, an opening angle of approximately 90 degrees and a vertical opening angle of region of interest (ROI) at around 35 degrees. To capture the 3D spatial data, a rotating 3D laser scanner (Velodyne HDL-64E) is used with 10Hz capture frequency, 64 beams, 0.09 degree angular resolution, 2cm distance accuracy, capable of capturing approximately 1.3 million points per second, a horizontal FOV of 360°, a vertical FOV of 26.8° and a range of 120m. For precise navigation and localization, an inertial and GPS navigation system (OXTS RT3003) is also used with a six-axis sensor, a

frequency of 100HZ, with L1/L2 RTK (Real-Time Kinematic) and a resolution of 0.02 meters for position and 0.1 degrees for orientation. The trunk of the vehicle houses a computer running on Ubuntu Linux (64 bit) with two six-core Intel XEON X5650 processors and a shock-absorbed RAID 5 hard disk storage with a capacity of 4 Terabytes.

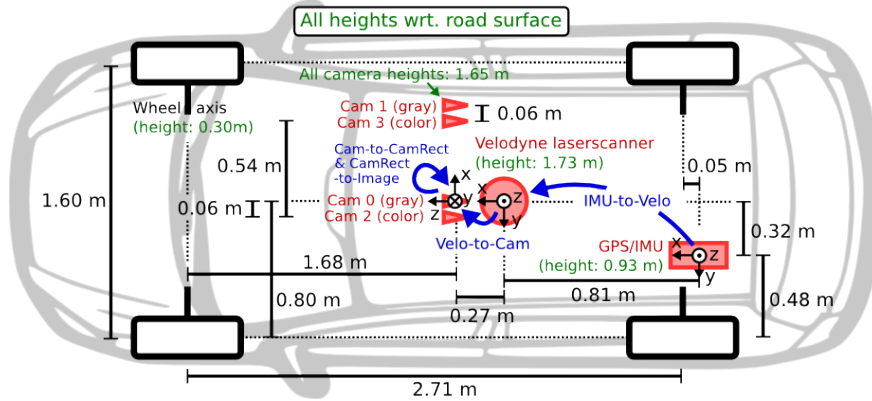


Figure 4.1: Top View of the KITTI Sensor Setup [23].

The data that makes up this dataset includes images, GPS/IMU information and point clouds. The images are stored in color and gray scale format with loss-less compression using 8-bit PNG files. Both the hood and the sky region have been cut out of the image. In addition to the original images, it also provides rectified images with an average size of approximately 0.5 Megapixels. For each frame, a text file stores 30 GPS/IMU variables, which include altitude, global orientation, speeds, accelerations, angular rates, accuracies and satellite information. For the case of accelerations and angular rates both are defined by two coordinate systems, one attached to the vehicle body (x, y, z) and the other mapped to the tangent plane of the earth's surface (f, l, u). On occasional communication outages that lasted around a second with the OXTS device the values were interpolated linearly and set the last three entries to '-1' to indicate the missing information. Velodyne scans are stored as floating-point binaries where each point contains its coordinates (x, y, z) and an additional reflectivity value (r). The number of points per scan is not constant but on average each frame has a size of around 1.9 Megapixels which roughly corresponds to 120,000 3D points.

For each object within the camera's field of view, annotations are provided in the form of 3D bounding box tracklets, represented in Velodyne coordinates. Each object is given a class and its 3D size (height,width,length) and for each frame the rotation and translation of the objects. In addition to the raw data, KITTI's website provides a development kit with details on data formats and utility functions for reading and writing label files, and also hosts evaluation benchmarks for various computer vision tasks such as stereo, optical flow, visual odometry, SLAM, 3D object detection and 3D object tracking.

The object classes are 'Car', 'Van', 'Truck', 'Pedestrian', 'Person (sitting)', 'Cyclist', 'Tram' and 'Misc'. In total there are 14,999 images and the corresponding point clouds, with a total of 80,256 annotated objects. Figure 4.2 shows the number of annotated objects for each possible class. As the dataset had a greater focus on annotating Cars and Pedestrians, these appear in

considerably greater quantities than the other classes. For the classes that we are going to use for the training phase of the models we can see that there is an imbalance, since 60% of the labels are Car while 34% are Pedestrians and only 6% are Cyclists, as we can see in Figure 4.3d. Regarding the distribution of classes per frame, from Figures 4.3a, 4.3b and 4.3c, we can see that a frame can contain up to 13 Cars, 19 Pedestrians and 6 Cyclists.

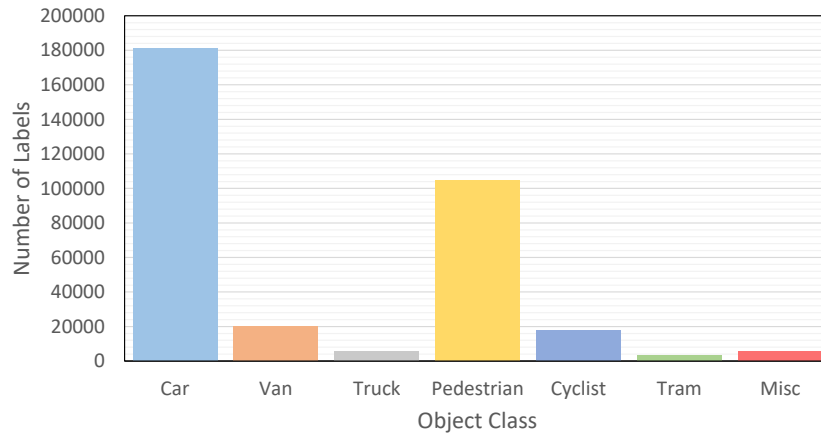
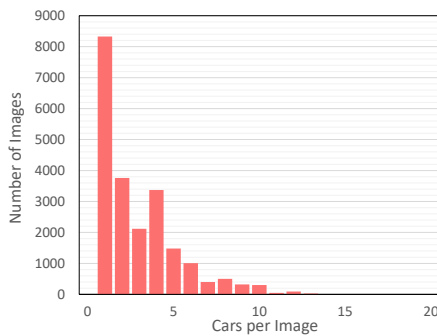
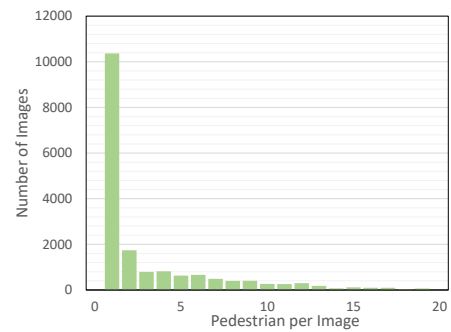


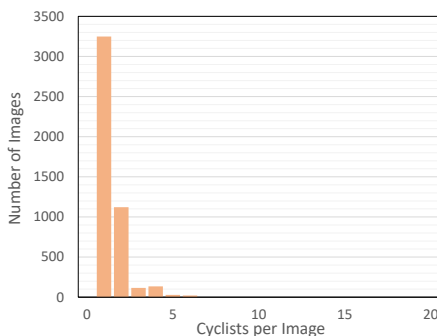
Figure 4.2: Total number of object occurrences. Adapted from [23].



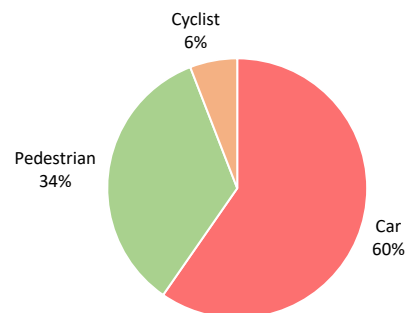
(a) Number of Object Labels for Car.



(b) Number of Object Labels for Pedestrian.



(c) Number of Object Labels for Cyclist.



(d) Proportion of Labels for Car, Pedestrian and Cyclists

Figure 4.3: Number of object labels per class and image for Car (a), Pedestrian (b) and Cyclist (c) and class distribution (d). Adapted from [23].

4.3 PointPillars

The PointPillars [36] network is a deep learning model designed for 3D object detection from point cloud data that tries to solve challenges related to processing large-scale point cloud data for real time applications like autonomous driving. This model achieves real time processing capabilities by efficiently encoding the point cloud into a pillar-based representation and thereby reducing computational and memory usage. Furthermore, its architecture is not only scalable thanks to its ability to handle point clouds of different sizes and densities, but it is also capable of maintaining high accuracy for 3D object detection tasks. As shown in Figure 4.4, this model is basically divided into three main stages: a Pillar Feature Network, a Backbone and a Detection Head.

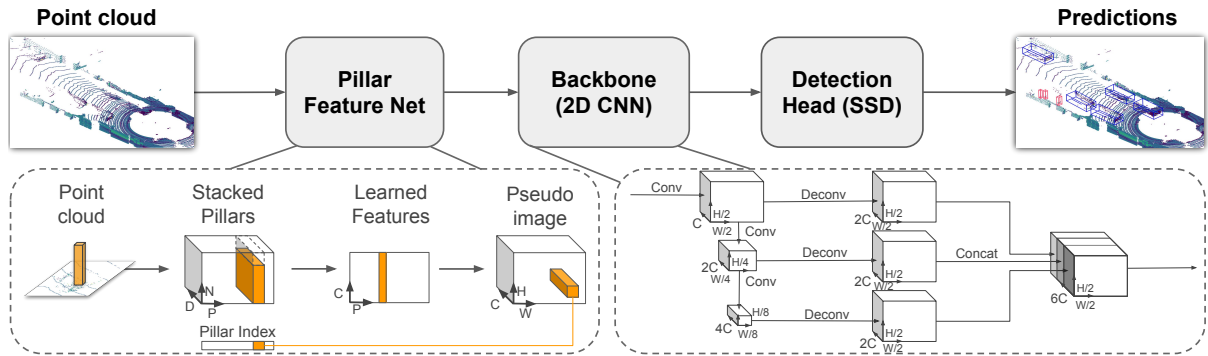


Figure 4.4: PointPillars complete network overview [36].

Pillar Feature Network: In this first component a feature encoder network is used to convert the point cloud data into a pseudo-image. Each point in the point cloud is represented by its x, y, z coordinates and its reflectivity r . It starts by discretizing the point cloud into an evenly spaced grid along the x-y plane to create a set of P pillars, with the total count of pillars denoted as B . Then for each pillar the points are augmented with additional values x_c, y_c, z_c, x_p , and y_p , where c indicates the distance to the arithmetic mean of all points in the pillar and p represents the offset to the x, y center of the pillar. This augmentation causes each point to now be represented in a 9 dimensional space, denoted as D . Then, because of the sparsity of the pointcloud that leads to many of the pillars in the set being empty and those that are not empty to have few points, a limit is imposed both on the number of non-empty pillars per sample, defined as P , as well as the number of points per pillar, denoted as N , thus creating a dense tensor of size (D, P, N) . After that, for each point, a linear layer is applied, succeeded by a BatchNorm [33] and ReLU [51], thus giving a tensor of (C, P, N) size, where C represents the number of channels. Subsequently, a max operation is performed across the channels which produces an output tensor with dimensions (C, P) . And finally, the extracted features are put back to their original locations on the pillars, resulting in a pseudo-image with dimension (C, H, W) , where H represents the height and W the width of the canvas. Figure 4.5, gives a better understanding of this network.

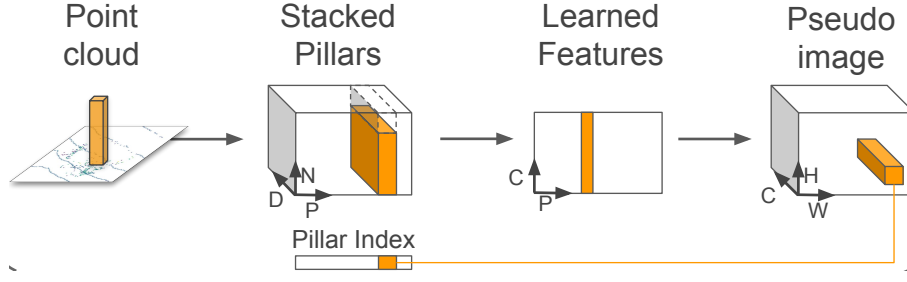


Figure 4.5: Pillar Feature Network overview [36].

Backbone: The backbone used by this model is similar to the RPN (Region Proposal Network) proposed by Zhou and Tuzel (VoxelNet) [90]. The network consists of three blocks of fully convolutional layers, where for each block (S, L, F) two sub-networks are applied. The first is a top-down network, that downsamples the feature map by applying L 3x3 2D convolutions with F output channels where the first convolution has a stride S while the rest have a stride size of 1, to which then a BatchNorm and a ReLU operation is applied. The second consists of upsampling the final features of the output from the top-down network for each block, $\text{Up}(S_{in}, S_{out}, F)$, from an initial stride S_{in} into a fixed size S_{out} by applying a transposed 2D convolution with F final features, followed by a BatchNorm and ReLU operation, ending with a final output of a concatenation of all the features with the different strides. The full structure is shown in Figure 4.6.

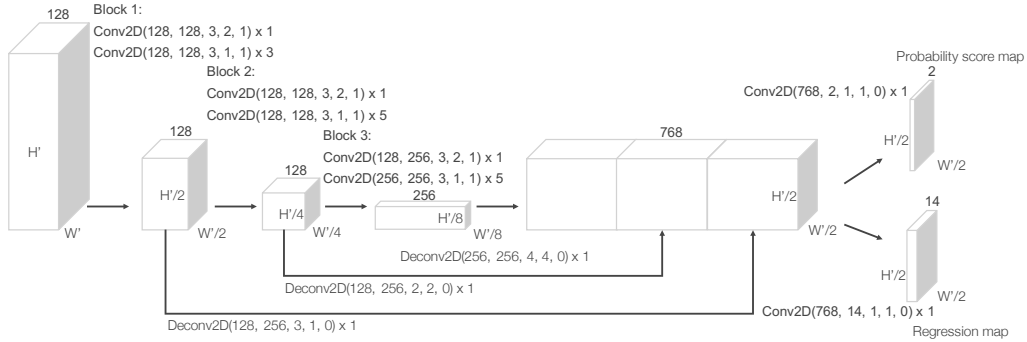


Figure 4.6: Backbone overview [90].

Detection Head: This component uses a Single Shot Detector (SSD) [40] for the 3D object detection. This approach produces bounding boxes and scores for any object present in those boxes, to which a non-maximum suppression step is applied to filter out redundant detections and thus produce the final detections. A 2D Intersection over Union (IoU) is used to match the priorboxes to the ground truth and given a 2D match, both height and elevation become additional regression targets. There is a multi-scale feature map to add extra convolutional layers to the network base (VGG-16) [63] that allows detections at multiple scales; convolutional predictors where for each layer added it manages to generate a set of detection predictions by using convolutional filters; default boxes and aspect ratios associated with each cell on multiple feature maps to predict the offset and class scores for each box.

4.4 GD-MAE

The GD-MAE [81] is a model created with the aim of solving the difficulty that MAE (Masked Autoencoders) [30] have in dealing with large-scale 3D point clouds. As shown in Figure 4.7, this model consists of 2 components where the first proposes a Sparse Pyramid Transformers to be the multi-scale encoder and the other component proposes a Generative Decoder with the objective of creating a more simplified form of MAE-style pre-training on multi-scale backbones. This model manages to eliminate the need for a complex decoder since it can automatically expand the visible region to the underlying masked area.

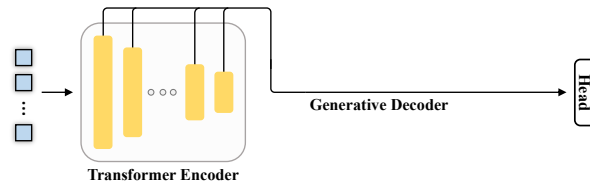


Figure 4.7: GD-MAE architecture overview [81].

Multi-Scale Encoder: For better feature extraction from sparse LiDAR point clouds, as shown in Figure 4.8, a hierarchical transformer architecture called Sparse Pyramid Transformer is proposed. As with Pointpillars, the input point cloud data is converted to a 2D pseudo image with the features and coordinates of the pillars being used as non-empty tokens. Then to generate feature maps of different scales these tokens are used as input of the sparse pyramid transformer which consists of three stages where in the first stage the tokens are processed only by the encoder and in the other two stages the tokens pass first through a SpConv [25] to be downsampled and then processed by the encoder. The encoder layer consists of two SRA (Sparse Regional Attention) [20] used to perform self-attention on same region tokens with the added positional encodings to the input embeddings of the decoder and one RS (Region Shift) [20] that by adding half of region size offsets and grouping the tokens based on the new regions is able to capture useful local contexts. For each stage i is applied L_i encoder layers, after which it broadens the tokens perceptual fields and aggregates long-range contexts. Due to the importance that the local features have for smaller objects a shortcut is added to fuse the previous features with the encoders output and then use a subConv [26] for an adaptive fusion.

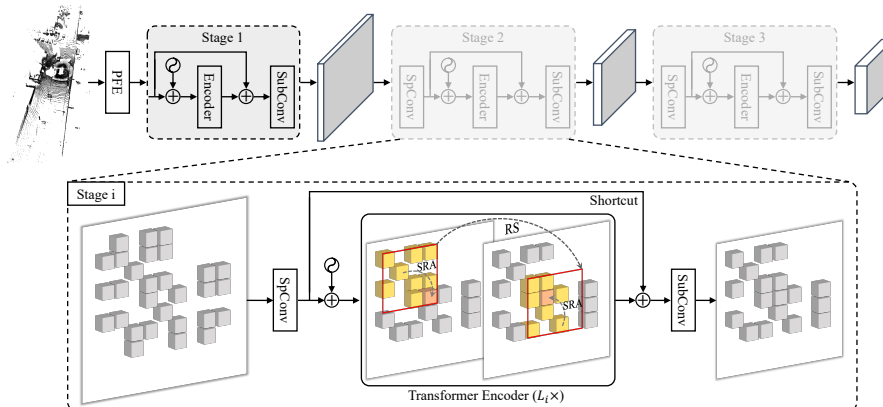


Figure 4.8: Sparse Pyramid Transformer overview [81].

Masking: After the multi-scale encoder has been used, it is then necessary to apply a mask but due to masking strategies like MAE leading to a large granularity of the last stage output tokens and making it difficult to recover from unmasked parts, three masking strategies with different granularities were designed by the authors. The first strategy is the Block-wise Masking shown in Figure 4.9(a) which involves masking a portion of non-empty tokens from downsampled feature maps and then reconstructing it, i.e., entire regions are masked out and then the model predicts the content of the masked areas. This masking focuses the model into capturing at a coarse level large-scale features and relationships. The second strategy is the Patch-wise Masking shown in Figure 4.9(b) which applies block-wise masking but for a smaller granularity that involves masking some of the upsampled tokens (granularity equal to that of the input tokens) but instead of masking entire regions, only patches within the tokens are masked. The focus of this masking is for the model to capture mid-level features and finer details as only smaller patches of the data have to be reconstructed. The third strategy is the Point-wise Masking shown in Figure 4.9(c) which applies a masking of the smallest granularity which involves masking of the individual points within the tokens. This masking improves the model’s ability to capture detailed features and relationships, making it important in capturing fine geometric details and local patterns.

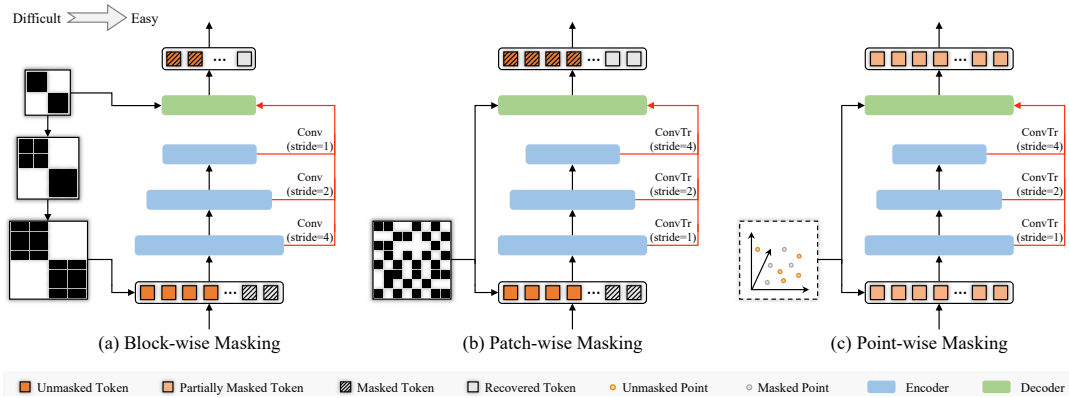


Figure 4.9: Proposed masking strategies [81].

Generative Decoder: This method of fusing the hierarchical features for reconstruction was proposed with the aim of allowing varied masking granularities and to output multi-scale representations for downstream tasks. The visible tokens for the different stages from the multi-scale encoder are used as input of the generative decoder. After this, the tokens are unified to the same scale, which is determined by the granularity of the masking strategy used. To carry out this unification we start by transforming the sparse tokens into a dense 2D feature map that involves scattering their features at their corresponding coordinates, then applying a series of standard convolutions, determined by scale, to fuse the multi-scale features, and finally indexing the feature vector (which relates to the coordinates of masked tokens) to the before mentioned step, as shown in Figure 4.10.

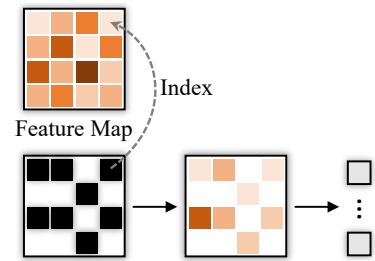


Figure 4.10: Overview of the generative decoder pipeline [81].

Reconstruction Target: This method aims to reconstruct the point clouds that fall within each masked token. Due to the varying number of points each token may contain, at most K points are randomly sampled to be used as the reconstruction target. To ensure stability during training, the point cloud data is normalized by transforming it into token related local coordinates. Therefore, given the features of the masked tokens, these are projected with a linear function and then a reshape operation is applied, ending up with a reconstructed point cloud. To measure how closely the reconstructed point clouds match the original point clouds a loss function called l2 Chamfer Distance [19] is used.

4.5 Graph R-CNN

The Graph R-CNN [80] is a 3D object detector that can be applied to other existing 3D detectors to improve their performance. As shown in Figure 4.11, this model created 3 modules where one of them proposes a Dynamic Point Aggregation to efficiently manage sparse point cloud data, the other proposes the RoI-graph Pooling with the objective of robustly capturing RoI features and lastly proposes a Visual Features Augmentation to fuse image features to compensate for sparse point cloud data.

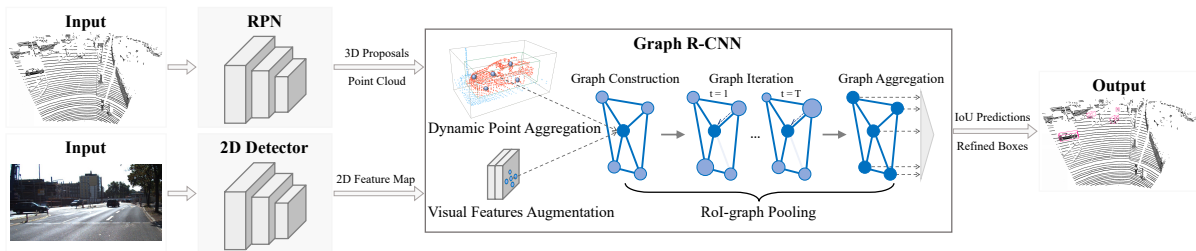


Figure 4.11: Graph R-CNN complete architecture overview [80].

Dynamic Point Aggregation: This method is used to efficiently group and sample points along with its features within each proposal, as shown in Figure 4.12. The process involves two main components. The first component is Patch Search, which optimally groups points within each enlarged proposal by dividing the scene into patches and looking only for points that fall within the patches occupied by the proposal. This process consists of first transforming the rotated boxes into axis-aligned boxes to make it easier to identify occupied patches, then two index arrays (point2patch and patch2box) are built to efficiently associate points with patches and patches with boxes, and finally the points are grouped for each proposal using the index arrays. The second component is Dynamic Farthest Voxel Sampling which tries to ensure that the entire object is sampled by maintaining a balance between efficient and accurate sampling. The process involves splitting the proposals into evenly distributed 3D voxels, then dynamically adjusting the voxel size based on the distance from the LiDAR sensor to ensure an efficient sampling of nearby objects and accurate sampling of distant objects, and finally iteratively sampling the most distant non-empty voxels using a technique called farthest point sampling (FPS) [48].

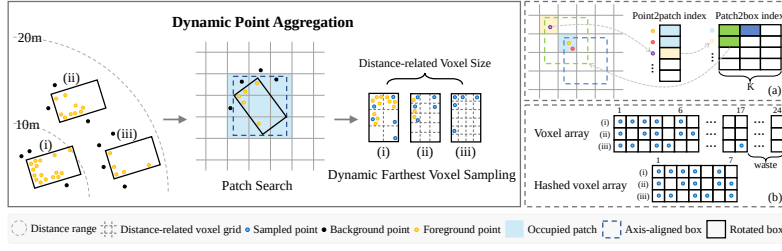


Figure 4.12: Dynamic Point Aggregation overview [80].

RoI-graph Pooling: This process aims to extract useful features from the 3D proposals by building local graphs and iteratively aggregating information. In the Graph Construction for each 3D proposal the sample points are treated as nodes and connected to each other based on the K-Nearest Neighbors [53] where the edges connect to the closest nodes in geometric distance. To preserve the fine-grained features, for each node the original neighbor points within a certain radius "r" are encoded using PointNet [11] and to differentiate similar nodes in different proposals, 3D proposal's local corners are added to each node to avoid ambiguity. After being constructed, the graph is processed iteratively where, at each iteration step, each node aggregates information from its neighboring nodes using a module called EdgeConv [70] to update the node's state based on the aggregated information. Finally, after several iterations, the nodes' features are fused to robustly capture RoI features by proposing a Multi-level attentive fusion, as shown in Figure 4.13. The node's features from different iterations are concatenated and passed through multiple Multilayer Perceptrons [69] to learn the weights for each channel which are then used to reweight the features and thus enhance their importance. The chosen aggregation technique was max pooling due to its better performance. To prevent overfitting, Dropout is used and a shortcut connection is added to combine features efficiently.

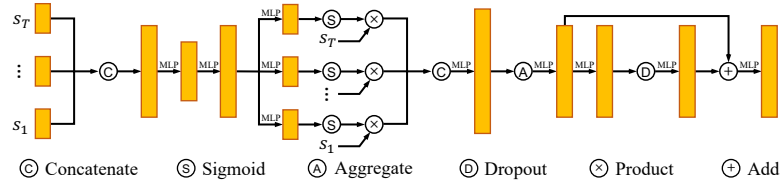


Figure 4.13: Multi-level attentive fusion overview [80].

Visual Features Augmentation: This module is a fusion method used to effectively fuse geometric features from LiDAR with semantic features from images and thus reduce false positives. For the geometric features from LiDAR a technique called Cut-and-Paste Augmentation [16] is used to augment the training data which involves randomly selecting parts of the point clouds of a sample and pasting it on other samples to create new training examples and consequently improve performance and speed of the training process. However, as this Graph R-CNN model is capable of extracting RoI features from raw point clouds with the RoI-graph Pooling module, this model does not depend on this technique and therefore is only used when necessary to pretrain the RPN. For the semantic features from images these features are extracted using a pretrained 2D detector, then applied two 1X1 convolutional kernels to reduce dimensionality allowing it to learn to choose features that significantly contribute to the refinement process and to simplify the

fusing of low-dimensional point features from the point clouds with the high-dimensional image features. Then the graph nodes are projected to the corresponding locations on the image, the feature vector at each of the projected pixels location are collected using a bilinear interpolation and finally these collected features are appended to the initial state of each node in the graph.

4.6 Experimental Setup

In this section we provide a detailed description of the experimental procedures and data analysis methods. To ensure a fair and accurate analysis of the models under study, we chose to maintain, whenever possible, the same configuration values used by the authors in the experimental setups. This extends to the data used from the dataset, model configuration, data augmentation techniques applied, the Loss functions and training configurations.

4.6.1 Equipment

For this dissertation, the Department of Computer Science at FCUP provided a machine equipped with an NVIDIA GeForce RTX 3090 GPU with 24 GB of RAM. This machine was the basis for all research, analysis and model training. To make a comparative analysis of the influence that hardware has on inference speeds, in addition to this machine, they also provided a machine equipped with four Tesla V100-SXM2 GPUs with 32GB of RAM each.

4.6.2 Dataset

The number of frames present in this dataset is 7,481 for training and 7,518 for testing, but as the testing set is not labeled, only the training set is used for the models training phase. Because of this, the training set will be divided, following the division criteria provided by the KITTI benchmark, into a training set of 3,712 frames and a validation set of 3,769 frames.

4.6.3 Model config

For the PointPillars model the x, y, z detection ranges are $[(0, 70.4), (-40, 40), (-3, 1)]$ for the car class and $[(0, 48), (-20, 20), (-2.5, 0.5)]$ for the pedestrian and cyclist class. It has a voxel size of (0.32m, 0.32m, 4m), a maximum of 16,000 pillars with at most 32 points per pillar. It has an anchor width, length and height of (1.6, 3.9, 1.5) for the car class, (0.6, 0.8, 1.73) for the pedestrian class and (0.6, 1.76, 1.73) for the cyclist class. For data augmentation, a lookup table was created with all 3D ground truth boxes and their respective point clouds. This lookup table, for each frame, will be used to randomly choose 15 ground truths from the car, pedestrian and cyclist classes and placed in the respective frame's point cloud. A random flip along x axis, a random rotation with an angle between $[-0.78539816, 0.78539816]$ radians and a random scaling with a range of $[0.95, 1.05]$ are also applied. The model is trained for 80 epochs.

For the GD-MAE model, the x , y , z detection ranges are $[(0, 69.12), (-39.68, 39.68), (-3, 1)]$ meters respectively, with a voxel size of $(0.32m, 0.32m, 4m)$. In the pre-training phase the data augmentation techniques implemented are random flipping along the x axis with a probability of 0.5, random scaling with a range of $[0.95, 1.05]$ and a random rotation with an angle of $[-0.78539816, 0.78539816]$ radians and during fine-tuning a Cut-and-Paste Augmentation was used. The mask ratio is 0.75 and for the point reconstruction the K is set to 64. The Adam optimizer was used, with a learning rate of $3 * 10^{-3}$ and a learning rate decay of 0.1. Because the KITTI dataset is being used, the model's network is the same as that of the PointPillars. The model is trained for 80 epochs.

In the Graph R-CNN model, the authors proposed three implementations to be used with KITTI. As two of the proposals are based on PointPillars, to add variety we decided to choose the Graph-Po implementation that uses the 3DSSD [85] model as the RPN and since this model does not depend on the Visual Features Augmentation module, we will not use it and thus remove the need for a 2D detector. The RPN, instead of using a detection range, chooses at random 16K points in each frame to be used as input. The first data augmentation technique adopted was the same as PointPillars, where ground truth boxes randomly chosen from other frames are added to the current point cloud. Other applied techniques were a random rotation with an angle between $[-0.78539816, 0.78539816]$ radians, a random flip along x axis and a random scaling $[0.95, 1.05]$. The Adam optimizer was used with an initial learning rate of $2 * 10^{-3}$ and a learning decay of 10 at epoch 40. For the Dynamic Point Aggregation of each proposal, 256 points are sampled with the σ set to 0.4. In the Patch Search the number of boxes per patch is set to 32 and the patch size is set to 1.0. In the dynamic farthest voxel sampling the hash size is 4099, $\lambda = 0.12$ and $\delta = 50$. For the RoI-graph Pooling the radius r is set to 0.4, the graph is updated with T set to 3 and the $K = 8$ in the K-Nearest Neighbors. The Multi-level attentive fusion embedding dimension is 256 and a dropout ratio set to 0.1. The model is trained for 80 epochs.

4.6.4 Loss functions

In Pointpillars the loss functions used are the same as in SECOND [78] which consists of calculating the localization loss (equation 4.1) and object classification loss (equation 4.2):

$$\mathcal{L}_{loc} = \sum_{b \in (x,y,z,w,l,h,\theta)} SmoothL1(\Delta b) \quad (4.1)$$

where Δb represents the difference between the ground truth boxes and anchors for the values (x,y,z,w,l,h,θ) and SmoothL1 [9] for box regression,

$$\mathcal{L}_{cls} = -\alpha_a (1 - p^a)^\gamma \log p^a \quad (4.2)$$

where p^a is the class probability of an anchor, α is set to 0.25 and γ is set to 2.

A softmax classification loss [78], $\mathcal{L}_{dir}$ is used to compute the heading loss. Therefore the

total loss is formulated as:

$$\mathcal{L} = \frac{1}{N_{pos}}(\beta_{loc}\mathcal{L}_{loc} + \beta_{cls}\mathcal{L}_{cls} + \beta_{dir}\mathcal{L}_{dir}) \quad (4.3)$$

where N_{pos} is the number of positive anchors and $\beta_{loc} = 2$, $\beta_{cls} = 1$ and $\beta_{dir} = 0.2$.

In the Graph R-CNN model, the loss functions used are the classification loss (equation 4.4) and the regression loss (equation 4.5):

$$\mathcal{L}_{cls} = \frac{1}{B} \sum_{i=1}^B -I_i \log(\hat{I}_i) - (1 - I_i) \log(1 - \hat{I}_i) \quad (4.4)$$

where B is the number of sampled region proposals, I_i is the confidence score target, $\hat{I}_i$ is the predicted confidence score and i represents the i -th proposal box,

$$\mathcal{L}_{reg} = \frac{1}{B_+} \sum_{i=1}^{B_+} L1(o_i - t_i) \quad (4.5)$$

where B is the number of positive sampled region proposals, $L1$ is a loss function to minimize the error, o_i is the output of the model's regression and t_i is the regression target for the center, size and orientation of the box prediction.

Therefore the total loss is formulated as:

$$\mathcal{L} = \mathcal{L}_{cls} + \alpha \mathcal{L}_{reg} \quad (4.6)$$

where α is set to 1.

The GD-MAE Reconstruction Target computes an l2 Chamfer Distance to obtain the reconstruction loss:

$$\mathcal{L}_{\mathcal{CD}} = ChamferDistance(\hat{P}_{mask}, P_{mask}) \quad (4.7)$$

where $\hat{P}_{mask}$ is the predicted point cloud reconstruction and P_{mask} is the K points sampled point clouds.

4.6.5 Experiments

The analysis in this work focuses on the influence that hardware has on the inference speeds of detection models. First, the training process of the three models is evaluated, by comparing the evolution of the training loss and the overall mAP for the car, pedestrian and cyclist classes.

After training the models a comparison is made between the results obtained and the results achieved in the literature by these models so that it is possible to assess not only the veracity of

these statements but also the quality of the implemented code in use. In addition, we also put them up against the best ranked models in the KITTI benchmark for each of the three classes under study, to have a better understanding of how good the models are performing in relation to state-of-the-art 3D object detection models. Since the KITTI dataset was used to train the models, we decided to compare the same metrics used by the KITTI benchmark, namely 3D Bounding Box Detection, Bird’s Eye View Detection, 2D Bounding Box Detection and AOS, for the most important classes for autonomous driving, namely Car, Pedestrian and Cyclist. The results presented also take into account the three levels of difficulty (easy, moderate and hard) and use 40 recall positions to evaluate. Furthermore, according to the KITTI benchmark, a prediction is considered as true positive if the overlap of the 3D bounding box is 70% for the Car class and 50% for the Pedestrians and Cyclists classes.

With the models now trained and evaluated in terms of mAP, we arrive at the most important part of our study, namely how many different factors affect their inference speeds. To this end, three important factors will be analysed, namely the increase in batch sizes, changes in hardware used for inference, such as the GPU, and data parallelism by leveraging of multi-GPU environments. For each GPU, we will compare the inference times and memory consumption for the models trained and tested with batch sizes [2, 4, 8, 16, 32, 64] . Then we evaluate the effect that a multi-GPU environment with 2, 3 or 4 GPUs has on the inference speed and the number of samples that can be inferred per second. Finally, a comparison is made between these speeds and the inference speeds of the best ranked models in the KITTI benchmark. It should be noted that in the analysis of model training and precision, we will use our models trained with a batch size of 8 and tested with a test batch size of 8 in order to maintain consistency between our results and the established benchmark methodologies.

4.7 Summary

This chapter has provided a better understanding of the reasons for selecting the KITTI dataset and the criteria that led us to select the PointPillars, GD-MAE and Graph R-CNN models. We then provide a more detailed description of the characteristics of the dataset, including details of the different sensors used, the types of data provided, the routes taken, the description of the annotations and their respective distributions. We also provide a detailed description of the different models, including their architecture and components, so that there is a clearer understanding of the different designs and potential strengths and weaknesses of each model. Finally, we also learn more about the experimental setups, including equipment, data used from the dataset, configurations for training the models, loss functions for each model, as well as a greater context for the set of experiments and analyses that we will conduct. We feel it is important to provide a detailed description of all these aspects, as all these details will have an impact on the final performance of the models, as well as ensuring that we have all the necessary information to critically interpret the experimental results that we will obtain in the following chapter.

Chapter 5

Results and Discussion

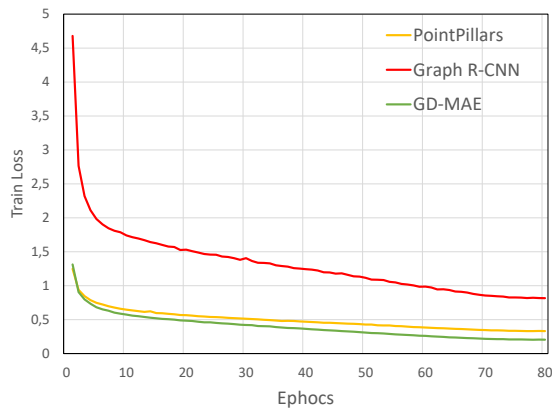
In this chapter, we present an overview of our research results. We begin with an exploration of the training process of our models and a comparison of the performance between our reproduced models, those of the original authors, and state-of-the-art models. We then assess the real-time capabilities of our models through an analysis of inference speed and memory consumption on different GPU configurations, including the impact of data parallelism in multi-GPU environments. We also evaluate the balance between AP and inference speed, and make comparisons with selected state-of-the-art models in terms of their inference speed.

5.1 Training models

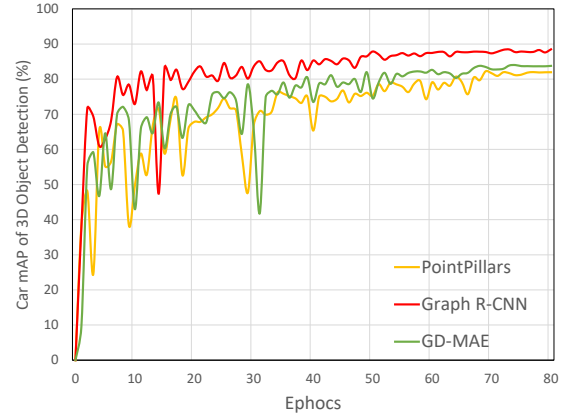
The three models were trained for eighty epochs for batch sizes [2, 4, 8, 16, 32, 64]. The training process took a total of 10.15 hours for the Pointpillars, 25.06 hours for the GD-MAE and 32.63 hours for the Graph R-CNN. Having verified that the evolution of the training process, as well as the total mAP, showed minimal changes between the different batch sizes, we decided to analyse in more detail the results obtained with the model trained with a batch size 8, in order to facilitate future comparisons with the results obtained by its authors, who also used a training batch size 8. For the pedestrian and cyclist classes, only the PointPillars and GD-MAE models are compared because the Graph R-CNN model has been implemented only for the car class.

The representation of the loss function during the training of the three models is present in Figure 5.1a, where it is shown that in the first ten epochs the train loss is reduced by more than half, after which it decreases linearly until it stabilizes in the last ten epochs, where in epoch 80 the loss is 0.3 for PointPillars, 0.2 for GD-MAE and 0.8 for Graph R-CNN.

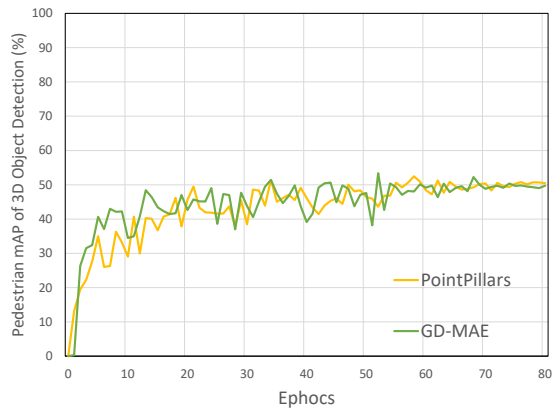
The evolution of the global mAP of the 3D object detection task on the Car class is represented by the graph in Figure 5.1b which shows that in the first 30 epochs the mAP grows considerably with a noisy progression, where this growth is more pronounced in the Graph R-CNN model. In the remaining training process the noise is minimized and the curve ends up stabilizing for the last 20 epochs, ending up with a global mAP of 81% for PointPillars, 83% for GD-MAE and



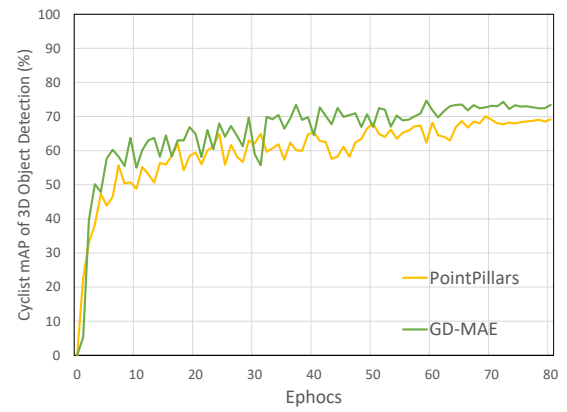
(a) Evolution of the training loss throughout the number of epochs for PointPillars, Graph R-CNN and GD-MAE models.



(b) Overall mAP of the 3D Object Detection on the Car class for PointPillars, Graph R-CNN and GD-MAE models



(c) Overall mAP of the 3D Object Detection on the Pedestrian class for PointPillars and GD-MAE models



(d) Overall mAP of the 3D Object Detection on the Cyclist class for PointPillars and GD-MAE models

Figure 5.1: Representation of the Train Loss (a) and Overall mAP of the three models on the 3D Object Detetion of the Car (b), Pedestrian (c) and Cyclist (d) classes

88% for Graph R-CNN.

For the Pedestrian class we have the graph in Figure 5.1c where there is a rapid increase in mAP in the first 20 epochs, which ends up stabilizing in the remaining epochs while the noise only becomes minimal for the last 30 epochs, ending with a global mAP of 50% for PointPillars and 49% for GD-MAE. At the end, we have the graph for the cyclist class in Figure 5.1d, which also shows a high growth in the first 20 epochs, but after that it grows linearly until it stabilizes, as well as the noise, in the last ten epochs, ending up with a global mAP of 69% for PointPillars and 73% for GD-MAE.

5.2 Comparing models

Once the models have been trained, the results obtained are compared with the results claimed by their authors, in order to assess not only the veracity of these claims, but also the quality of the implemented code used. In addition, the models are compared with the top-ranked models in the KITTI benchmark for each of the three classes studied, in order to gain a better understanding of how well the models perform against state-of-the-art 3D object detection models.

In Tables 5.1, 5.2 and 5.3 we present a performance comparison of the models for the four KITTI benchmark tasks. The notation is as follows:

- **AP_3D(%)**: Average Precision for the 3D Bounding Box Detection task.
- **AP_BEV(%)**: Average Precision for the BEV Detection task.
- **AP_2D(%)**: Average Precision for the 2D Bounding Box Detection task.
- **AP_AOS(%)**: Average Precision for the AOS task.

For each task, we assess the model's performance under three difficulty levels:

- **Easy**: Only considers ground truths with bounding boxes larger than 40 pixels, fully visible occlusion level and a truncation of up to 15%.
- **Moderate**: In this difficulty level, bounding boxes can be at least 25 pixels in size, occlusion level up to partially occluded and a maximum truncation of 30%.
- **Hard**: Also includes cases with up to the difficult to see occlusion level and a maximum truncation of 50%.

Each trained model is represented in the "Orig." row with the values obtained by the original authors and in the "Rep." row with the values reproduced by our work. Values underlined in green indicate the results obtained by the best-ranked model in the KITTI benchmark for the respective class.

Table 5.1: AP Results for KITTI Car Class Benchmark Tasks.

		AP_3D(%)			AP_BEV(%)			AP_2D(%)			AP_AOS(%)		
		easy	mod.	hard	easy	mod.	hard	easy	mod.	hard	easy	mod.	hard
VirConv-S	Best	92,48	87,20	82,45	95,99	93,52	90,38	98,00	97,27	94,53	96,99	96,46	93,74
PointPillars	Orig.	82,58	74,31	68,99	90,07	86,56	82,81	94,00	91,19	88,17	93,84	90,70	87,47
	Rep.	84,65	74,45	68,93	90,04	85,75	79,79	90,60	88,08	80,78	90,49	87,49	80,06
GD-MAE	Orig.	88,71	78,25	77,05	89,89	87,91	86,56	97,11	90,11	89,23	97,00	89,86	88,87
	Rep.	88,30	78,00	76,75	89,77	88,20	86,95	90,81	90,11	89,28	90,74	89,89	88,97
Graph R-CNN	Orig.	89,81	85,89	79,35	90,57	89,21	88,51	99,32	90,30	89,94	99,30	90,24	89,83
	Rep.	89,83	86,08	79,30	90,53	89,08	88,44	98,74	90,30	89,92	97,98	90,22	89,79

Table 5.1 shows the AP results of the models for the Car class. For Pointpillars, we can see that the performance of the model is a little distant from the AP obtained by its authors. Regarding the results obtained by the authors in the 3D Bounding Box task, our reproduced model has a higher AP of up to 2%, for the Bird’s Eye View task the AP is up to 3% lower and for the 2D Bounding Box & Average Orientation Similarity tasks the AP can be up to 8% lower. In the other two reproduced models, we can see that the performance remained within the AP parameters originally obtained by the authors, where the GD-MAE model only presents lower AP up to 7% in the tasks of 2D Bounding Box & Average Orientation Similarity in the easy difficulty. With all this we can consider that of the three, the Graph R-CNN at the level of AP is the best model, but even so it continues to have lower AP than the best ranking model for the car class from the KITTI benchmark (VirConv-S [74]) where as for the 3D Bounding Box & Average Orientation Similarity tasks it is lower by 3% and for the Bird’s Eye View & 2D Bounding Box tasks it is lower by 4%.

Table 5.2: AP Results for KITTI Pedestrian Class Benchmark Tasks.

		AP_3D(%)			AP_BEV(%)			AP_2D(%)			AP_AOS(%)		
		easy	mod.	hard	easy	mod.	hard	easy	mod.	hard	easy	mod.	hard
CasA++	Best	56,33	49,29	46,7	60,14	53,84	51,35	74,41	64,94	62,35	71,78	61,59	58,71
PointPillars	Orig.	51,46	47,94	43,80	59,17	54,35	50,50	64,62	61,42	57,60	49,38	46,73	43,84
	Rep.	47,73	43,12	39,46	58,92	54,09	50,06	66,91	63,11	58,83	47,38	43,99	40,91
GD-MAE	Orig.	53,76	48,87	46,74	61,38	56,53	51,20	67,60	65,65	61,08	59,32	57,16	52,64
	Rep.	55,42	50,03	47,10	61,67	56,30	50,99	70,10	66,84	61,73	61,52	57,75	52,88

The AP results of the models for the Pedestrian class are shown in Table 5.2. Once again, for Pointpillars, we can see that the performance of the reproduced model differs from that obtained by the authors. Our reproduced model has a AP lower by up to 4% for the 3D Bounding Box task and for the Average Orientation Similarity task, the AP is lower by up to 3% while for the 2D task Bounding Box the AP is up to 2% higher and for the Bird’s Eye View task the AP remained within expectations. In the case of the GD-MAE, we do not have major differences between the results reproduced and those obtained by the authors, where there is only a greater AP of up to 2% for the 3D Bounding Box task and a greater AP of up to 3% for the easy difficulty in 2D Bounding Box & Average Orientation Similarity tasks. Of these two models, the GD-MAE can be considered the best in terms of AP and when compared to the best model for the pedestrian class (CasA++ [72]), contrary to expectations, it has slightly better results in two tasks, where as the 3D task Bounding Box reaches a higher AP of up to 1% and the Bird’s Eye View task a higher AP of almost 2%. Meanwhile in the 2D Bounding Box task it has a lower AP of up to 4% and for the Average Orientation task Similarity comes down to a lower AP of up to 10%.

For the Cyclist class, Table 5.3 shows the AP results of the models. For the reproduced models, both PointPillars and GD-MAE, their performance remained the same in relation to that obtained by the authors. In the case of PointPillars, for all tasks, differences in AP do not even reach 1%, except only in the Average Orientation Similarity task, where the reproduced AP is up to 2% higher than that of the authors. For the case of GD-MAE the same behaviour is observed for all tasks and at most the AP is 2% higher than that of the authors, where the

Table 5.3: AP Results for KITTI Cyclist Class Benchmark Tasks.

		AP_3D(%)			AP_BEV(%)			AP_2D(%)			AP_AOS(%)		
		Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
TED	Best	88,82	74,12	66,84	89,54	76,95	70,31	92,6	84,36	78,43	92,46	84,08	78,07
PointPillars	Orig.	81,87	63,66	60,91	84,43	67,14	63,74	86,26	73,08	70,17	85,04	69,10	66,28
	Rep.	81,95	64,43	61,73	82,72	67,16	62,79	85,63	73,35	70,63	85,13	71,58	68,68
GD-MAE	Orig.	83,81	65,29	62,71	90,95	71,42	65,57	91,87	74,76	68,24	90,64	70,63	64,57
	Rep.	84,89	66,11	63,72	87,15	72,67	66,24	92,84	74,98	68,55	92,02	72,53	66,54

only case in which this does not happen is in the task Bird’s Eye View on the easy difficulty, where the AP difference reaches 3%. For the pedestrian class we can say that the GD-MAE has the best AP between the two but when it is put up against the best model in the KITTI benchmark (TED [73]) it continues to have worse results, where the 3D Bounding Box & Average Orientation Similarity tasks have a lower AP of up to 8%, the Bird’s Eye View task has a lower AP of up to 4% and the 2D Bounding Box task has a lower AP of up to 10%.

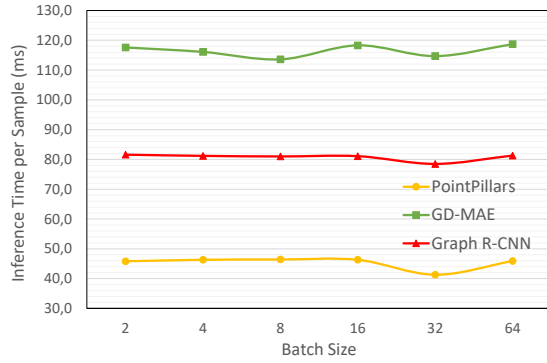
5.3 Evaluate inference times

With the models now trained and evaluated in terms of AP, we arrive at the most important part of our study, namely how much different factors affect their inference speeds. To this end, three important factors were analysed, namely the increase in batch sizes both in training and in testing, changes in hardware used for inference, such as the GPU, and data parallelism by leveraging of multi-GPU environments.

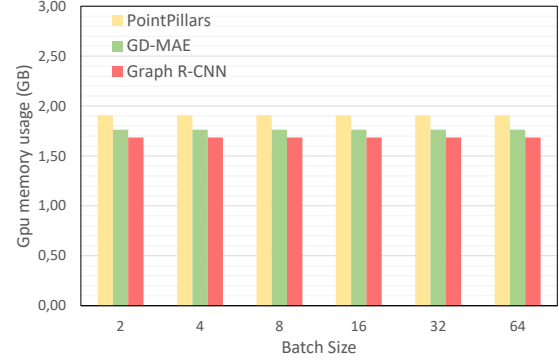
5.3.1 Exploring the impact of training batch size on inference speed

In this first experiment we will analyse the performance of models trained with different batch sizes for real-time scenarios. To achieve this, all tests will be done with a batch size of 1, that is, the models will infer one image at a time, simulating the way in which the sensors capture and provide data in real-time. In this analysis the focus will be on the effects that training with different batch sizes can have on the inference speed of the models.

For the Tesla V100 SMX2 GPU machine, Figure 5.2a shows the inference times of the models trained with the different batch sizes and Figure 5.2b shows the corresponding memory consumption. In terms of inference, we were able to verify that the batch size used for training has no effect on the speed of inference, nor does it have any effect on the memory consumption. The GD-MAE model has an average inference speed of 116.5 ms, with a standard deviation of 1.88 ms, and consumes 1.76 GB of GPU memory. The PointPillars model has an average inference speed of 45.3 ms, with a standard deviation of 1.82 ms, and consumes 1.91 GB of GPU memory. Finally, the Graph R-CNN model has an average inference speed of 80.8 ms, with a standard deviation of 1.04 ms, and consumes 1.68 GB of GPU memory.



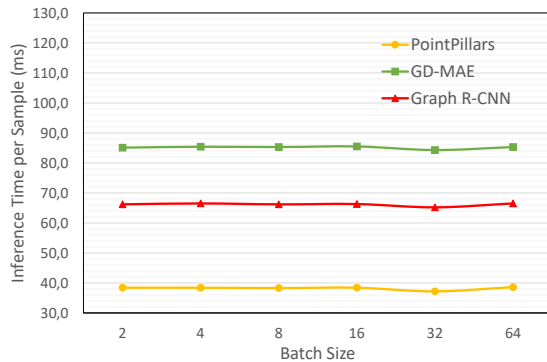
(a) Tesla V100-SXM2 GPU inference time per image of the PointPillars, GD-MAE and Graph R-CNN models trained with different batch sizes



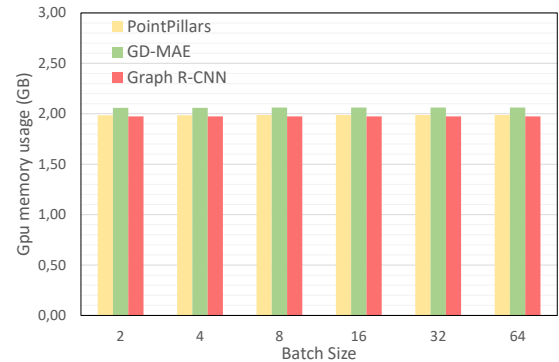
(b) Tesla V100-SXM2 GPU memory usage of the PointPillars, GD-MAE and Graph R-CNN models trained with different batch sizes

Figure 5.2: Inference Times and GPU memory usage of the trained models for different batch sizes on the Tesla V100-SXM2 GPU

For the machine with a GeForce RTX3090 GPU, we again have Figure 5.3a, which shows the inference times for the models trained with the different batch sizes, and Figure 5.3b, which shows the corresponding memory consumption. As expected, we can see that different batch sizes for model training have no effect on inference speed or memory consumption. For the GD-MAE model, the average inference speed is 85.2 ms, with a standard deviation of 0.40 ms, and consumes 2.06 GB of GPU memory. The PointPillars model has an average inference speed of 38.2 ms, with a standard deviation of 0.46 ms, and consumes 1.99 GB of GPU memory. Finally, the Graph R-CNN model has an average inference speed of 66.2 ms, with a standard deviation of 0.44 ms, and consumes 1.97 GB of GPU memory.



(a) GeForce RTX3090 GPU inference time per image of the PointPillars, GD-MAE and Graph R-CNN models trained with different batch sizes



(b) GeForce RTX3090 GPU memory usage of the PointPillars, GD-MAE and Graph R-CNN models trained with different batch sizes

Figure 5.3: Inference Times and GPU memory usage of the trained models for different batch sizes on the GeForce RTX3090 GPU

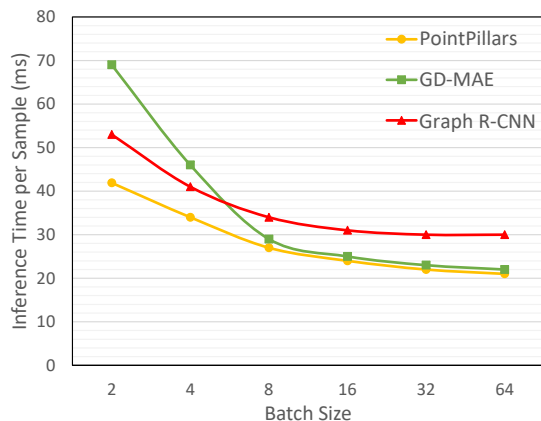
From this analysis we were able to conclude that the use of different batch sizes for training is only relevant during training and has no effect on inference speed or memory consumption. In

terms of models, we were able to verify that PointPillars is the fastest model, followed by Graph R-CNN and GD-MAE as the slowest. In addition to the different models, it is also possible to verify that changes in hardware also affect their inference speed. For GD-MAE the inference speed went from 116.5ms on the Tesla GPU to 85.2ms on the GeForce GPU, for PointPillars the inference speed went from 45.3ms on the Tesla GPU to 38.2ms on the GeForce GPU and for Graph R-CNN the inference speed went from 80.8ms on the Tesla GPU to 66.2ms on the GeForce GPU. From this we can also conclude that of the two GPUs used, the Geforce GPU is better for model inference with a 26.8% speed gain in GD-MAE, 15.7% speed gain in PointPillars and 18.1% speed gain in Graph R-CNN, all compared to the Tesla GPU.

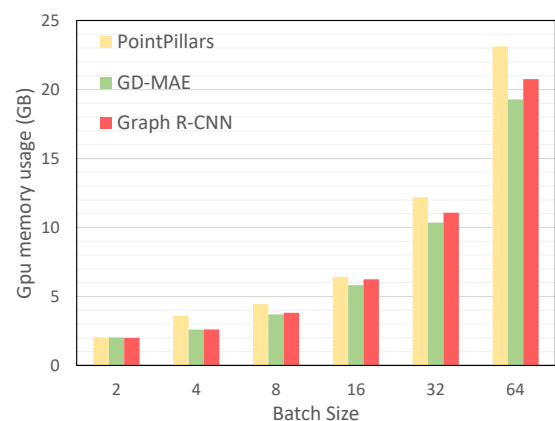
5.3.2 Exploring test batch size variations in inference speed

In this second experiment we will analyse the effect of different test batch sizes on inference speed. As in the previous experiment we found that variations in training batch size did not affect the speed of the models, we will only use the model trained with batch size 8.

For the Tesla V100 SMX2 GPU machine, Figure 5.4a shows the model inference times for the different test batch sizes. Up to batch size 8, there is a sharp reduction in inference time, particularly pronounced for the GD-MAE model, which was reduced by more than half. After that, at batch size 16, this reduction becomes smaller, with all three models showing a reduction in inference time of 3ms. At batch sizes 32 and 64 the reduction is only 1ms each. Thus for PointPillars the inference time decreased from 42ms at batch size 2 to 21ms at batch size 64. For GD-MAE the inference time decreased from 69ms at batch size 2 to 22ms at batch size 64. And for Graph R-CNN, the inference time decreased from 53ms with a batch size of 2 to 30ms with a batch size of 64.



(a) Tesla V100-SXM2 GPU inference time per image of the PointPillars, GD-MAE and Graph R-CNN models for different batch sizes

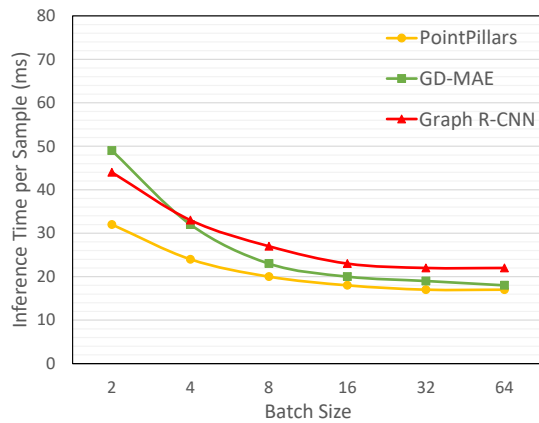


(b) Tesla V100-SXM2 GPU memory usage of the PointPillars, GD-MAE and Graph R-CNN models for different batch sizes

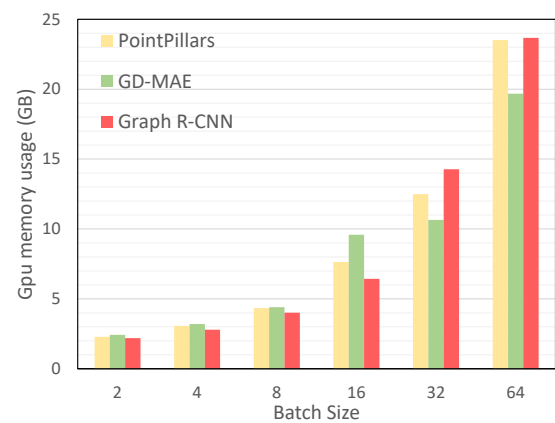
Figure 5.4: Inference Times and GPU memory usage of the trained models for different batch sizes on the Tesla V100-SXM2 GPU

In addition to inference time, we also analysed the impact that different batch sizes have on the GPU memory usage, as shown in Figure 5.4b. It is evident that for all three models, up to a batch size 16, there is a slight increase in the amount of memory required. The memory usage goes from 2GB for a batch size 2, to around 6GB for a batch size 16. However, at batch size 32 the memory usage doubles to around 11 GB, just like in the batch size 64 which again doubles to around 22GB. The behaviour of the three models is more or less the same but PointPillars tend to use a little more memory, followed by Graph R-CNN and then GD-MAE.

From these graphical representations we can see that up to a batch size 16 there is a considerable reduction in the inference time in exchange for a small increase in the memory used, but after that, for batch sizes 32 and 64, the inference time is only reduced by 1 ms, while the memory usage doubles. With this, we can surmise that, for the Tesla V100-SMX2, a batch size 16 is what brings the greatest gain in inference speed relative to the increase in memory required.



(a) GeForce RTX3090 GPU inference time per image of the PointPillars, GD-MAE and Graph R-CNN models for different batch sizes



(b) GeForce RTX3090 GPU memory usage of the PointPillars, GD-MAE and Graph R-CNN models for different batch sizes

Figure 5.5: Inference Times and GPU memory usage of the trained models for different batch sizes on the GeForce RTX3090 GPU

To assess the impact of hardware changes, namely the GPU on the inference speed, we run the same models and evaluate the effects of different test batch sizes but this time on the GeForce RTX3090 GPU. Information about the inference times for different batch sizes can be found in Figure 5.5a. Right from the start we noticed that the inference times using this GPU are lower, at least in a batch size 2. The PointPillars & Graph R-CNN models got 10ms faster and the GD-MAE model showed a 20ms improvement. Once again, up to a batch size 8 there is a high reduction in inference time, being more pronounced in GD-MAE where it was reduced in half. After that, at a batch size 16, there is again a 3ms decrease in the inference time across the three models. Finally between the batch sizes 32 and 64 the inference time barely decreases by 1ms. Despite realizing that this GPU is faster, we noted that the impact of increasing the batch size is not as pronounced as on the Tesla GPU. For PointPillars the inference time decreased from 32ms at a batch size from 2 to 17ms at a batch size 64. For GD-MAE the inference time decreased from 49ms at a batch size 2 to 18ms at a batch size 64. Lastly, for Graph R-CNN the

inference time decreased from 44ms at a batch size 2 to 22ms at a batch size 64.

Figure 5.5b displays the impact of different batch sizes on the GPU memory usage. As with the other GPU, up to a batch size 8, the memory used across the three models remained the same between them, going from 2GB in a batch size 2 to 4GB in a batch size 8. For a batch size 16, the PointPillars & Graph R-CNN have an increase of only 2.5GB while in the GD-MAE this value more than doubles reaching 9GB of memory usage. For the batch size 32 the opposite happens where the GD-MAE only increased by 1GB while for PointPillars & Graph R-CNN the memory usage raised by 5GB and 8GB, respectively. Finally, at batch size 64, the memory usage was 23GB for PointPillars & Graph R-CNN and 19GB for GD-MAE. Interestingly, in this GPU we observed that the memory usage does not differ a lot from the memory used between both models and GPUs. Even with the GD-MAE, if we ignore the unexpected increase in memory usage in the batch size 16, it holds true.

With this information, we can see that the GeForce GPU, like the Tesla GPU, has the most significant reduction in inference time at batch size 16, with a small increase in memory required. From all this we can conclude that although the reduction in inference time is not as pronounced as for the Tesla GPU, the end result is still lower and therefore faster. Comparing the two, we keep the previous experiment conclusion that the GeForce GPU is better for model inference.

5.3.3 Exploring inference speed in multi-GPU settings

Finally, in this third experiment, taking advantage of the fact that one of the machines used contains four Tesla V100-SXM2 GPUs, we will analyse how multi-GPU environments affect model inference speeds when data parallelism is employed.

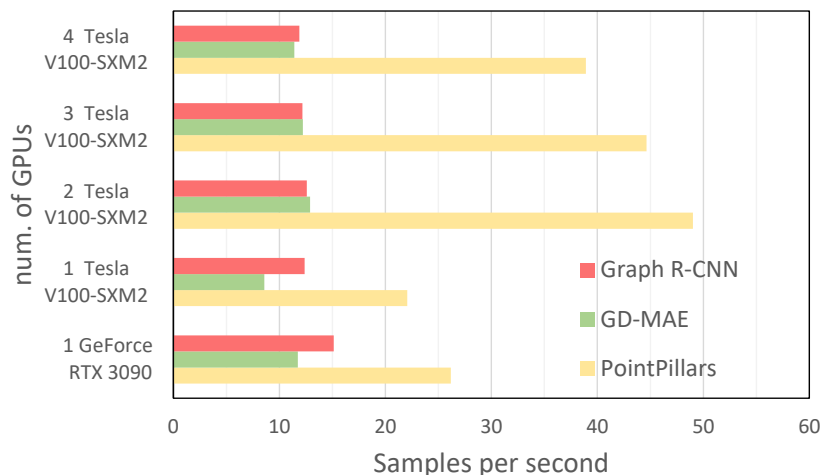


Figure 5.6: Inference Speed of the trained models with a test batch size 1.

Figure 5.6 shows the number of samples per second that each model can infer for each possible number of GPUs, with a batch size equal to 1, with the aim of evaluating real-time performance. We can see that when using only one GPU, the GeForce GPU, it can infer 4 more images per second with PointPillars and 3 more images per second with GD-MAE and Graph R-CNN than

the Tesla GPU. If we increase the number of Tesla GPUs in use to two, we can see that there is a change in the number of samples that the models can infer per second, where PointPillars is now able to infer 49 images, GD-MAE is now able to infer 13 images, while Graph R-CNN shows no change and continues to infer 12 images. If we then increase the number of GPUs to three, there is a reduction in the number of samples, with PointPillars inferring 44 images per second and GD-MAE inferring 12 images per second. Finally, if we increase the number of GPUs again to four, there is a reduction in the number of samples inferred per second, with PointPillars starting to infer 38 images and GD-MAE starting to infer 11 images. In the case of Graph R-CNN, none of the increases in the number of GPUs working in parallel resulted in any change in the number of images inferred per second.

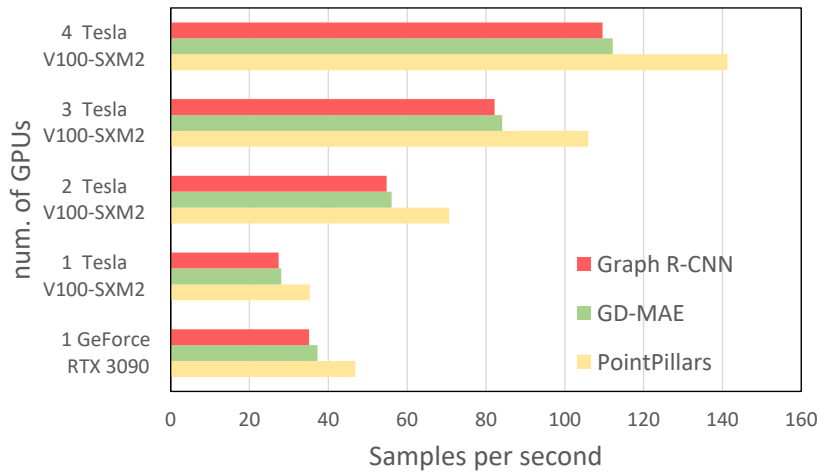


Figure 5.7: Inference Speed of the trained models with a test batch size 8.

Figure 5.7 shows the number of samples per second that the models are capable of inferring for each possible number of GPUs but this time for a batch size equal to 8, to evaluate how the models behave when given a group of images at a time. Once again, in the two bottom lines, it is evident that a GeForce RTX 3090 GPU is better than a Tesla V100-SXM2. The GeForce GPU, per second, can infer 11 more images with PointPillars, 9 more images with GD-MAE and 8 more images with Graph R-CNN. However, in the next line where two Tesla GPUs are used, this changes. The number of samples per second that the three models can infer almost doubles, and when compared to a GeForce GPU it infers 20 more images with the GD-MAE and Graph R-CNN, and 24 more images with PointPillars. This behaviour continues for the remaining lines where as for three and four Tesla GPUs this value triples and quadruples, respectively, in relation to just one Tesla GPU. In this analysis of multi-GPU environments, we focused on evaluating the number of samples inferred per second because it directly impacts the real-time performances of the models.

For a more comprehensive understanding of how hardware affects inference speeds, Figure 5.8 shows a comparison between inference times and the mAP for the car class, considering both different and parallelized GPUs. From this analysis it is possible to deduce three key insights. First, hardware changes alone have no effect on the prediction quality of the models. Second, employing multiple GPUs can be a good alternative to compensate for the use of lower-quality

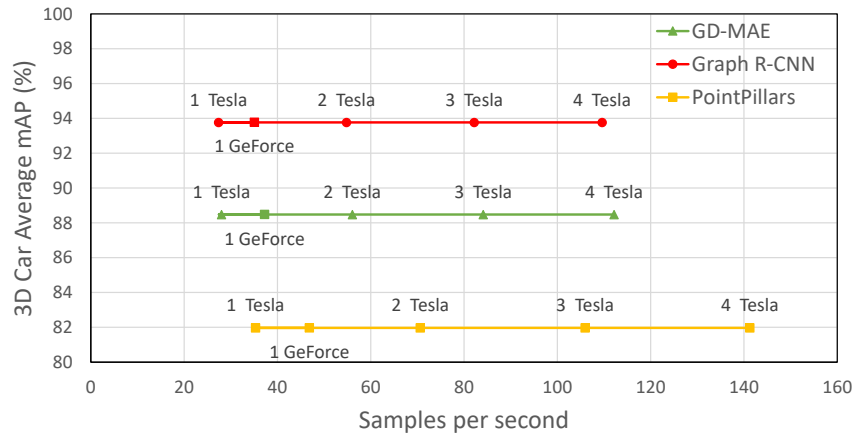


Figure 5.8: Unified View: GPU Setup, Inference speed, and Car AP_{3D}(%) analysis (with test batch size 8).

GPUs, and potentially achieve better performances than a single higher-quality GPU. Lastly, it might be worth opting for models with a slightly lower precision in exchange for a higher inference speed, especially for real-time tasks such as autonomous driving.

5.3.4 Summary

As a way of summarizing the important results obtained, in Figure 5.9 we analyse the inference speeds and precisions of the trained models in relation to the benchmark performances of the best ranked models that we defined in the previous section. While this comparison may not be fair due to the variation in hardware configurations, this analysis is intended to provide a preliminary understanding of our trained models performances in a different context.

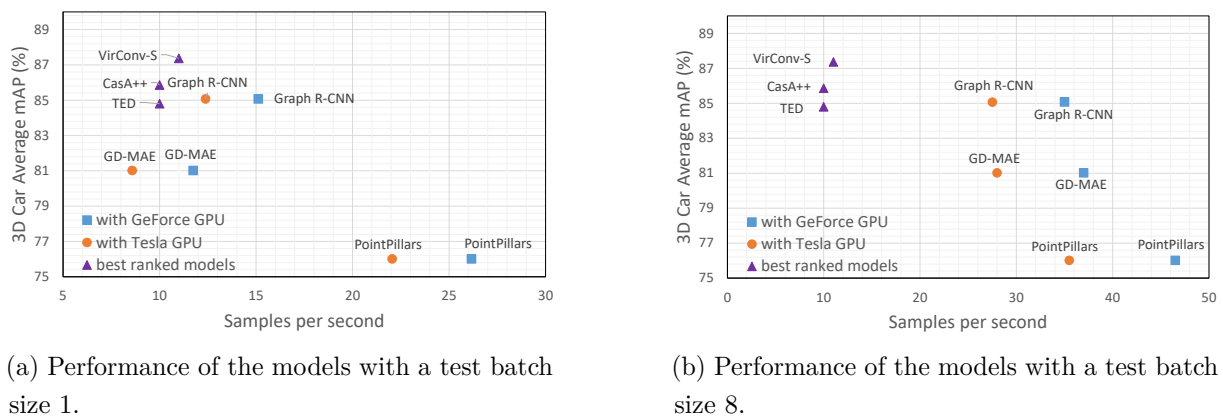


Figure 5.9: Performance evaluation of the trained models versus best-ranked models

As we can see in Figure 5.9a, where we use a test batch size 1 to evaluate the performance of the models in real-time scenarios, we can see that at the precision level only Graph R-CNN gives results similar to the best ranked models, with GD-MAE being about 5% less precise and PointPillars being up to 10% less precise. Now, if we compare the inference speed, we can see

that, per second, Graph R-CNN infers between 2 and 5 more samples, PointPillars infers between 12 and 16 more samples and GD-MAE infers between less two and two more images, with respect to the best ranked models. In Figure 5.9b, where we analyse the test batch size 8 to evaluate the performance of the models in cases where the model receives data in batches of images, we have no changes at the level of precision, but in the speed of inference we already have changes where, per second, Graph R-CNN starts to infer between 17 and 25 more samples, PointPillars starts to infer between 25 and 37 more samples and GD-MAE starts to infer between 18 and 27 more samples, with respect to the best ranked models.

The main conclusions drawn from the results of the experiments will now be listed:

- **Exploring the impact of training batch size on inference speed:**

- Different batch sizes for training are only relevant during training, where they have no effect on inference speed or memory consumption during testing.
- PointPillars leads as the fastest model, with Graph R-CNN securing the second position, and GD-MAE trailing behind in third place
- For real-time scenarios, the aspect with the greatest impact on inference speed is the architecture of the models, followed to a lesser extent by the hardware, namely the GPU.
- Of the two GPUs used, the Geforce GPU has the edge in terms of speed of inference, but it has a slightly higher memory consumption, albeit at a negligible level that does not warrant any significant concern.

- **Exploring test batch size variations in inference speed:**

- Increasing the test batch size lead to a reduction of the inference time per sample but with a increase in the memory consumption.
- A test batch size 16 is what brings the greatest gain in inference speed relative to the increase in memory required
- For each increase in test batch size, the reduction in inference time becomes smaller and smaller, with only a 1ms reduction in inference time from batch size 32 to 64, while memory usage doubles.
- Increasing the test batch size has more of an effect on GD-MAE. With this increase, it goes from being the slowest to having inference speeds comparable to PointPillars.
- By increasing the test batch size the reduction in inference time is more pronounced in the Tesla GPU but in absolute values the GeForce GPU continues to be better.

- **Exploring inference speed in multi-GPU settings:**

- For real-time scenarios, when moving from a one to a two Tesla GPU environment, per second, PointPillars doubles the number of samples inferred, GD-MAE inferes only four more samples and Graph R-CNN shows no change.

- For real time scenarios, the use of 3 or 4 GPU environments does not lead to an increase in the number of samples inferred per second; on the contrary, there is a reduction in this number where, per second, PointPillars infers up to 10 fewer samples, GD-MAE infers only 2 fewer samples and Graph R-CNN shows no change.
- The impact of the use of multi-GPU environments, at least for real-time scenarios, is dependent on the architecture of the models.
- When the models receive one set of samples at a time, the behaviour is different, with the number of samples inferred per second doubling in all three models for a test batch size of 8, from one GPU to two GPU environments.
- For applications that need to perform 3D object detection on batches of samples at a time, this is where data parallelism brings the greatest benefits to these systems.

5.4 Discussion

From all this analysis, it is clear to all that the main challenge for autonomous driving systems is to balance the increasing complexity of architectures with the practical constraints imposed by the variety and speed of sensors, such as LiDAR, for sensing the environment, and the time constraints for safety-critical decisions that require rapid decision-making.

By analysing the KITTI dataset, as well as others such as the nuScenes dataset, the Waymo Open dataset and the Pandaset dataset, we found that in most cases LiDAR sensors are used with a capture rate of 10 FPS, which means that the capture of each sample takes about 100 ms. Adding this to the inference time of the models, the perception phase alone takes between 150 and 200 ms to complete the detection. Knowing that up to 1 second is considered a safe reaction time for a human being, extrapolating to autonomous driving systems leaves only 800 ms for its remaining 4 components. This fact further emphasises the importance of continuing work on the development of models with a focus on reducing inference time, not only for the perception phase, but also for the localisation and mapping, prediction, planning and control phases, so that autonomous driving systems as a whole can guarantee reliability and safety.

The use of multi-GPUs has a very positive effect in increasing the number of samples that the models can infer per second, even if only in cases where data is received in batches. For example, for the planning component models, such as overtaking predictors, where a set of previous samples is used to determine future trajectories, the data arrives in a single batch. However, it is important to note that even with this improvement in inference speed, four Tesla V100-SXM2 GPUs are up to eight times more expensive than a single NVIDIA GeForce RTX 3090 GPU, while the number of samples inferred per second increases by a maximum of four times. The choice of hardware for autonomous driving systems must therefore be based not only on performance, but also on cost.

Chapter 6

Conclusions

In conclusion, in this dissertation we delve into the field of 3D object detection in autonomous driving based on LiDAR, focusing on inference speed. The series of experiments conducted shed light on several critical aspects that contribute to the development of models capable of high precision while maintaining high inference speeds. Most notable in these analyses is the lack of established benchmarks or metrics that allow a standardised way to evaluate and compare the balance between precision and inference speed, which is one of the major challenges in this field.

Our results emphasise that the architectures of the models used play an important role in inference speed, where we could see a correlation between the increase in model complexity and the decrease in inference speed. PointPillars, with the least complexity of the three, emerges as the fastest model, while Graph R-CNN and GD-MAE have higher accuracy but lower inference speed due to their complexity. Furthermore, this study has shown that the choice of hardware, such as the GPU, has a significant impact on the speed of inference, with the GeForce GPU showing better results among those tested. When exploring data parallelism with multi-GPU environments, we found that, as expected, the models saw the greatest gain in inference speed when data was received in batches, but when we tested for real-time scenarios where data was received one at a time, we found that depending on the architecture, we either saw an increase, decrease or no change in the inference speed of the models. This highlights the need to implement architectures that are able to take advantage of data parallelism when the opportunity arises.

With this dissertation, we believe that this analysis has provided valuable insights that contribute to ongoing efforts in autonomous driving research. We identify some key factors that influence inference speed and highlight the need for standardised metrics and benchmarks that allow the balance between speed and precision to be assessed. We hope that this research can contribute to the development of evaluation metrics and more efficient and robust object detection models, which are crucial for the safety and reliability of autonomous vehicles.

6.1 Limitations

During the preparation of this dissertation, we encountered some limitations with respect to datasets and models for 3D object detection. The first limitation is that most of the models are implemented very fine-tuned for specific datasets, with the KITTI dataset being the predominant choice. This means that these implementations lose their adaptability to multiple data sources and datasets, limiting their usefulness in the real world. Fortunately, open source projects such as MMDetection3D and OpenMMLab [13] provide toolboxes and benchmarks that facilitate the adaptation of models to work with multiple datasets and different data sources. The other limitation is that even with data augmentation techniques that improve the robustness of the models, once trained, the models are very sensitive to changes in LiDAR sensors or datasets, leading to significant performance degradation. This highlights the need for researchers to develop models that can adapt to different sensor configurations and to explore new data augmentation techniques that improve their performance across different dataset and sensor variations.

6.2 Future Work

In terms of future work, it would be useful to conduct a more intensive analysis with a greater number and variety of hardware configurations, datasets and model architectures. It would also be interesting to evaluate the effect of other characteristics on inference speed, such as variations in deep learning frameworks, e.g. TensorFlow [2] and Pytorch [52], which provide different optimisation tools, and operating system changes, as they may present differences in resource management and system optimisations. With this analysis in place, the next step would be to create a set of benchmarks and metrics that would allow the best models for autonomous driving systems to be quantitatively compared and identified, taking into account not only the precision or inference speed of the model, but a balance between the two.

Appendix A

Datasets Description

A.1 The nuScenes Dataset

The team at Motional developed the nuScenes dataset and made it available free of charge to researchers and the academic community in order to solve the lack of multimodal datasets and aid the public research in autonomous driving. The driving scenarios present in this dataset were captured driving in Boston (Seaport and South Boston) and Singapore (One North, Holland Village and Queenstown) with different sensors including cameras, LiDAR, Radar, GPS and IMU. Thanks to its annotations and size it becomes a large benchmark capable of increasing the performance of object detection, tracking, prediction and localization tasks.[8]

To capture the data, two Renault Zoe cars were used, equipped with sensors placed around the car as in Figure A.1. It uses six Basler ace cameras (model acA1600-60gc) with a resolution of 2 megapixels, a frame rate of 60 FPS, 1/ 1.8" EV76C570 CMOS and a global and rolling shutter. Each camera uses a fixed Evetar lens with a focal length of 8mm, horizontal angle of view of 64.5° and vertical angle of view of 51° . LiDAR sensor (Velodyne HDL32E) with a capture frequency of 20Hz, 32 beams, an angular resolution between 0.08° and 0.33° , 2cm distance accuracy, a capture rate of approximately 1.39 million points per second, a 360° horizontal fov and vertical fov of 41.33° and a range of up to 100m. Five long range radar sensors are also used (Continental ARS 408-21) with a capture frequency of 13Hz, a distance accuracy of up to 40cm, a range of up to 250 m and a velocity accuracy of approximately 0.1km/h. For precise navigation and localization we have a GNSS aided inertial navigation system (Advanced Navigation Spatial) with a frequency of 1000Hz, a position accuracy of 20mm, heading accuracy of 0.2° and roll and pitch accuracy of 0.1° .

The data present in this dataset includes images, LiDAR, Radar and GPS/IMU. Images are unpacked in BGR format and then compressed to JPEG. To reduce processing and transmission bandwidth, an 1600x900 ROI is cropped from the original image resolution. To compensate for the vulnerability of GPS outages and to be able to accurately locate the vehicle, a detailed HD map was created in an offline step based on information from LiDAR and odometry, using

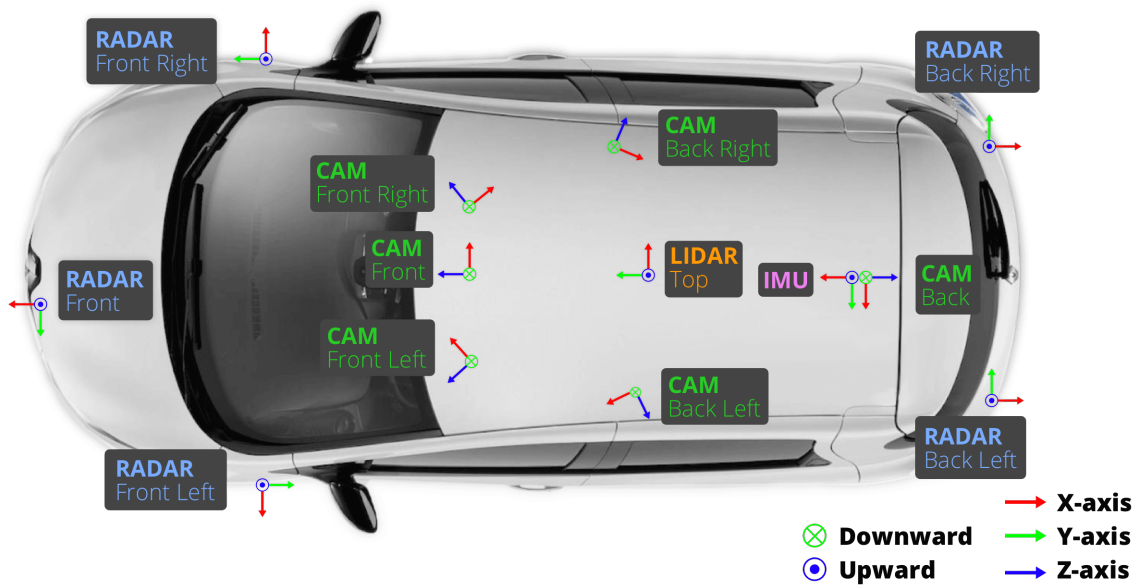


Figure A.1: Top View of the nuScenes Sensor Setup.

a Monte Carlo Localization scheme. In addition, information on speeds, accelerations, torque, steering angle and wheel speeds are also given. As the camera frame rates were reduced to run at 12 Hz and the LiDAR run at 20Hz, the 12 camera images are spread across the 20 LiDAR scans, so not all LiDAR scans have a corresponding camera frame. The Velodyne scans are stored as floating-point binaries where each point contains its coordinates (x,y,z) and an additional reflectivity value (r) . The number of points per scan is not constant but on average each frame roughly corresponds to 116 666 3D points.

After the data was captured, sample keyframes at 2Hz are selected and in each key frame all objects present that are covered by at least one LiDAR or radar points are annotated. Object classes include 23 classes between different types of vehicles, pedestrians and other objects. Each object is given a class with its semantic category, visibility, activity, pose and a 3D bounding box with the values of x, y, z , width, length, height and yaw angle.

A.2 The Waymo Open Dataset

In order to address the limitations of existing datasets in terms of scale, researchers at Google and Waymo have developed and publicly released one of the largest and most diverse dataset to date, the Waymo Open Dataset. Data was captured on a set of different driving scenarios in three different US cities, namely San Francisco, Phoenix, and Mountain View. Thanks to its substantial area and diversity of geographic coverage as well as its high number of objects annotated in each frame, it is considered one of the most complete datasets available for research on autonomous driving related tasks.^[64]

For data collection, five LiDAR sensors and five high-resolution pinhole cameras were used. The LiDAR sensors have a frame rate of 2 FPS, where the one on top of the vehicle has a vertical fov between -17.6° and $+2.4^\circ$ and a range of 75 m, and the remaining sensors that are located at the front, rear, side-left and side-right have a vertical fov between -90° and 30° and a range of 20 meters. For the 5 cameras used, all have a horizontal fov of approximately 25.2° where the front, front-left and front-right cameras have a resolution of 1920×1280 and the two remaining ones that are on the side-left and side-right have a resolution of 1920×1040 . In Figure A.2 we can see the positioning of the sensors in the vehicle used. It should be noted that little information is given about the sensors when compared to the information given by the other datasets present in this dissertation.

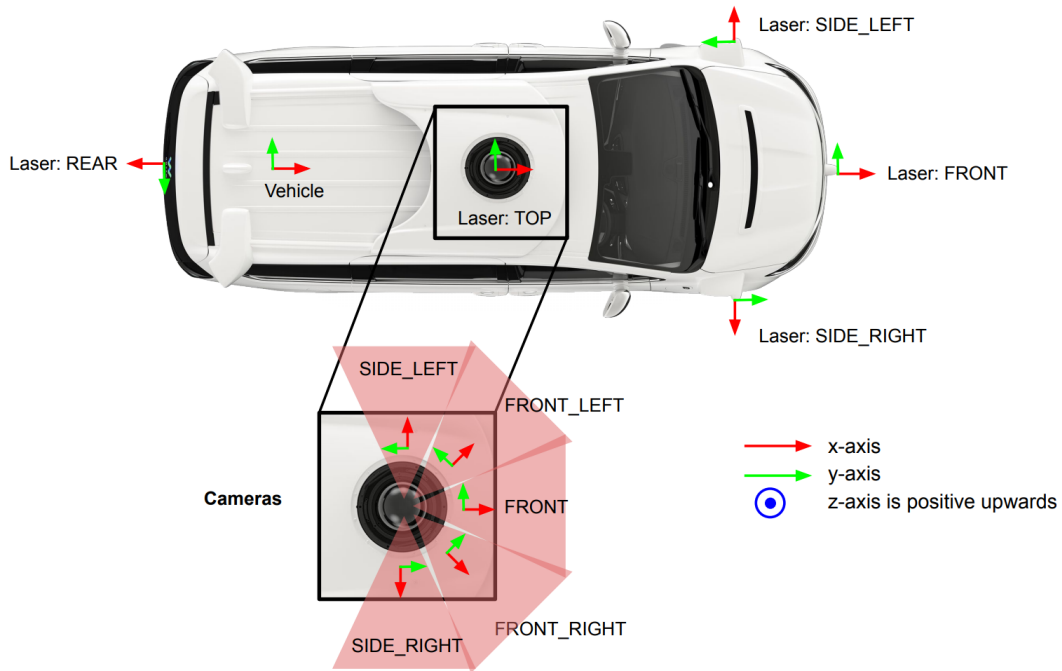


Figure A.2: Top View of the Waymo Open Dataset Sensor Setup.

The data provided by this dataset are the images and the LiDAR data. The images are compressed in JPEG format after being cropped and downsampled from the original camera data. LiDAR data is stored as range images where each pixel of the image corresponds to a LiDAR return. Each pixel in the image includes its range, intensity, elongation, no label zone and vehicle pose. Both data was captured with rolling shutter scanning from left to right. In addition to this information, we also have the coordinate system of the global frame that is defined before starting the movement, the vehicle frame that represents the position of the vehicle in relation to the global frame and the sensor frame that defines the position of each sensor in relation to the vehicle frame. For the images it has the coordinate system where x represents the width and y represents the height and the LiDAR data coordinate system is based on the Cartesian coordinate system.

Annotations for both camera images and LiDAR sensor data readings are provided. The object classes are "Vehicles", "Pedestrians", "Cyclists" and "Signs". For LiDAR sensor data

readings each object is associated with a class, a unique tracking ID and a 3D bounding box $(cx, cy, cz, l, w, h, \theta)$ where cx, cy, cz represent the center of coordinates, the l, w, h represent the length, width, height and θ the heading angle in radians. For the camera images we have the same information but instead of 3D, a 2D bounding box is associated (cx, cy, l, w) where cx, cy represent the center pixel of the box and l, w are the length and width of the box.

A.3 The Pandaset Dataset

The Pandaset dataset was developed by Hesai and Scale with the purpose of promoting autonomous driving research. For data collection, two routes in Silicon Valley were followed, one in San Francisco and the other in El Camino Real. Not only does this dataset combine the best of Hesai sensors with Scale’s high-quality annotations, it is also the first dataset available for both academic and commercial use.[76]

Data capture was performed with a Chrysler Pacifista minivan equipped with six cameras, two LiDARs and a GNSS/IMU device with the layout shown in Figure A.3. It consists of six Leopard cameras (model LI-USB30-AR023ZWDRB) with a 1920x1080 resolution, frame rate of 30 FPS, 1/2.7” CMOS and auto exposure. The front camera uses a Foctek lens (model CS-6IR(5MP)-L) with a focal length of 6mm, horizontal angle of view of 60.5° and vertical angle of view of 43.5. The other cameras use an Evetar lens (model M128B02820WM12R1) with a focal length of 2.8mm, horizontal angle of view of 122° and vertical angle of view of 74°. For forward-facing LiDAR, a 2D scanning solid-state LiDAR (Hesai PandarGT) is used with a capture frequency of 10Hz, 150 beams, 0.16° angular resolution, 2cm distance accuracy, capable of capturing around 0.945 million points per second, a horizontal fov of 60°, a vertical fov of 20° and a range of 300m at 10 % reflectivity. The other LiDAR sensor used was a mechanical spinning LiDAR (Hesai Pandar64) with a capture frequency of 10 Hz, 64 beams, 0.167° angular resolution, 2cm distance accuracy, capture rate of 2.3 million points per second, a horizontal fov of 360°, a 40° vertical fov and a range of 200 to 10% reflectivity. For forward-facing LiDAR, a 2D scanning solid-state LiDAR (Hesai PandarGT) is used with a capture frequency of 10Hz, 150 beams, 0.16° angular resolution, 2cm distance accuracy, capable of capturing around 0.945 million points per second, a horizontal fov of 60°, a vertical fov of 20° and a range of 300m at 10% reflectivity. The other LiDAR sensor used was a mechanical spinning LiDAR (Hesai Pandar64) with a capture frequency of 10 Hz, 64 beams, 0.167° angular resolution, 2cm distance accuracy, capture rate of 2.3 million points per second, a horizontal fov of 360°, a 40° vertical fov and a range of 200 to 10% reflectivity. Finally, a GNSS&IMU (NovAtel PwrPak7) is used with a data rate of up to 100Hz and a speed accuracy of 0.03 m per second.

In addition to the images and LiDAR data, for each captured frame, information is also provided on the poses and intrinsics of each sensor, respective timestamps and the vehicle’s GPS location. In order to have a better alignment between LiDAR and images, the ego motion is estimated at each timestamp by making a linear interpolation from the vehicle’s GNSS/IMU data. Objects that are within the LiDAR point clouds and contain at least 5 points are annotated with

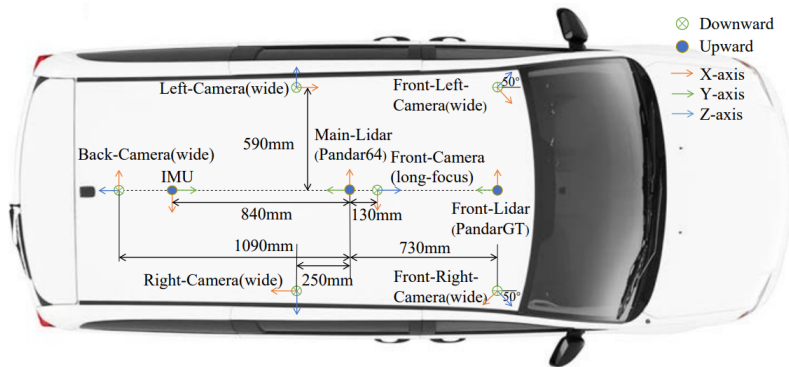


Figure A.3: Top View of the Pandaset Sensor Setup.

a 3D bounding box associated with one of the 28 object classes, and for semantic segmentation tasks associated with one of the 37 semantic labels.

Bibliography

- [1] [Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles](#), 2021 [visited May 2023].
- [2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. [TensorFlow: Large-scale machine learning on heterogeneous systems](#), 2015. Software available from tensorflow.org.
- [3] Autoridade Nacional de Segurança Rodoviária. [Relatorio de sinistralidade e fiscalizacao dezembro 2022](#), 2022 [visited September 2023].
- [4] Jorge Beltrán, Carlos Guindel, Francisco Miguel Moreno, Daniel Cruzado, Fernando García, and Arturo De La Escalera. [Birdnet: A 3d object detection framework from lidar information](#). In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3517–3523, 2018. doi:10.1109/ITSC.2018.8569311.
- [5] Kai Berger. [A State of the Art Report on Multiple RGB-D Sensor Research and on Publicly Available RGB-D Datasets](#), pages 27–44. Springer International Publishing, Cham, 2014. ISBN: 978-3-319-08651-4. doi:10.1007/978-3-319-08651-4\2.
- [6] BigML. [Democratizing object detection](#), 2021 [visited September 2023].
- [7] Avrim Blum. [On-Line Algorithms in Machine Learning](#), pages 306–325. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN: 978-3-540-68311-7. doi:10.1007/BFb0029575.
- [8] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. [nusenes: A multimodal dataset for autonomous driving](#). In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, 2020. doi:10.1109/CVPR42600.2020.01164.

- [9] Qianyu Cao. [Experimental study on the effect of loss function on object detection](#). In *Proceedings of the 2021 International Conference on Pattern Recognition and Intelligent Systems*, PRIS '21, page 81–87, New York, NY, USA, 2021. Association for Computing Machinery. ISBN: 9781450390392. doi:10.1145/3480651.3480690.
- [10] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. [Argoverse: 3d tracking and forecasting with rich maps](#). In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8740–8749, 2019. doi:10.1109/CVPR.2019.00895.
- [11] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. [Pointnet: Deep learning on point sets for 3d classification and segmentation](#). In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. doi:10.1109/CVPR.2017.16.
- [12] Shuyang Cheng, Chunyan Bai, Yang Song, Peisheng Li, Zhaoqi Leng, Ekin Dogus Cubuk, Jiquan Ngiam, and Barret Zoph. [Using automated data augmentation to advance our waymo driver](#), 2020 [visited September 2023].
- [13] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. <https://github.com/open-mmlab/mmdetection3d>, 2020.
- [14] Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wengang Zhou, Yanyong Zhang, and Houqiang Li. [Voxel r-cnn: Towards high performance voxel-based 3d object detection](#). *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(2):1201–1209, May 2021. doi:10.1609/aaai.v35i2.16207.
- [15] P. Dong and Q. Chen. *LiDAR Remote Sensing and Applications*. Remote Sensing Applications Series. CRC Press, 2018. ISBN: 9781138747241.
- [16] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. [Cut, paste and learn: Surprisingly easy synthesis for instance detection](#). In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1310–1319, 2017. doi:10.1109/ICCV.2017.146.
- [17] Yole Développement. [Lidar for automotive and industrial applications 2021](#), 2021 [visited September 2023].
- [18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2): 303–338, June 2010.
- [19] Haoqiang Fan, Hao Su, and Leonidas Guibas. [A point set generation network for 3d object reconstruction from a single image](#). In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2463–2471, 2017. doi:10.1109/CVPR.2017.264.
- [20] Lue Fan, Ziqi Pang, Tianyuan Zhang, Yu-Xiong Wang, Hang Zhao, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang. [Embracing single stride 3d object detector with sparse](#)

- [transformer](#). In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8448–8458, 2022. doi:10.1109/CVPR52688.2022.00827.
- [21] Rui Fan, Jianhao Jiao, Haoyang Ye, Yang Yu, Ioannis Pitas, and Ming Liu. [Key ingredients of self-driving cars](#). *CoRR*, abs/1906.02939, 2019.
- [22] Duarte Fernandes, António Silva, Rafael Névoa, Cláudia Simões, Dibet Gonzalez, Miguel Guevara, Paulo Novais, João Monteiro, and Pedro Melo-Pinto. [Point-cloud based 3d object detection and classification methods for self-driving applications: A survey and taxonomy](#). *Information Fusion*, 68:161–191, 2021. ISSN: 1566-2535. doi:https://doi.org/10.1016/j.inffus.2020.11.002.
- [23] Andreas Geiger, Philip Lenz, and Raquel Urtasun. [Are we ready for autonomous driving? the kitti vision benchmark suite](#). In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. doi:10.1109/CVPR.2012.6248074.
- [24] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. [A2D2: Audi Autonomous Driving Dataset](#). 2020.
- [25] Ben Graham. [Sparse 3d convolutional neural networks](#). In Xianghua Xie, Mark W. Jones, , and Gary K. L. Tam, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 150.1–150.9. BMVA Press, September 2015. ISBN: 1-901725-53-7. doi:10.5244/C.29.150.
- [26] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. [3d semantic segmentation with submanifold sparse convolutional networks](#). In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018. doi:10.1109/CVPR.2018.00961.
- [27] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. [Deep learning for 3d point clouds: A survey](#). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4338–4364, 2021. doi:10.1109/TPAMI.2020.3005434.
- [28] Arsalan Haider, Yongjae Cho, Marcell Pigniczki, Michael H. Köhler, Lukas Haas, Ludwig Kastner, Maximilian Fink, Michael Schardt, Yannik Cichy, Shotaro Koyama, Thomas Zeh, Tim Poguntke, Hideo Inoue, Martin Jakobi, and Alexander W. Koch. [Performance evaluation of mems-based automotive lidar sensor and its simulation model as per astm e3125-17 standard](#). *Sensors*, 23(6), 2023. ISSN: 1424-8220. doi:10.3390/s23063113.
- [29] Chenhang He, Ruihuang Li, Shuai Li, and Lei Zhang. [Voxel set transformer: A set-to-set approach to 3d object detection from point clouds](#). In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8407–8417, 2022. doi:10.1109/CVPR52688.2022.00823.

- [30] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. [Masked autoencoders are scalable vision learners](#). In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15979–15988, 2022. doi:10.1109/CVPR52688.2022.01553.
- [31] J. Horswell. [Recording](#). In Jay A. Siegel, Pekka J. Saukko, and Max M. Houck, editors, *Encyclopedia of Forensic Sciences (Second Edition)*, pages 368–371. Academic Press, Waltham, second edition edition, 2013. ISBN: 978-0-12-382166-9. doi:https://doi.org/10.1016/B978-0-12-382165-2.00207-5.
- [32] IBM. [What is machine learning?](#), 2019.
- [33] Sergey Ioffe and Christian Szegedy. [Batch normalization: Accelerating deep network training by reducing internal covariate shift](#). In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.
- [34] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285, may 1996. ISSN: 1076-9757.
- [35] Lindsay Kleeman and Roman Kuc. [Sonar Sensing](#), pages 753–782. Springer International Publishing, Cham, 2016. ISBN: 978-3-319-32552-1. doi:10.1007/978-3-319-32552-1\30.
- [36] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. [Pointpillars: Fast encoders for object detection from point clouds](#). In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12689–12697, 2019. doi:10.1109/CVPR.2019.01298.
- [37] Nanxi Li, Chong Pei Ho, Jin Xue, Leh Woon Lim, Guanyu Chen, Yuan Hsing Fu, and Lennon Yao Ting Lee. [A progress review on solid-state lidar and nanophotonics-based lidar sensors](#). *Laser & Photonics Reviews*, 16(11):2100511, 2022. doi:https://doi.org/10.1002/lpor.202100511.
- [38] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. ISBN: 978-3-319-10602-1.
- [39] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. [Deep learning for generic object detection: A survey](#). *International Journal of Computer Vision*, 128(2):261–318, Feb 2020. ISSN: 1573-1405. doi:10.1007/s11263-019-01247-4.
- [40] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri

- Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing. ISBN: 978-3-319-46448-0.
- [41] Clément Le Ludec, Maxime Cornet, and Antonio A Casilli. [The problem with annotation. human labour and outsourcing between france and madagascar](#). *Big Data & Society*, 10(2): 20539517231188723, 2023. doi:10.1177/20539517231188723.
- [42] Jiaxin Ma, Yoshitaka Ushiku, and Miori Sagara. [The effect of improving annotation quality on object detection datasets: A preliminary study](#). In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4849–4858, 2022. doi:10.1109/CVPRW56347.2022.00532.
- [43] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. doi:10.1017/CBO9780511809071.
- [44] Jiageng Mao, Niu Minzhe, ChenHan Jiang, hanxue liang, Jingheng Chen, Xiaodan Liang, Yamin Li, Chaoqiang Ye, Wei Zhang, Zhenguo Li, Jie Yu, Chunjing XU, and Hang Xu. [One million scenes for autonomous driving: Once dataset](#). In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran, 2021.
- [45] Paul F. McManamon, Paul Stuart Banks, Jeffrey D. Beck, Dale G. Fried, Andrew S. Huntington, and Edward A. Watson. [Comparison of flash lidar detector options](#). *Optical Engineering*, 56(3):031223, 2017. doi:10.1117/1.OE.56.3.031223.
- [46] H. Meikle. *Modern Radar Systems*. Artech House radar library. Artech House, 2008. ISBN: 9781596932432.
- [47] Gregory P. Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K. Wellington. [Lasernet: An efficient probabilistic 3d object detector for autonomous driving](#). In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12669–12678, 2019. doi:10.1109/CVPR.2019.01296.
- [48] Carsten Moenning and Neil A. Dodgson. [Fast Marching farthest point sampling](#). In *Eurographics 2003 - Posters*. Eurographics Association, 2003. doi:10.2312/egp.20031024.
- [49] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning, second edition*. Adaptive Computation and Machine Learning series. MIT Press, 2018. ISBN: 9780262039406.
- [50] Vasyl Molebny, Gary Kamerman, and Ove Steinvall. [Laser radar: from early history to new trends](#). In Gary W. Kamerman, Ove Steinvall, Keith L. Lewis, Richard C. Hollins, Thomas J. Merlet, Gary J. Bishop, and John D. Gonglewski, editors, *Electro-Optical Remote Sensing, Photonic Technologies, and Applications IV*, volume 7835, page 783502. International Society for Optics and Photonics, SPIE, 2010. doi:10.1117/12.867906.

- [51] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN: 9781605589077.
- [52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. [Pytorch: An imperative style, high-performance deep learning library](#). In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [53] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985. ISBN: 3-540-96131-3. doi:10.1007/978-1-4612-1098-6.
- [54] Prevenção Rodoviária Portuguesa. [Prevenção rodoviária](#), 2022 [visited September 2023].
- [55] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5105–5114, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN: 9781510860964.
- [56] Thinal Raj, Fazida Hanim Hashim, Aqilah Baseri Huddin, Mohd Faisal Ibrahim, and Aini Hussain. [A survey on lidar scanning mechanisms](#). *Electronics*, 9(5), 2020. ISSN: 2079-9292. doi:10.3390/electronics9050741.
- [57] Ricardo Roriz, Jorge Cabral, and Tiago Gomes. [Automotive lidar technology: A survey](#). *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6282–6297, 2022. doi:10.1109/TITS.2021.3086804.
- [58] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.
- [59] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. [Pointrcnn: 3d object proposal generation and detection from point cloud](#). In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–779, 2019. doi:10.1109/CVPR.2019.00086.
- [60] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. [Pv-rcnn: Point-voxel feature set abstraction for 3d object detection](#). In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10526–10535, 2020. doi:10.1109/CVPR42600.2020.01054.
- [61] Shaoshuai Shi, Li Jiang, Jiajun Deng, Zhe Wang, Chaoxu Guo, Jianping Shi, Xiaogang Wang, and Hongsheng Li. [Pv-rcnn++: Point-voxel feature set abstraction with local vector representation for 3d object detection](#). *International Journal of Computer Vision*, 131(2): 531–551, Feb 2023. ISSN: 1573-1405. doi:10.1007/s11263-022-01710-9.

- [62] 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)*, pages 1708–1716, 2020. doi:10.1109/CVPR42600.2020.00178.
- [63] Karen Simonyan and Andrew Zisserman. [Very deep convolutional networks for large-scale image recognition](#). In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [64] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurélien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. [Scalability in perception for autonomous driving: Waymo open dataset](#). In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2443–2451, 2020. doi:10.1109/CVPR42600.2020.00252.
- [65] Masahiro Takahashi, Yonghoon Ji, Kazunori Umeda, and Alessandro Moro. [Expandable yolo: 3d object detection from rgb-d images](#). In *2020 21st International Conference on Research and Education in Mechatronics (REM)*, pages 1–5, 2020. doi:10.1109/REM49740.2020.9313886.
- [66] T.S. Taylor. [Introduction to Laser Science and Engineering](#). CRC Press, 2019. ISBN: 9781351713757.
- [67] The MathWorks, Inc. [What are organized and unorganized point clouds?](#) Online, August 2022 [visited May 2023].
- [68] Michal Uricár, David Hurych, Pavel Krízek, and Senthil Kumar Yogamani. [Challenges in designing datasets and validation for autonomous driving](#). In Alain Trémeau, Giovanni Maria Farinella, and José Braz, editors, *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2019, Volume 5: VISAPP, Prague, Czech Republic, February 25-27, 2019*, pages 653–659. SciTePress, 2019. doi:10.5220/0007690706530659.
- [69] Leonardo Vanneschi and Mauro Castelli. [Multilayer perceptrons](#). In Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian Schönbach, editors, *Encyclopedia of Bioinformatics and Computational Biology*, pages 612–620. Academic Press, Oxford, 2019. ISBN: 978-0-12-811432-2. doi:https://doi.org/10.1016/B978-0-12-809633-8.20339-7.
- [70] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. [Dynamic graph cnn for learning on point clouds](#). *ACM Trans. Graph.*, 38(5), oct 2019. ISSN: 0730-0301. doi:10.1145/3326362.
- [71] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. [A survey of transfer learning](#). *Journal of Big Data*, 3(1):9, May 2016. ISSN: 2196-1115. doi:10.1186/s40537-016-0043-6.

- [72] Hai Wu, Jinhao Deng, Chenglu Wen, Xin Li, Cheng Wang, and Jonathan Li. [Casa: A cascade attention network for 3-d object detection from lidar point clouds](#). *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–11, 2022. doi:10.1109/TGRS.2022.3203163.
- [73] Hai Wu, Chenglu Wen, Wei Li, Xin Li, Ruigang Yang, and Cheng Wang. Transformation-equivariant 3d object detection for autonomous driving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 2795–2802, 2023.
- [74] Hai Wu, Chenglu Wen, Shaoshuai Shi, Xin Li, and Cheng Wang. [Virtual sparse convolution for multimodal 3d object detection](#). In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21653–21662, 2023. doi:10.1109/CVPR52729.2023.02074.
- [75] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. *Graph Neural Networks*, pages 27–37. Springer Nature Singapore, Singapore, 2022. ISBN: 978-981-16-6054-2. doi:10.1007/978-981-16-6054-2_3.
- [76] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, Yunlong Wang, and Diange Yang. [Pandaset: Advanced sensor suite dataset for autonomous driving](#). In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3095–3101, 2021. doi:10.1109/ITSC48978.2021.9565009.
- [77] Shimin Xiong, Bin Li, and Shiao Zhu. [Dcgmm: a single-stage 3d object detection network based on density clustering and graph neural network](#). *Complex & Intelligent Systems*, 9(3): 3399–3408, Jun 2023. ISSN: 2198-6053. doi:10.1007/s40747-022-00926-z.
- [78] Yan Yan, Yuxing Mao, and Bo Li. [Second: Sparsely embedded convolutional detection](#). *Sensors*, 18(10), 2018. ISSN: 1424-8220. doi:10.3390/s18103337.
- [79] Bin Yang, Wenjie Luo, and Raquel Urtasun. [Pixor: Real-time 3d object detection from point clouds](#). In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018. doi:10.1109/CVPR.2018.00798.
- [80] Honghui Yang, Zili Liu, Xiaopei Wu, Wenxiao Wang, Wei Qian, Xiaofei He, and Deng Cai. Graph r-cnn: Towards accurate 3d object detection with semantic-decorated local graph. In *European Conference on Computer Vision*, pages 662–679. Springer, 2022.
- [81] Honghui Yang, Tong He, Jiaheng Liu, Hua Chen, Boxi Wu, Binbin Lin, Xiaofei He, and Wanli Ouyang. Gd-mae: Generative decoder for mae pre-training on lidar point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9403–9414, June 2023.
- [82] Honghui Yang, Zili Liu, Xiaopei Wu, Wenxiao Wang, Wei Qian, Xiaofei He, and Deng Cai. Gd-mae. <https://github.com/Nightmare-n/GD-MAE>, 2023.
- [83] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. [Ipod: Intensive point-based object detector for point cloud](#). *ArXiv*, abs/1812.05276, 2018.

- [84] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. [Std: Sparse-to-dense 3d object detector for point cloud](#). In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1951–1960, 2019. doi:10.1109/ICCV.2019.00204.
- [85] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. [3dssd: Point-based 3d single stage object detector](#). In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11037–11045, 2020. doi:10.1109/CVPR42600.2020.01105.
- [86] V. Yildiz, S. Ioannidis, I. Yildiz, P. Tian, J. P. Campbell, S. Ostmo, J. Kalpathy-Cramer, M. F. Chiang, D. Erdoğan, and J. Dy. [Structural visual guidance attention networks in retinopathy of prematurity](#). In *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pages 353–357, 2021. doi:10.1109/ISBI48211.2021.9433881.
- [87] Han Woong Yoo, Norbert Druml, David Brunner, Christian Schwarzl, Thomas Thurner, Marcus Hennecke, and Georg Schitter. [Mems-based lidar for autonomous driving](#). *Elektrotechnik und Informationstechnik*, 135(6):408–415, Oct 2018. ISSN: 1613-7620. doi:10.1007/s00502-018-0635-2.
- [88] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, and Ruigang Yang. [Iou loss for 2d/3d object detection](#). In *2019 International Conference on 3D Vision (3DV)*, pages 85–94, 2019. doi:10.1109/3DV.2019.00019.
- [89] Haoran Zhou, Wei Wang, Gang Liu, and Qingguo Zhou. [Pointgat: Graph attention networks for 3d object detection](#). *Intelligent and Converged Networks*, 3(2):204–216, 2022. doi:10.23919/ICN.2022.0014.
- [90] Yin Zhou and Oncel Tuzel. [Voxelnet: End-to-end learning for point cloud based 3d object detection](#). In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018. doi:10.1109/CVPR.2018.00472.
- [91] Qinfeng Zhu, Lei Fan, and Ningxin Weng. Advancements in point cloud data augmentation for deep learning: A survey. *arXiv preprint arXiv:2308.12113*, 2023.