Universidade de Aveiro
2021

Diogo Alexandre
Amaral Conde Guedes

Identificação de objetos para veículos autónomos com base em aprendizagem automática

Object identification for autonomous vehicles based on Machine Learning

Universidade de Aveiro
2021

Diogo Alexandre
Amaral Conde Guedes

Identificação de objetos para veículos autónomos com base em aprendizagem automática

Object identification for autonomous vehicles based on Machine Learning

Dissertacao apresentada a Universidade de Aveiro para cumprimento dos requisitos necessarios a obtencao do grau de Mestre em Engenharia de Computadores e Telematica, realizada sob a orientacao cient ca do Doutor Miguel Drummond, Investigador Auxiliar no Instituto de Telecomunicacoes e da Professora Doutora Petia Georgieva do Departamento de Eletronica, Telecomunicacoes e Informatica da Universidade de Aveiro.

o júri / the jury

presidente / president	**Prof. Doutor Armando José Formoso de Pinho** Professor Catedratico da Universidade de Aveiro
vogais / examiners committee	**Doutor Hélder Filipe Pinto de Oliveira** Professor Auxiliar da Faculdade de Ciências da Universidade do Porto
	Doutor Miguel Vidal Drummond Investigador Auxiliar do Instituto de Telecomunicacoes de Aveiro

agradecimentos / acknowledgements

Gostaria de agradecer,

Aos meus pais, a minha irma e aos meus familiares mais proximos pelo total apoio que me deram ao longo de todos estes anos, nos meus melhores e piores momentos.

Ao Armando Sousa, Carlos Ribeiro, Filipe Reis, Gisela Pinto e Tiago Almeida por me terem acompanhado a enfrentar os momentos mais arduos e desa antes do curso.

Ao Fabio Cunha, Igor Bernardo, Pedro Silva, Rafael Silva, Tiago Machado e Tiago Miguel pela amizade, encorajamento e por proporcianerem momentos de descontracao que tornaram tudo mais facil.

Ao Tiago Madeira e Diogo Duarte pelos momentos de diversao, inspiracao e descoberta de novos paladares.

Em especial ao orientador e co-orientadora, Miguel Drummond e Petia Georgieva, pela con anca e total disponibilidade para partilha de conhecimento, experiências e ajuda durante todo o processo de desenvolvimento desta dissertacao.

A Bosch Car Multimedia, por ter proporcionado a oportunidade de trabalhar neste tema e fornecido conselhos no decorrer da dissertacao.

A Universidade de Aveiro e ao DETI por me terem proporciado um ambiente onde pude crescer, tanto a n vel pessoal como pro ssional, e conhecer todos os colegas com quem vivenciei a passagem por este arduo, mas enriquecedor, curso.

palavras-chave

Conducao autonoma, LiDAR, simulador, point cloud, Deep Learning, identi cacao de objetos

resumo

Conducao autonoma e uma das areas mais ativamente estudadas em inteligência arti cial. E esperado que os ve culos autonomos reduzam signi- cativamente os acidentes rodoviarios e v timas mortais quando se tornarem su cientemente maturos como opcao de transporte. Atualmente, muitos dos esforcos estao focados na prova de conceito de ve culos autonomos serem baseados num conjunto de sensores que observam o ambiente em redor. Em particular, a camara e o LiDAR sao estudados como sendo uma combinacao e ciente de sensores para realizacao de identi cacao de objectos on-line nas estradas.

Identi cacao de objetos 2D e uma area de estudo ja estabelecida no campo de Computacao Visual. O sucesso na aplicacao de tecnicas de Deep Learning levou a que a visao 2D atingisse uma precisao ao n vel Humano. No entanto, de forma a melhorar a seguranca, abordagens mais avancadas sugerem que o ve culo nao deve depender de uma unica classe de sensores. O LiDAR foi proposto como sendo um sensor adicional, particularmente devido a sua capacidade de visao 3D. Visao 3D depende dos dados capturados pelo LiDAR para reconhecer objetos em 3D. No entanto, em contraste com a identi cacao de objetos 2D, a identi cacao de objetos 3D e um campo de estudos relativamente imaturo e ainda possui muitos desa os para ultrapassar. Adicionalmente, LiDARs sao sensores dispendiosos, o que tambem torna a aquisicao de dados necessarios para o treino de tecnicas de reconhecimento de objetos 3D mais cara.

Neste contexto, esta tese de Mestrado tem como objetivo principal facilitar a identi cacao de objetos 3D, baseada em Deep Learning (DL), para ve culos autonomos. As contribuicoes espec cas deste trabalho sao as seguintes.

Primeiro, uma visao global compreensiva do estado de arte relativo as arquiteturas Deep Learning para identi cacao de objetos 3D baseadas em point clouds. O proposito desta visao global e para perceber como melhor abordar este tipo de problema no contexto de conducao autonoma.

Segundo, foi gerado um dataset sintetico, mas realista, com dados capturados por um LiDAR no ambiente virtual do GTA V. Foram desenvolvidas ferramentas para converter os dados gerados no formato do dataset do KITTI, que se tornou num standard para avaliacao de tecnicas de detecao de objetos 3D para conducao autonoma.

Terceiro, algumas das arquiteturas DL de identi cacao de objetos 3D revistas foram avaliadas com o dataset gerado. Apesar da sua performance com o dataset gerado ter sido pior que os resultados no dataset original do KITTI, os models chegaram a conseguir processar corretamente os dados sinteticos sem serem retreinados. O benef cio futuro deste trabalho consiste nos modelos poderem ser adicionalmente treinados com dados produzidos localmente e testados em cenarios variados.

O mod do GTA V implementado provou ser capaz de fornecer datasets ricos, bem estruturados e compat veis com o estado de arte em arquiteturas de identi cacao de objetos 3D.

A ferramenta desenvolvida esta dispon vel publicamente e esperamos que seja util para o avanco da identi cacao de objetos 3D para conducao autonoma, ja que remove a dependência de datasets fornecidos por terceiros.

keywords Autonomous driving, LiDAR, simulator, point cloud, Deep Learning, object identi cation

abstract

Autonomous driving is one of the most actively researched elds in arti cial intelligence. The autonomous vehicles are expected to signi cantly reduce the road accidents and casualties one day when they become su ciently mature transport option. Currently much e ort is focused to prove the concept of autonomous vehicles that is based on a suit of sensors to observe their surroundings. In particular, camera and LiDAR are researched as an e cient combination of sensors for on-line object identi cation on the road.

2D object identi cation is an already established eld in Computer Vision. The successful application of Deep Learning techniques has led to 2D vision with Human-level accuracy. However, for a matter of improved safety more advanced approaches suggest that the vehicle should not rely on a single class of sensors. LiDAR has been proposed as an additional sensor, particularly due to its 3D vision capability. 3D vision relies on LiDAR captured data to recognize objects in 3D. However, in contrast to the 2D object identi - cation, 3D object detection is a relatively immature eld and still has many challenges to overcome. In addition, LiDARs are expensive sensors, which makes the acquisition of data required for training 3D object recognition techniques expensive tasks as well.

In this context, this Master's thesis has the major goal to further facilitate the 3D object identi cation for autonomous vehicles based on Deep Learning (DL). The speci c contributions of the present work are the following.

First, a comprehensive overview of the state of the art Deep Learning architectures for 3D object identi cation based on Point Clouds. The purpose of this overview is to understand how to better approach such a problem in the context of autonomous driving.

Second, synthetic but realistic Lidar captured data was generated in the GTA V virtual environment. Tools were developed to convert the generated data into the KITTI dataset format, which has become standard in 3D object detection techniques for autonomous driving.

Third, some of the overviewed 3D object identi cation DL architectures were evaluated with the generated data. Though their performance with the generated data was worse than with the original KITTI data, the models were still able to correctly process the synthetic data without being re-trained. The future bene t of this work is that the models can be further trained with home-made data and varying testing scenarios.

The implemented GTA V mod has proved to be capable of providing rich, well-structured and compatible datasets with the state of the art 3D object identi cation architectures.

The developed tool is publicly available and we hope it will be useful in advancing 3D object identi cation for autonomous driving, as it removes the dependency from datasets provided by a third party.

Contents

Contents i

List of Figures iii

List of Tables vii

Acronyms 1

1 Introduction 2
- 1.1 Scope and motivation . . . 2
- 1.2 Objectives . . . 5
- 1.3 Contributions . . . 6
- 1.4 Document structure . . . 6

2 State of the art 8
- 2.1 Deep Learning for 2D Object Identi cation . . . 8
  - 2.1.1 Deep Learning for Image Classi cation . . . 9
  - 2.1.2 Deep Learning for 2D Object Detection . . . 10
  - 2.1.3 Deep Learning for Image Segmentation . . . 11
  - 2.1.4 Summary . . . 12
- 2.2 3D Object Identi cation Datasets Overview . . . 12
  - 2.2.1 Modelnet40 dataset . . . 12
  - 2.2.2 KITTI Object detection dataset . . . 12
- 2.3 3D Object Identi cation Evaluation Metrics . . . 14
  - 2.3.1 Precision . . . 14
  - 2.3.2 Recall . . . 14
  - 2.3.3 Precision-Recall Curve . . . 14
  - 2.3.4 Intersection over Union . . . 14
  - 2.3.5 Average Precision . . . 15
  - 2.3.6 Mean Average Precision . . . 15
  - 2.3.7 Average Orientation Similarity . . . 16
- 2.4 Deep Learning for 3D Object Identi cation . . . 16
  - 2.4.1 Multi-View Convolutional Neural Network (MVCNN) . . . 17
  - 2.4.2 PointNet . . . 20
  - 2.4.3 Multi-View 3D Object Detection (MV3D) . . . 23
  - 2.4.4 PointNet++ . . . 27

2.4.5 VoxelNet 32
2.4.6 Frustum PointNet 36
2.4.7 Complex-YOLO 39
2.4.8 PointRCNN 42
2.4.9 PointPillars 45
2.4.10 Pseudo-LiDAR 48
2.4.11 Pseudo-LiDAR++ 49
2.4.12 Point Voxel-RCNN (PV-RCNN) 52
2.4.13 Summary 54
2.5 KITTI Objection Detection Dataset in Detail 56
2.5.1 Data acquisition sensors 56
2.5.2 KITTI dataset le structure 57
2.5.3 Evaluation on KITTI Object Detection Dataset 58
2.5.4 Sample data analyses 58

3 Dataset Generation for 3D Object Identi cation 60
3.1 GTA V Mod Project 61
3.2 Implemented Features 63
3.3 LiDAR Con guration 63
3.4 Dataset generation using GTA V 65
3.4.1 Setup of GTA V Mod 65
3.4.2 Project development 65
3.4.3 Details 66
3.5 Conversion of GTA Dataset to KITTI Format 68
3.5.1 Project development 69
3.5.2 Details 69
3.6 Summary 75

4 Assessment of the Generated Datasets 77
4.1 Generated Dataset Criteria 77
4.2 LiDAR-based 3D Object Identi cation 78
4.2.1 Frustum PointNet Framework 78
4.2.2 OpenPCDet Toolbox 82
4.2.3 Results 82
4.2.4 Final Remarks 88
4.3 Image-based 3D Object Identi cation 89
4.3.1 Stereo-image capture in GTA V mod 89
4.3.2 Pseudo-LiDAR 90
4.3.3 Pseudo-LiDAR++ 91

5 Discussion and Future Work 94
5.1 Summary of Developed Work 94
5.2 Main Outcomes 95
5.3 Limitations of this Work 96
5.4 Discussion and Future Work 96

Bibliography 98

List of Figures

1.1 Number of fatalities and fatality rate on the road in the US. 2
1.2 Performance comparison between camera, RaDAR, Ultrasonic and LiDAR sensors. 3
1.3 A simpli ed drawing of the various sensors present in Google's self-driving cars. 4

2.1 ImageNet Classi cation top-5 error. 9
2.2 Evolution of the two 2D object detection frameworks. 11
2.3 Sample object of the ModelNet dataset. 12
2.4 Front view and point cloud of a KITTI object detection sample. 13
2.5 Representations of 3D data: multi-view, volumetric, mesh and point cloud. . 17
2.6 MVCNN architecture diagram. 17
2.7 Classi cation and retrieval results on the ModelNet40 dataset. 19
2.8 PointNet (Vanilla) architecture diagram. 20
2.9 Pointnet architecture diagram. 21
2.10 Classi cation results on ModelNet40 dataset. 22
2.11 Results on semantic segmentation in scenes provided by the Stanford 3D semantic parsing dataset. 22
2.12 Qualitative results for semantic segmentation. 23
2.13 MV3D architecture diagram. 24
2.14 3D bounding box proposal Recall. 26
2.15 3D localization performance on vehicles, using the KITTI validation set. . . . 26
2.16 3D detection performance on vehicles, using the KITTI validation set. 26
2.17 Ablation study of multi-view features, for 3D object detection on vehicles. . . 27
2.18 PointNet++ architecture diagram. 28
2.19 Formation of point groups using the ball query approach. 28
2.20 Creation of the multi-scale feature vector using MSG. 29
2.21 Creation of the multi-scale feature vector using MRG. 30
2.22 Classi cation results on the ModelNet40 dataset. 31
2.23 Point clouds with random point dropout on the left, and the accuracy across di erent grouping and training techniques with varying number of sampled points per point cloud. 32
2.24 VoxelNet architecture diagram. 32
2.25 Architecture of a Voxel feature encoding layer. 33
2.26 Illustration of a voxel feature vector in a 4D tensor. 34
2.27 Region proposal network architecture. 34

2.28 Performance comparison in bird's-eye view detection: AP on KITTI validation set. 35
2.29 Performance comparison in 3D object detection using AP on KITTI validation set, with an IoU of 0.7 for the car category, and 0.5 for the pedestrian and cyclist categories. 35
2.30 Depiction of the process performed by Frustum PointNet in order to detect a vehicle in a sample. 36
2.31 Frustum PointNet architecture diagram. 36
2.32 3D object localization AP score (bird's eye view) on KITTI test set. 37
2.33 3D object detection 3D AP on KITTI test set. 38
2.34 3D object detection and localization using the KITTI val set. 38
2.35 Complex-YOLO pipeline. 39
2.36 Example of a bird's eye view RGB map, disregarding the bounding boxes. . . 39
2.37 Runtime performance comparison using mAP. 40
2.38 Performance on bird's eye view detection using AP. 41
2.39 Performance comparison for 3D object detection using AP. 41
2.40 PointRCNN pipeline. 42
2.41 Illustration of bin-based localization. 43
2.42 Illustration of canonical transformation. 44
2.43 3D object detection performance on KITTI test set using AP. 44
2.44 3D object detection performance on the car class of KITTI validation set. . . 45
2.45 PointPillars pipeline. 45
2.46 Results on the KITTI test BEV detection benchmark. 46
2.47 Results on the KITTI test 3D detection benchmark. 47
2.48 Results of Average Orientation Similarity (AOS) detection benchmark, on the KITTI test set. 47
2.49 Pseudo-LiDAR 3D object detection pipeline. 48
2.50 Pseudo-LiDAR car 3D object detection performance on the KITTI validation set. 49
2.51 Pseudo-LiDAR pedestrian and cyclist 3D object detection performance on the KITTI validation set. 49
2.52 Pseudo-LiDAR car 3D object detection performance on the KITTI test set. . 49
2.53 The disparity-to-depth transform using the KITTI dataset. 50
2.54 Pseudo-LiDAR++ pipeline. 50
2.55 Pseudo-LiDAR++ object detection results on KITTI validation set of the car category. 51
2.56 Pseudo-LiDAR++ object detection results of pedestrians and cyclists, on the KITTI validation set. 51
2.57 Pseudo-LiDAR++ object detection results of the car category, on the KITTI test set. 52
2.58 PV-RCNN architecture. 53
2.59 Performance comparison on the KITTI test set, using mAP with 40 recall positions. 54
2.60 Vehicle sensor setup. 56
2.61 Sensors coordinate systems. 57

3.1 Comparison between colorless point cloud, with added color for visualization of the mod's object classes, and the colored point cloud. 60
3.2 The three weather conditions captured for each sample. 61
3.3 Comparison between the ideal point cloud and the point cloud with noise. . . 62
3.4 Illustration of the object classes. 62
3.5 Illustration of individual identi ers for objects within the same class. 62
3.6 Top-down view of the point cloud generated from the GTA V mod and a point cloud from the KITTI dataset. 64
3.7 On the left, a sample with a 3D bounding box that perfectly ts the vehicle. On the right, a sample with a 3D bounding box that does not perfectly t the vehicle. 68
3.8 KITTI sample illustrating the relation between the direction of the point cloud and the direction that the vehicle is facing. 70
3.9 Output of the sample validation scripts available in the Frustum PointNet project, on a sample produced by the GTA V mod. 71
3.10 GTA V mod location vs the location used in KITTI. 72
3.11 Coordinate system conversions from the GTA V point cloud object location to the correspondent location in the camera's coordinate system. 73
3.12 Object rotation angles. 73
3.13 Vehicle rotations a). 74
3.14 Vehicle rotations b). 74
3.15 Vehicle rotations c). 74
3.16 Transformation of the 1382 512 pixels to a 1224 470 pixels image. 75

4.1 Front view images of samples from the generated dataset. 78
4.2 Projected 2D bounding box. 79
4.3 Projected 3D bounding box. 79
4.4 Projected 3D bounding box. 79
4.5 Visualization of frustum point clouds from the samples of the generated dataset, listed in the pickled le. 81
4.6 Visualization of the intensity of each point in a KITTI point cloud. 81
4.7 Comparison between the overlap area of axis-aligned bounding boxes and rotated bounding boxes. 84
4.8 Visualization of the IoU of the predictions made by Frustum PointNet. Yellow represents the ground truth BEV bounding box, magenta the inferred BEV bounding box and cyan the overlapped area. 85
4.9 Visualization of the errors produced by Frustum PointNet on selected GTA predictions. 86
4.10 Visualization of erroneous orientation predictions. 86
4.11 Capturing stereo-image of a vehicle in GTA V. 89
4.12 Left camera image view. 90
4.13 Right camera image view. 90
4.14 Result of the PSMNet trained on the KITTI dataset. 90
4.15 Dense point cloud generated from the depth map. 91
4.16 Predicted depth map from the stereo image. 91
4.17 Generated dense Pseudo-LiDAR point loud from the created depth map. . . . 92
4.18 Final result of the Pseudo-LiDAR architecture. 92

4.19 Isolated points of the target vehicle. 92
4.20 Visualization of the previous sample on the PointNet architecture. 93

List of Tables

2.1 KITTI object detection di culty levels. This partition into di erent sample categories is a stark contrast to the ModelNet dataset. It allows us to almost intuitively infer that training a model is highly dependent on its application and scenarios, as well as how the labeling is de ned. A more thorough description of this dataset is given in Section 2.5. 13
2.2 Sensors coordinate systems comparison 57
2.3 Values for each object present in the KITTI label le. 58
2.4 KITTI label properties default values. 59

3.1 Coordinate system of the KITTI sensors and the GTA V generated point cloud and front view image. 71

4.1 Average Precision of 2D object detection using 11 and 40 recall positions. . . 83
4.2 Average Precision of BEV object detection using 11 and 40 recall positions. . 85
4.3 Average Precision of 3D object detection using 11 recall positions. 87
4.4 Average Precision of 3D object detection using 40 recall positions. 87
4.5 Average Orientation Similarity results. The evaluation scripts used by Frustum PointNet do not include a measurement for the AOS. 88

Acronyms

ADAS Advanced Driver-Assistance Systems
AOS Average Orientation Similarity
AP Average Precision
AVOD Aggregate View Object Detection
CCS Canonical Coordinate System
CNN Convolutional Neural Network
DL Deep Learning
E-RPN Euler Region Proposal Network
ECU Electronic Control Unit
FPN Feature Pyramid Networks
FPS Farthest Point Sampling
FTN Feature Transformation Network
GDC Graph-based Depth Correction
ILSVRC ImageNet Large Scale Visual Recognition Challenge
IoU Intersection over Union
ITN Input Transformation Network
KNN K-Nearest Neighbors
LiDAR Light Detection and Ranging
mAP Mean Average Precision
MLP Multi-Layered Perceptron
MRG Multi-Resolution Grouping
MSG Multi-Scale Grouping
MV3D Multi-View 3D Object Detection
MVCNN Multi-View Convolutional Neural Network
NHTSA National Highway Tra c Safety Administration
NMS Non-Maximum Suppression
PSMNet Pyramid Stereo Matching Network
PV-RCNN Point Voxel-RCNN
RaDAR Radio Detection and Ranging
RoI Region of Interest
RPN Region Proposal Network
SDN Stereo Depth Network
SIFT Scale-Invariant Feature Transform
SoNAR Sound Navigation and Ranging
SSD Single Shot Detection
SSG Single-Scale Grouping
STN Spatial Transformation Network
SVM Support Vector Machine
VFE Voxel Feature Encoding

Chapter 1

Introduction

1.1 Scope and motivation

Driving is an essential part of our everyday life, as vehicles constitute an accessible means of transportation for work, travel or leisure related activities. However, with it comes various dangers for both the driver and public road users alike. Dangers stem not only from fortuitous errors, but practically due to speeding, impaired driving and violation of tra c rules and regulations.

To prevent or diminish the repercussions derived from the widespread use of personal transportation vehicles, the automotive industry progressively developed a breadth of mechanisms and technologies, that would later be collectively known as Advanced Driver-Assistance Systems (ADAS). These include the standard seat belt, Traction Control System, Anti-lock Braking System and newer additions such as Adaptive Cruise Control, Lane Keeping Assist and Automatic Emergency Braking.

Statistical records published by National Highway Tra c Safety Administration (NHTSA) and other agencies support the e ectiveness of these safety systems. For instance, a recent study by NHTSA [46], presents the overall tendency of the fatality rate decreasing with technology advances centered around driver and road safety, illustrated in Figure 1.1.

Figure 1.1: Fatalities and fatality rate per 100 Million VMT (vehicle miles traveled) in the US by year [46].

As the human factor signi cantly contributes to driving related injuries and casualties, the next step towards a more secure transportation environment may rely on autonomous vehicles. Recent advances in Computer Science in general, and in pattern recognition in particular, constitute a major stepping stone towards this goal, enabling accurate object identi cation. This means that it is now possible for a vehicle to recognize other vehicles, pedestrians, lanes and other objects of interest, with substantial accuracy.

In practice, the pattern recognition algorithms used in self-driving vehicles are paired with a multi-modal sensor suite that allows for the perception of the surrounding 3D environment. The comprised sensors, that may include cameras, SoNAR, RaDAR and LiDAR, acquire information using di erent techniques, that take advantage of their penetration capabilities and range, to capture information only available to them. Figure 1.2 brie y highlights their strengths and weaknesses.

Figure 1.2: Performance comparison between camera (passive visual), RaDAR, Ultrasonic and LiDAR sensors [3].

Camera / Passive Visual. Digital cameras are the sensors responsible for capturing visual information of the environment. These are very sensitive to light, small and low cost due to their mass production, and are considered to be the main sensor of a self-driving vehicle. However, due to their inability of capturing environment information in poor lighting and weather conditions, and only providing indirect depth estimation for 3D mapping, RaDAR and LiDAR are used as additional sensors.

RaDAR. RaDAR systems are small, have a reasonable cost and are useful in situations where bad weather conditions such as fog, rain and snow, impair the measurements of the

camera and LiDAR sensors. RaDAR typically resorts to short pulses or frequency-modulated continuous wave (FMCW) modulation to enable time-of- ight (ToF) depth estimation for 3D mapping. Because the cost signi cantly increases with the angular resolution of the RaDAR, the phased-array antenna used in vehicles tends to have a limited number of elements, thus resulting in poor angular resolution and 3D mapping capabilities. At very short distances, they are also less e ective than ultrasonic sensors.

Ultrasonic. Ultrasonic sensors are small and inexpensive sensors that emit high-frequency sound for low-range 3D mapping, being restricted to very short distances ($\lesssim 2$ m). These sensors are capable of detecting small di erences of a centimeter or less.

LiDAR. LiDAR measurements are based on visible or infrared light, which enables very high angular resolution due to the simplicity of generating collimated laser beams. Laser modulation also enables ToF depth estimation. Both features make it possible to obtain a very detailed point cloud. However, the use of the visible and infrared light spectrums comes at a disadvantage. The rays are not capable of penetrating fog, for example. The high cost and size are also aspects that are hindering their deployment in current self-driving vehicles.

The synchronization and analysis of the information ows produced by the vehicle sensors is performed by a central Electronic Control Unit (ECU), illustrated in Figure 1.3. It is an embedded system tasked with the steering, acceleration and braking of the vehicle, based on an interpretation of the scene. The data frames captured by the vehicle sensors are supplied to Machine Learning algorithms in order to perform object identi cation, thus allowing the ECU to act according to the current situation.

Figure 1.3: A simpli ed drawing of the various sensors present in Google's self-driving cars [2].

The Deep Learning algorithms for object identi cation require the development of models trained on labeled datasets. Real-life datasets are often used for training, as they mirror real-world scenarios and environments, and the complexity associated with them. Also, Deep Learning models for complex problems require massive amounts of data as these are comprised of a huge number of parameters. Otherwise, the models would be susceptible to over tting.

In the context of autonomous vehicles, gathering such a sized dataset is not a particular limitation, as it is just a matter of collecting data from many prototype cars deployed in various locations. However, for such data to be valuable it must be accurately labeled. At best, this requires Human-supervised labeling, and, at worst manual labeling. Both are painstaking and expensive tasks that hamper the generation of and access to high-quality datasets.

Such downsides of real-life datasets have motivated many R&D groups to work on synthetic datasets. Synthetic datasets' most important advantages are:

- the ability to leverage the maturity in generating realistic scenarios brought by game engines.
- the ability to produce automatic and yet highly precise labeling.
- the simplicity involved in generating scenarios that could hardly be captured in real-life situations, such as life-threatening accidents.
- the ability to easily replicate scenarios where the model performs worst, with the objective to further improve its performance on less favorable settings.

The main motivation behind this thesis is to contribute to the generation of high-quality labeled datasets for autonomous driving.

1.2 Objectives

The main objective of this Master's thesis consists in the development of software to generate synthetic datasets for object identi cation, in the context of autonomous vehicles. An analysis of the performance of selected Deep Learning models (originally trained using real-life datasets) on a synthetically generated dataset is also performed, in order to evaluate the quality of the developed software's output and the adaptability of the models to fully synthetic datasets.

To this end, the following intermediate objectives were proposed:

1. Study of the state of the art Deep Learning (DL) architectures for object identi cation for autonomous vehicles.
2. Selection of a subset of the studied architectures to be used for the evaluation of a generated synthetic dataset.
3. Study of the publicly available solutions for the generation of synthetic datasets.
4. Development of software that leverages the environments provided by GTA V to generate labeled synthetic datasets.
5. Conversion of generated datasets to the desired KITTI dataset format.
6. Generation of a small synthetic dataset using the developed software.
7. Evaluation of the generated dataset using the selected state of the art DL architectures.

1.3 Contributions

The topic behind this thesis was originally discussed with an R&D team of Bosch Car Multimedia Portugal, in order to gain knowledge in synthetic datasets, as well as in Deep Learning architectures for object identi cation in the context of autonomous vehicles. The work done in this thesis contributes to such a knowledge gain.

The performed study of the state of the art is also a pioneering work in the University of Aveiro, involving Deep Learning approaches to 3D computer vision in the context of autonomous vehicles.

The developed software to meet the objective of this thesis is based of previous work developed in the 3rd year course of Informatics Engineering Project, available orhttps://github.com/gta-project/mod , and can be found on the GitHub pagehttps://github.com/Diogo525/dataset-generation-gtav . This repository contains all the developed projects in the course of this thesis and the associated documentations on how to setup the various environments and use the projects. It includes:

- GTA V mod for the generation of synthetic datasets.
- Python project used to convert the GTA V mod output to the KITTI dataset format.
- Python scripts to generate ground truth 2D object detection results for Frustum PointNet evaluation and to visualize segmented objects directly on point clouds.
- Python project for the calculation of the IoU of inferred 2D and 3D bounding boxes, along with bird's eye view visualizations of the overlapping area.

The synthetic dataset generated from the developed software solution is publicly available in the project's repository and consists of 50 samples, in the KITTI dataset format, based on the virtual environments provided by GTA V.

The rst-ever comparison between the performance of state of the art object identi cation architectures on a GTA V-based synthetic dataset and the original KITTI testing set.

1.4 Document structure

This document is divided into 5 chapters:

Chapter 1 - Introduction : de nition of the theme and scope of this document, along with the motivation for the developed work and the provided functionalities. It describes the contributions associated with the study of the state of the art in 3D object identi cation and the developed software solution. A summary of the document's structure is also presented.

Chapter 2 - State of the Art: a brief overview of relevant 2D object identi cation architectures, the datasets and metrics commonly used to train and evaluate the 3D object identi cation models, a review of the state of the art for 3D object identi cation in the context of autonomous vehicles, a comparison between the reviewed architectures and the selection of a subset for experimenting with the generated synthetic dataset.

Chapter 3 - Dataset Generation for 3D Object Identi cation : description of the process performed to generate synthetic datasets from GTA V and correspondent conversion to the KITTI dataset format.

Chapter 4 - Evaluation of Generated Dataset the generated dataset is evaluated with two di erent approaches to 3D object identi cation: image-based and LiDAR-based.

Chapter 5 - Conclusion and Future Work: an overview of the results and outcomes presented throughout the document, the limitations of the provided solution and a discussion towards future work.

Chapter 2

State of the art

In this chapter we present the foundation of this work, consisting in a thorough review of the current state of the art Deep Learning architectures for object identi cation, in the context of autonomous vehicles, and the correspondent analysis towards an understanding of relevant approaches to the unsolved 3D object identi cation problem. It is also discussed the reasons behind the selected architectures for the evaluation of the generated synthetic dataset and the engine used for its generation.

This chapter is divided into the following sections. Section 2.1 contains a brief overview of current Deep Learning architectures that accurately perform 2D object identi cation, as these are frequently used and adapted by networks aiming to solve the same problem but in a higher dimension. Section 2.2 brie y introduces the datasets used to train and evaluate the 3D object detection models analysed in Section 2.4. Section 2.3 describes the commonly used metrics to evaluate 3D object identi cation models. Finally, Section 2.4 explores the state of the art approaches to 3D object identi cation and discusses their advantages and downsides in the context of autonomous vehicles.

2.1 Deep Learning for 2D Object Identi cation

Machine Learning algorithms have completely outperformed handcrafted techniques in object identi cation tasks. The Machine Learning models can be easily adapted to more speci c scenarios simply by feeding the models with suitable training datasets; in addition, the lack of hard-wired handcrafted features enables a greater generalization to the input data.

More precisely, Deep Learning architectures streamlined the approaches for dealing with several challenging self-driving tasks that required a processing pipeline to break these down into explicit stages, such as lane marking detection, path planning and control for vehicle steering [5].

From predicting steering decisions, to object identi cation for tra c sign classi cation and pedestrian detection, many of the proposed DL models were initially developed to process 2D image datasets.

In general, Deep Learning algorithms for 2D image datasets can be classi ed according to their inference granularity:

- DL for 2D Image Classi cation aims to classify the entire input image with a single output label.
- DL for 2D Object Detection aims to infer the location of multiple objects present in the input image by classifying and generating bounding boxes around possible targets.
- DL for Image Segmentation aims to classify every pixel of the input image to output a segmentation map that distinguishes regions belonging to di erent object classes.

2.1.1 Deep Learning for Image Classi cation

Deep Learning for 2D image classi cation became extremely popular since the introduction of the AlexNet architecture in 2012 [33], which signi cantly outperformed previous record holders of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [58], with a classi cation error of 15.3% [29], through the use of a deep convolutional neural network.

AlexNet is the successor to the rst practical implementation of convolutional neural networks, published in 1998 by Yann Lecun et al [37], and after its remarkable success, the following years were met with several publications of CNN architectures that continued to reduce the classi cation error. In the ILSVRC 2017, the SENet [28] was able to reach an error of 2.3% [29], greatly surpassing the human top-5 classi cation error rate of 5.1%, reported in [58], on the same dataset. Figure 2.1 presents the evolution of the classi cation error results of the ImageNet Challenge competitors over the years.

Figure 2.1: ImageNet Classi cation top-5 error (%), adapted from [25].

This rapid evolution of the Deep Learning approaches can be attributed to the increased availability of large scale annotated datasets, high performance GPU clusters and signi cant design improvements of network structures and training strategies [77].

Compared to CNNs, fully-connected neural networks have several disadvantages when operating with higher dimensional datasets, such as images. The interaction between all the previous and next layer neurons of the network in combination with this type of input data, originates a network with a signi cant number of parameters and di cult to train, due to its increased capacity and larger dataset requirements [38].

The usage of CNNs over fully-connected neural networks for image-based Deep Learning tasks

is encouraged by the following simplications:

Sparse interactions. In a fully connected Neural Network, the output of each unit is dependent on the interactions between all the units of the previous layer, so the number of required parameters grows with the input image resolution. Because convolution layers use filters, which are generally several orders of magnitude smaller than the size of the input feature maps, to extract meaningful local features, the input image resolution does not have a significant impact on the number of parameters of these layers, leading to smaller memory requirements and less computational costs [24].

Parameter sharing. In a fully connected Neural Network, each weight is only used once to calculate the result of the connected output unit, whereas in a Convolution layer the same kernel parameters are used throughout the entire input feature map, according to the kernel size, padding and stride configurations [24].

Translation equivariance. It is a property that directly relates input transformations to feature vector representations. A convolution layer is translation equivariant because the application of the same translation before or after the convolution results in the same output. This property is achieved by the reuse of the kernel throughout the entire input feature map and it is useful to generate predictable transformations in the output feature vectors through corresponding input image transformations, making the learning process easier [74].

Translation invariance. The translation invariance property of a CNN means that translation transformations applied to the input image do not significantly affect the output results of the network.

Hierarchical representation. Hierarchical feature extraction is achieved by stacking multiple convolution layers, which enables the formation of intermediate representations of the input image from low to high levels of abstraction [32, 33]. In an effort to understand the representations learned by these networks, several methods were devised to visualize the receptive field of the units present in the hidden layers [43].

Current state of the art approaches to image classification are the Res2Net [17], which introduces a new CNN building block for increasing the multi-scale representation ability of the networks, and the EfficientNet [68], that studies model scaling and proposes a method to uniformly scale the depth, width and resolution of a network. The top-5 classification accuracy on ILSVRC of both publications are 94.43% and 97.1%, respectively, and the latter was capable of achieving a score of 84.4%, the highest score in top-1 classification accuracy.

2.1.2 Deep Learning for 2D Object Detection

The objective of 2D object detection is to locate and classify multiple objects in a given input image, and can be generally divided into the following categories [77]:

- Region proposal based methods : they are composed of a sequence of correlated stages comprising region proposal generation, feature extraction using a CNN, classification and bounding box regression, usually trained separately. One precursor to these

methods is the Overfeat architecture [59], that combines a CNN with the sliding window method to predict bounding boxes according to the classi cation score for each object class. Relevant architectures in this category are the R-CNN network variants [23, 22, 56, 26] and the FPN network [39].

- Regression / Classi cation based methods : the YOLO [54], YOLOv2 [55] and SSD [41] are amongst the architectures encapsulated by this category. They consist in one-step frameworks that directly map the input images to the predicted bounding box coordinates and classi cation probabilities, and thus have faster inference times and are trainable end-to-end.

The progression of the 2D object detection architectures in the past few years is illustrated in Figure 2.2.

Figure 2.2: Evolution of the two 2D object detection frameworks. SPP: Spatial Pyramid Pooling [39], FRCN: Faster R-CNN [26], RPN: Region Proposal Network [56], FCN: Fully Convolutional Network [39], BN: Batch Normalization [30], Deconvolution layers [76], sourced from [77].

2.1.3 Deep Learning for Image Segmentation

Image Segmentation studies also bene ted from the popularization of convolutional neural networks, and can divided into three di erent categories [18]:

- Semantic Segmentation : classi cation of every pixel in an image without di erentiating between objects within the same class.
- Instance Segmentation : each segmented object of the same class has a unique label. In images with multiple objects, this task is performed by rst masking the regions where objects have been detected, followed by the segmentation step.
- Part-based Segmentation : consists in a low-level decomposition into parts of already segmented entities.

The paper [42] published by Jonathan Long et al, was the rst to adapt CNNs for image segmentation. From the GoogLeNet and VGG16 classi cation networks, the authors replaced the fully connected layers with convolution layers in order to produce an output heatmap, With this transformation, the authors were able to apply transfer learning and ne-tuning for the segmentation task, and being fully convolutional networks it became possible to take input maps of variable sizes to produce correspondent output maps.

Other relevant approaches in the context of this thesis are Encoder-Decoder Based Models, Multi-Scale and Pyramid Network Based Models and R-CNN Based Models for Instance Segmentation, as described in [45].

2.1.4 Summary

This section presented a brief overview of the various 2D object identi cation approaches released in the last decade, and provides a guideline to the Deep Learning architectures that serve as the bases for the 3D object identi cation approaches described in the Section 2.4.

The strategies used to develop 2D object identi cation architectures have reached a point where the models are able to outperform Human accuracy in image classi cation and reliably detect and segment target images. The same cannot be said in relation to 3D object identi cation architectures, which are still in early stages of research and development.

2.2 3D Object Identi cation Datasets Overview

Before proceeding with the analyses of the Deep Learning architectures for 3D object identi- cation, it is important to rst get to know the datasets handled by such models.

2.2.1 Modelnet40 dataset

The ModelNet dataset is provided by Princeton University [70] and contains 127,915 CAD models of regular objects, with 662 object categories. The models are represented in a mesh format, as illustrated in Figure 2.3.

Figure 2.3: Sample object of the ModelNet dataset [70].

The ModelNet40, in particular, is a subset consisting of 12,311 shapes and de nes 40 object categories. The reviewed state of the art that makes use of this subset, the MVCNN, the PointNet and the PointNet ++, perform a dataset split of 80/20 for the training and testing sets, respectively.

2.2.2 KITTI Object detection dataset

The KITTI Object Detection dataset was published in 2012 by Andreas Geiger et al, [21] and consists of a dataset focused on autonomous vehicles. The samples provide information to perform the benchmarks for 2D object detection and orientation estimation, the 3D and bird's eye view object detection.

The authors also hold a competition on 3D object identi cation and, as such, divide the supplied dataset into two parts:

- train set: consists of 7,481 samples and is used by competitors for training and validating the Deep Learning models.
- test set: consists of 7,518 unlabeled samples. This dataset is used by the KITTI team to evaluate submitted models, referred to as online evaluation. The competitors do not have access to the labels and use this set to perform o ine evaluation of their models.

The dataset is labeled with 8 di erent classes, but only the car, pedestrian and cyclist classes are used in the benchmarking process due to their higher appearance count. Figure 2.4 illustrates a KITTI object detection sample.

Figure 2.4: Front view and point cloud of a KITTI object detection sample.

The majority of the reviewed state of the art performs a dataset split of 50/50 for training and validation sets, respectively, as proposed by [11]. With the exception of Complex-YOLO that uses a 85/15 dataset split.

The authors also de ned 3 di culty levels (Easy, Moderate and Hard) for the samples, according to an object's minimum 2D bounding box height, occlusion and truncation levels. The associated eld values for each di culty level are shown in Table 2.1.

Di culty	Min. 2D bounding box height	Occlusion level	Max. Truncation
Easy	40 px	Fully visible	15%
Moderate	25 px	Partly occluded	30%
Hard	25 px	Di cult to see	50%

Table 2.1: KITTI object detection di culty levels. This partition into di erent sample categories is a stark contrast to the ModelNet dataset. It allows us to almost intuitively infer that training a model is highly dependent on its application and scenarios, as well as how the labeling is de ned. A more thorough description of this dataset is given in Section 2.5.

2.3 3D Object Identi cation Evaluation Metrics

This section presents the various metrics used to evaluate the reviewed state of the art 3D object detection models. The Average Precision and Mean Average Precision are calculated from the Intersection-over-Union metric, that is commonly used to evaluate the performance of 3D bounding box, bird's eye view bounding box (on the top-to-bottom view) and 2D bounding box (on the front view image) detections.

2.3.1 Precision

The precision metric measures the accuracy of a model in classifying a sample as positive [16], and is calculated using Equation 2.1.

$$P = \frac{TP}{TP + FP}; \tag{2.1}$$

where TP is the number of True Positives and FP the number of False Positives.

2.3.2 Recall

The recall metric measures the ability of the model to detect positive samples, independently of how the negative samples are classi ed. It is calculated using Equation 2.2.

$$R = \frac{TP}{TP + FN}; \tag{2.2}$$

where TP is the number of True Positives and FN the number of False Negatives.

2.3.3 Precision-Recall Curve

The Precision-Recall curve shows the trade-o between the precision and recall values for di erent thresholds. It is used to determine which threshold value maximizes the precision and recall.

2.3.4 Intersection over Union

The Intersection over Union (IoU) metric, or Jaccard Index, is used to calculate the percentage of overlap between the ground truth and the predicted bounding boxes. It is calculated using the Equation 2.3.

$$\frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{\text{GT Area} \setminus \text{Predicted Area}}{\text{GT Area} [\text{Predicted Area}} \tag{2.3}$$

By computing a normalized result focusing on the bounding boxes' areas, or volumes, the IoU metric is invariant to scale. Due to this crucial property, the IoU metric is the base for the majority of the performance measures for segmentation, object detection and tracking [57].

2.3.5 Average Precision

(Approximated) Average Precision

The Average Precision summarizes the Precision-Recall curve as the weighted mean of precision's achieved at each IoU threshold, where the weight is the increase in recall from the previous threshold [36]. It is equal to the area under the Precision-Recall curve, calculated using Equation 2.4.

$$AP = \int_0^1 p(r)dr \quad \sum_{k=1}^{N} P(k) \quad r(k); \tag{2.4}$$

where $P(k)$ is the precision and $r(k)$ the recall at threshold k, and $r(k)$ is the di erence between the current and the next recall.

(Interpolated) Average Precision

Interpolated Average Precision is another way to calculate the Average Precision. Instead of using the exact precision at threshold k in each iteration, it uses the maximum precision observed across all thresholds with higher recall [44]. It is calculated using Equation 2.5.

$$AP = \sum_{k=1}^{N} \max_{\tilde{k} \quad k} P(\tilde{k}) \quad r(k); \tag{2.5}$$

where $\tilde{k}$ represents the threshold, greater than or equal than the current iteration's threshold, at which the recall is the highest. So, the selected $\tilde{k}$ corresponds to when the recall is the highest, for $\tilde{k} \quad k$.

11-Point and 40-Point Interpolated Average Precision

Other variations, instead of using the exact precision value at $\tilde{k}$, de ne a range of sampled recall values on which to observe the $\tilde{k}^{th}$ value corresponding to the highest sampled recall value. The sampled recall values used in both variations are uniformly spaced, with AP_{11} using 11 recall values in total, $R_{11} = f0; 0:1; :::; 0:9; 1g$, and AP_{40} using 41 values in total, $R_{40} = f\frac{1}{40}; \frac{2}{40}; :::; \frac{39}{40}; 1g$.

KITTI originally used the AP_{11} metric to evaluate the submitted models, as proposed by the original Pascal VOC benchmark [14]. However, in 2019, in the paper [63], published by Andrea Simonelli et al, the proposition to use AP_{40} was made due to an unwanted consequence of the AP_{11}'s recall interval starting at 0. In this case, if a single correctly matched prediction occurs, in a given IoU level, the precision will be 100% at the recall value 0, resulting in a defective evaluation of the quality of the algorithm.

2.3.6 Mean Average Precision

Whereas the Average Precision is calculated for each class, the Mean Average Precision is the mean of the Average Precision's for all the classes, given by Equation 2.6.

$$mAP = \frac{\sum_{c=1}^{C} AP(c)}{C}; \tag{2.6}$$

where C is the number of object classes.

2.3.7 Average Orientation Similarity

The AOS is the metric used by KITTI to evaluate the orientation of the 3D rotated bounding box detections, calculated using Equation 2.7.

$$AOS = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}}^{N} \max_{\tilde{r}: \tilde{r} \geq r} s(\tilde{r}); \tag{2.7}$$

where r is the recall, taking into account the 0.5 IoU threshold used on the inferred 2D bounding boxes, and $s \in [0, 1]$ is the method to compute the orientation similarity, as described in the KITTI object detection paper [21].

2.4 Deep Learning for 3D Object Identification

The previous Sections 2.2 and 2.3 provided an introduction to commonly used datasets and evaluation metrics to perform training, validation and testing of state of the art 3D object identification models.

In contrast to 2D object identification, 3D object identification is a relatively new subject in the field of Computer Vision, that emerged with the affordability of 3D mapping sensors, such as LiDAR and RGB-D cameras.

As presented in Section 1.1, many systems rely on 3D mapping technologies to take advantage of their ability to build a more accurate 3D representation of the scene and to provide assistance in situations of poor visibility conditions. These sensors collect and store the information gathered from the 3D environment scans in a point cloud representation, which consists of a set of points in euclidean space, where each point not only contains its 3D coordinates, but can also be associated with additional information such as color, surface normals and reflectivity.

In spite of the success of Convolutional Neural Networks (CNNs) for Computer Vision tasks based on 2D images, these architectures are not suitable for processing data in the form of point clouds, mainly due to their unstructured nature. There are also other characteristics of the point cloud representation that makes the direct application of CNN architectures non-trivial, including variable number of points due to object occlusions, material reflectivity, atmospheric conditions, range and other equipment limitations, measurement errors, for example due to noise, the susceptibility to suffer rotations around the local axes of the sensor's coordinate system and point permutation.

To address the unconventional characteristics of the point cloud representation, many architectures rely on pre-processing the point cloud samples in order to convert these into more structured representations, such as voxels or image views, that allow for the application of

already proven architectures used in 2D object identi cation problems. Amongst the various representations used to store and process 3D data, Figure 2.5 illustrates the commonly used ones in 3D object identi cation architectures.

Figure 2.5: Representations of 3D data: multi-view [71], volumetric [49], mesh [50] and point cloud [52].

In order to understand the progress of 3D object identi cation algorithms and the motivations behind the approaches to this unsolved problem, the rest of this section involves the analysis, in chronological order, of relevant state of the art architectures that contributed to this eld of study.

2.4.1 Multi-View Convolutional Neural Network (MVCNN)

The MVCNN architecture is one of the rst to tackle 3D object classi cation. It was published by Hang Su et al [66], in 2015, and approached this challenge by creating multiple rendered views of the input 3D object, with mesh representation, and using these for the classi cation and retrieval tasks.

Figure 2.6: MVCNN architecture diagram [66].

The decision to use the multi-view representation relied on several factors:

- convolution operations over 2D representations are less computationally expensive than over voxel-based representations, where it requires a 3D kernel to operate over the voxel grid.

- the size of the volumetric representation grows cubically with respect to the input resolution, preventing the use of models with higher resolutions due to memory and computational constraints.
- 3D object datasets are scarce compared to the massive datasets of image samples.

The authors proposed a model for combining multiple view-based descriptors of an object into a single \compiled" object descriptor. A descriptor is a oat or binary vector that represents the signature of an object, i.e. a feature vector, and can be classi ed into 2 general categories:

- 3D shape descriptors : describe native representations of 3D objects, such as meshes, voxel grids and point clouds. They present some drawbacks in regard to their high-dimensionality and low availability of labeled 3D object datasets.
- View-based descriptors : describe 3D objects from image views. They were chosen for their low-dimensionality, relatively higher e ciency and robustness against artifacts present in 3D representations of shapes, such as holes, imperfect tessellations and surface noise.

In addition, the view-based descriptors are obtained through CNN activation features, instead of handcrafted features based on geometric properties of the object. This approach enables the network to be more generalized and to better learn which views are the most informative for more accurate object classi cation.

Two camera setups were experimented with in order to create the multi-view representations of the 3D objects:

- the rst camera con guration used 12 virtual cameras pointed at a 30º angle towards the centroid of the object and are positioned around the mesh every 30 degrees, as show in Figure 2.6. With this setup, every 3D object was positioned upright.
- the second camera con guration used a total of 80 views. This setup placed a camera at each of the 20 vertices of an icosahedron, also pointing at the centroid of the model, and applied rotations of 0, 90, 180 and 270 degrees, along the axis passing through each camera and the object centroid. For this camera con guration, due to the greater number of captured views, there is no assumption of consistent upright orientation of shapes.

To increase the transformation invariance of the network, data augmentation was performed by applying a ne transformations to the multi-views of the input object.

Architecture Overview

The operations of the MVCNN architecture can be summarized in 3 main steps, illustrated in Figure 2.6, as follow:

1. The architecture starts by passing each individual object view through the CNN_1 to produce a set of view descriptors, one per view.
2. The resulting descriptors are then fed through the View-Pooling layer, which uses the element-wise maximum operation to aggregate these into a single object descriptor.

3. The object descriptor is nally given to the CNN_2 in order to produce the classi cation results.

Performance Analysis

The authors experimented with two approaches for the feature extraction step:

- Fisher Vectors (FV) : Fisher vectors with multi-scale SIFT to extract handcrafted features for object classi cation [67].
- CNN : a VGG-M network to extract \learnable" features taken from the last hidden layer nodes of the network [64].

All the developed approaches were compared between each other and with state of the art image 3D shape descriptors of the time. The results are shown in in Figure 2.7.

Figure 2.7: Classi cation and retrieval results on the ModelNet40 dataset [66].

In regards to the classi cation task, there are three main comparisons between all the approaches:

- 3D shape vs view-based descriptors : all the models using view-based descriptors outperform the 3D shape descriptor models, even when only one object view is supplied.
- handcrafted features vs \learnable" features : all the models using a CNN to extract object features were able to achieve better accuracy than the models using handcrafted features.
- feature aggregation : both the Fisher Vector and standalone CNN models produce image descriptors to be separately sent to a one-vs-rest linear SVM for classi cation. Because there could be multiple views for a given input object, the various SVM results are aggregated by simply summing up the individual class values, being the class with the highest score returned.

Comments

The MVCNN models are capable of classifying 3D objects with relatively high accuracy, reaching the 90% mark. However, this architecture is not capable of operating with input

data representing a scene, being able to classify a single input object at a time, and the transformation of the 3D mesh to 2D views may result in the loss of important features that would help to describe the input sample. The pre-processing of the 3D objects also hinders the ability of the architecture to be deployed in real-time applications that are bound by inference time.

2.4.2 PointNet

The PointNet architecture, published by Charles Qi et al, was the rst architecture to directly process and extract relevant features directly from point cloud data. Compared to other representations, PointNet is able to extract features from point clouds in their raw format, without any pre-processing steps required to transform them into multi-view or volumetric representations.

The goal of this architecture was to achieve transformation invariance, such as to rotations and scaling transformations, and permutation invariance, in order for the network to output the similar results independently of the point order of each point cloud sample.

Two iterations of PointNet were conceived:

- PointNet (Vanilla) , which is a classi cation network with the point permutation invariance property.
- PointNet , which is the full version of the network with both permutation and transformation invariance, capable of not only classifying objects but also perform semantic scene and object part segmentation.

PointNet (Vanilla) Architecture

Figure 2.8: PointNet (Vanilla) architecture diagram.

To achieve the permutation invariance property, the authors' approach was to nd a symmetric function to aggregate all the local (point) feature vectors into a single global feature vector, originating a vector independent of point order that describes the entire input point cloud. The chosen symmetric function was max pooling, represented by g in Figure 2.8.

Each local feature vector represents the features of a point in a higher dimensional embedding space, which corresponds to the projection of the input points from 3-dimensional vectors to 1024-dimensional vectors. This transformation is represented by h, where n points with 3 dimensions are rst projected into a 64-dimensional space, before being nally projected to 1024-dimensional vectors.

This transformation is achieved through shared MLPs, which are fully connected layers with the same weight shared across all the connections. At the end, the generated global feature vector, which is the structure containing the features that characterize the entire input point cloud, is passed through a nal MLP, represented by , for point cloud classi cation.

PointNet Architecture

Figure 2.9: Pointnet architecture diagram [52].

While still being invariant to point permutations, the full PointNet architecture is also invariant to point cloud transformations. This property was achieved by the introduction of two networks based of the Spatial Transformation Network (STN) [31]:

- Input Transformation Network (ITN) : transforms the entire point cloud comprised of points with 3 dimensions.
- Feature Transformation Network (FTN) : transforms the entire point cloud comprised of 64-dimensional local feature vectors, that represent the point cloud points in a higher dimensional embedding space.

The purpose of these networks is to nd a transformation matrix that is capable of transforming the input point cloud into its canonical pose, an ideal pose that has not been a ected by any transformations.

Segmentation Network

The PointNet architecture is also capable of processing points with additional channels such as color and surface normals. For the scene semantic segmentation task, the authors used a dataset of point clouds comprised of points represented by 9-dimensional vectors, corresponding to the 3D coordinates, RGB values and normalized location. In order to compare determine the e ects of handcrafted features on the inference results, the authors created a baseline model that used the same 9 channels, plus 3 additional dimensions for the local point density, local curvature and surface normal.

The portion corresponding to the Segmentation Network performs the following processing steps:

1. First, the global feature vector is separately concatenated with every local feature vector, resulting in a 1088-dimensional vector.

2. The vectors are fed to a shared MLP for a new feature extraction stage, where the previous combination of local features with the global feature vector allows the network to learn some information about the local geometry and assign quantities to each point.

3. The last shared MLP outputs a classi cation score for each point, where m is the number of classes with which to segment the target point cloud.

Performance Analysis

The classi cation accuracy of the PointNet architecture was compared with other methods that used common representations of 3D data, which were multi-view, volumetric and mesh. An analysis of the Figure 2.10 concludes that PointNet was able to outperform the state of the art mesh and volumetric approaches by a slight gap compared to the MVCNN model. The di erence in performance compared to the MVCNN architecture is due to the high maturity level of the current Deep Learning architectures for image classi cation.

Figure 2.10: Classi cation results on ModelNet40 dataset [52].

The scene segmentation performance was also measured and compared to the baseline model, that used 3 additional handcrafted point features.

Figure 2.11: Results on semantic segmentation in scenes provided by the Stanford 3D semantic parsing dataset [52].

Figure 2.12: Qualitative results for semantic segmentation [52].

It can be seen in Figure 2.11 that PointNet was able to signi cantly outperform the baseline approach, in both mean IoU and overall accuracy, supporting the signi cance of \learnable" features over handcrafted ones.

By observing the qualitative results of the scene segmentation shown in Figure 2.12, it can be concluded that the network is capable of returning smooth predictions of the scene's surfaces, even when subjected to missing points and object occlusions.

Comments

The PointNet architecture is a major advancement on 3D object identi cation. It can directly process and evaluate raw point cloud samples, avoiding the loss of important features that occurs when converting to discrete data representations (voxels) and to a lower dimensional space (image views), it is invariant to point permutation and point cloud transformations, and supports variable point attributes.

The downsides to this architecture are the support for only single object classi cation, it is not capable of detecting objects in a scene, does not take into consideration local features of neighboring points, only aggregating all individual point feature vectors into a single global feature vector, may be computationally expensive when operating with higher resolution point clouds due to the use of MLPs.

2.4.3 Multi-View 3D Object Detection (MV3D)

MV3D, published by Xiaozhi Chen et al [12], is a fusion framework that receives LiDAR point clouds and RGB images as input, and predicts oriented 3D bounding boxes around targeted objects. It aims to localize and recognize 3D objects with high precision in self-driving car scenarios.

The MV3D architecture accepts as input an RGB image of the front view of the vehicle and a multi-view representation of the point cloud, consisting in a bird's eye view and a front view.

The bird's eye view representation is the result of encoding the point cloud using the height, intensity and density of its points into a grid. The point cloud is discretized onto a 2D grid, with resolution of 0.1m, and several maps are created using the three attributes previously mentioned:

- Height maps : each cell of the grid can contain multiple points of the point cloud. To build a height map, each cell is given the highest height value that it contains. In order to decrease data loss due to the selection of one height value per cell, the point cloud is vertically subdivided into equally spaced slices. For every cell, the highest height value within a slice is determined, and the result will be M height maps for M slices.
- Intensity map : each cell of the grid contains the highest re ectivity value from all the points within, resulting in a single intensity map.
- Density map : the point density is calculated for each grid cell. In order to normalize the density values into the range [0, 1], Equation 2.8 is used.

$$\min(1:0; \frac{\log(N + 1)}{\log(64)}); \tag{2.8}$$

where N is the number of points in the cell. This means that the bird's eye view is encoded with M +2 channel features.

The front view representation of the point cloud is obtained by projecting the point cloud into a cylinder plane, and not onto an image plane, in order to generate a dense map. The projected coordinates of a point in the point cloud can be calculated with the following equations:

$$c = \frac{\text{atan2}(y; x)}{} \tag{2.9}$$

$$c = \frac{\text{atan2}(z; \sqrt{x^2 + y^2})}{} ; \tag{2.10}$$

where and are the horizontal and vertical resolution of the LiDAR sensor, and $(x; y; z)$ are the coordinates of a point of the point cloud. The front view map is encoded with the height, distance and intensity values.

MV3D network architecture

Figure 2.13: MV3D architecture diagram [12].

The MV3D architecture is divided in two parts:

1. 3D Proposal Network: the rst part of the framework extracts feature maps from the input data (bird's eye view, front view and RGB image), through successive convolution layers followed by deconvolution layers, and creates 3D bounding box proposals from the bird's eye view, to be projected onto the three input representations.

    - The bird's eye view was selected as the source for the object proposals because it preserves the objects' dimensions, they are better distributed and separate across the scene, reducing the occlusion problem, and are typically located at similar heights on the ground plane. Due to this characteristics, it is possible to obtain accurate 3D bounding boxes from the bird's eye view representation.

    - The 3D Proposal Network uses a Bounding Box Regressor to create candidate bounding boxes and determines if each proposed bounding box contains an actual object or not, which is known as Objectness Score. The regressor operates on an upsampled feature map, created by passing the bird's eye view feature map through through deconvolution layers.

2. Region-based Fusion Network: aims to fuse the multimodal features in order to perform multi-class classi cation on the 3D object proposals and predict oriented 3D bounding boxes.

    - The fusion network layers operate on aggregated feature vectors, which are the result of the element-wise mean operation between the multimodal features derived from the di erent views. Because of their di erent resolutions, the Region of Interest (RoI) Pooling operation is applied to each view feature map, resulting in three feature vectors with shared dimensions. The element-wise mean operation is marked as M in Figure 2.13.

    - The developed approach to combine multimodal feature vectors is called Deep Fusion. It uses a hierarchy of join operations (for example: concatenation, summation) and feature transformation functions before reaching the nal multiclass classi er and 3D box regressor networks.

    - A feature transformation function is a technique used to create new features from existing ones. For each hierarchy level exists 3 of these feature transformation functions, which are networks trained to capture features from the bird's eye view, front view and RGB image.

    - It is at the last 3D box regressor network that orientation regression is considered and, in order to simplify the training of proposal bounding box regression, the orientation is restricted to f 0º, 90ºg, taking into account the most common vehicle orientations.

Performance Analysis

The MV3D network was evaluated on the KITTI object detection benchmark and uses 3D bounding box recall as the metric for the evaluation of 3D object proposals. The framework was only tested on the vehicle category. To compute the IoU, two cuboids are used (the predicted cuboid and ground truth cuboid) instead of two 2D bounding boxes.

Box proposal recall is an important evaluation metric as only the regions enveloped by the proposed bounding boxes will be subjected to further processing and predictions. So, in order to reduce the number of falsely discarded proposed bounding boxes, i.e. reduce the number of false negatives, a higher recall value is the goal.

Figure 2.14 illustrates the evolution of the recall over three di erent scenarios and the results of MV3D framework are compared with the Mono3D [10] and 3DOP [9] approaches:

- the left graph shows the recall value as the IoU threshold increases from 0 to 1. The number of proposals used for this comparison is 300.
- the middle graph shows the recall value for various numbers of proposals, with a xed IoU threshold of 0.25.
- the right graph also shows the recall value for various numbers of proposals, but with a xed IoU threshold of 0.5.

Figure 2.14: 3D bounding box proposal Recall [12].

The Average Precision score (AP_{loc}) is used as the metric for evaluating 3D bounding boxes projected onto the bird's eye view, where their orientation is taken into account.

Figure 2.15: 3D localization performance on vehicles, using the KITTI validation set [12].

The 3D object detection evaluation consists in calculating the Average Precision taking into account both predicted 3D bounding boxes and bird's eye view bounding boxes. Their orientation is implicitly considered as both types have oriented bounding boxes.

Figure 2.16: 3D detection performance on vehicles, using the KITTI validation set [12].

An ablation study aims to evaluate the impact of each input representation and their combinations on the performance of the model. As observed in Figure 2.17, the bird's eye view had the most impact in the majority of the Average Precision scores, followed by the RGB view. The relevance of both bird's eye view and RGB view can be veri ed due to their pair combination resulting in the highest AP scores.

Overall, the best performance achieved by the architecture was with the feature combination from all the views.

Figure 2.17: Ablation study of multi-view features, for 3D object detection on vehicles [12].

Comments

The MV3D architecture was developed for 3D object detection in scenes represented by sparse point clouds, as the ones obtained from the LiDAR sensor.

However, the used approach relies on the projection of the point cloud onto a grid in order to create height maps, a density map and an intensity map. This projection and discretization of the original point cloud results in loss of information and data distortion. MV3D also uses the VGG-16 network [65], an already outdated image classi cation network, that requires a large number of parameters, thus not suitable for real-time application in autonomous vehicles.

2.4.4 PointNet ++

The original PointNet architecture attempted to learn a spatial encoding of each point of the point cloud and aggregate all of their individual signatures into a global feature vector.

By design, the original PointNet did not capture features from local structures of the point cloud but, as demonstrated by the success of CNNs, this is an important step towards a better understanding of the scene. As such, PointNet++, published by Charles Qi et al [53], focuses on the hierarchical feature extraction from a set of sampled points of the point cloud, in euclidean space.

PointNet ++ network architecture

Figure 2.18: PointNet++ architecture diagram [53].

The hierarchical feature extraction consists in a sequence of \Set Abstracion" levels, which abstract point cloud local regions by encoding these into feature vectors. For single scale feature extraction, where the variability of point density in the local regions is not considered, each level is comprised of the following layers and operations:

- Sampling layer: The rst step to group local sets of points is to determine their centroids. The sampling of the point cloud can be accomplished by randomly selecting N' points. However, this can lead to the selection of points close to each other and can potentially miss prominent local structures. Instead, in order to better cover the entire point cloud, the iterative Furthest Point Sampling algorithm is used. This technique nds the N' furthest points of the point cloud to create the set of cluster centroids.

- Grouping layer: Given the centroids found in the previous layer, the Grouping layer employs the Ball Query approach, illustrated in Figure 2.19, to group the neighboring points of each cluster centroid. Ball query was chosen instead of the K-Nearest Neighbors (KNN) algorithm because of its ability to guarantee a xed scale for the neighboring volume from where the points are selected.

Figure 2.19: Formation of point groups using the ball query approach.

- PointNet layer: Each set of points, comprising a local region and the correspondent centroid, is separately passed to the PointNet layer for the feature extraction step. The output is a pair formed by the centroid and the local region feature vector that encodes its neighborhood.

The PointNet implementation used in this layer does not include the feature transformation networks, and it is called \mini PointNet". It needs to be capable of outputting a feature vector of xed length for local regions with variable number of points, that are obtained by using the Ball Query algorithm.

The non-uniform sampling density, caused by density variations of points on the local region volumes, can a ect the performance of the architecture due to the inability of the learned features to generalize well for both sparse and dense regions. This is the case when using Single-Scale Grouping (SSG), where a xed range is used for the ball query algorithm to group neighboring points.

To overcome this problem, the authors propose two types of density adaptive layers to extract and combine multiple scales of local pattern features:

- Multi-Scale Grouping (MSG): This approach consists in de ning several ranges / scales for the Ball Query algorithm to nd points in the neighborhood of the cluster centroids, i.e. each cluster centroid will be associated to several local regions of di erent scales, as illustrated in Figure 2.20. The PointNet layer receives each local region's set of points separately and generates local region feature vectors for each scale. At the end, all the local region feature vectors associated to same centroid are concatenated into a single multi-scale feature vector. This approach has the disadvantage of being computationally expensive because large scale neighborhoods need to be processed for every centroid

Figure 2.20: Creation of the multi-scale feature vector using MSG [53].

- Multi-Resolution Grouping (MRG): the feature vector of a local region, at some scale level L_i, is the result of the concatenation of two feature vectors, as illustrated in Figure 2.21:

  - the right part of the feature vector is obtained by passing each subregion through the set abstraction level and summarizing the resulting feature vectors.

  - the left part of the feature vector corresponds to the global feature vector returned by a mini PointNet, after being fed the region's set of points.

  The ability of the two vectors to capture useful features varies according to the density of the local region: the left feature vector is more reliable when the local region is denser in contrast to the right feature vector, which becomes more reliable when the local region is sparser.

Figure 2.21: Creation of the multi-scale feature vector using MRG, adapted from [53].

Similar to PointNet, PointNet ++ is capable of classifying and segmenting the input point clouds:

- Classi cation submodule : for the classi cation step, the network uses each local region feature vector as input to the mini PointNet, as if they were points. They are then passed through am MLP to produce the classi cation scores.

- Segmentation submodule : given that the Set Abstraction layers subsample the original point cloud, and the need for feature extraction per point for achieving semantic segmentation, the applied solution involved the propagation of the features from the subsampled points to the originals.

  Like the hierarchical approach to feature learning, the authors adopted a hierarchical propagation strategy for extracting features for each point of the original point cloud. Each level of the hierarchy applies the same two processing steps to the corresponding input data, and the following description is related to the rst level of the Segmentation submodule:

  1. Interpolation : the output of the second Set Abstraction layer, which consists of signatures for the correspondent subsampled group of points, is subjected to interpolation of type inverse distance weighted average based on k nearest neighbors, in order to generate the same number of feature vectors outputted by the rst Set Abstraction layer.

  2. Unit PointNet : after generating the feature vectors, each is concatenated with the corresponding feature vector that belonged to the input of the second Set Abstraction layer. To update these new feature vectors, whose parts are known due to the concatenation, they are passed through a Unit PointNet. Each feature vector is passed individually and, because the Unit PointNet is similar to one-by-one convolutions in CNNs, the output is a feature vector of the same length.

  This process is repeated until the network originates a feature vector per point of the original point cloud.

Performance Analysis

To train and test the proposed architecture variations, the authors converted the ModelNet40 mesh samples to point clouds by samples points on the surface of the 3D models.

In an e ort to increase the model's performance, a second test was performed by modifying the existing dataset. Instead of 1024 points per point cloud, 5000 points were sampled from each sample of the ModelNet40 dataset, and the corresponding surface normals were added to each point. The results correponds to the entry \Ours (with normals)" in Figure 2.22.

Figure 2.22: Classi cation results on the ModelNet40 dataset [53].

The new approach was able to achieve better performance than the original PointNet architectures in both classi cation experiments, with the same number of sampled 1024 points per point cloud and only using point coordinate features.

A bigger increase in performance occurs when the point clouds are denser, with 5000 sampled points, and with the added surface normals information to the points' features, being able to surpass the performance of the MVCNN network.

Because of the constant presence of sampling irregularities in the data gathered by sensors, the model was tested by randomly dropping points during test time. The results of this experiment are shown in Figure 2.23, which compares the performance di erence between training with and without random input dropout (DP), and between the approaches for addressing sampling density variations: Single-Scale Grouping (SSG), MSG and MRG.

By observing the graph in Figure 2.23, it is possible to make the following conclusions:

- the performance of vanilla PointNet drops relatively fast with sparser point clouds but, with DP it is able to maintain its accuracy until the 256 points mark.

- the PointNet++ with SSG is able to achieve good performance with a dense point cloud of 1024 points, but it quickly falls with sparser point clouds. With DP it is able to better maintain the performance but, at the 512 points mark, starts to decrease due to the single-scale feature vectors.

- the best performance is achieved by the PointNet++ with MSG, followed by the more e cient PointNet ++ with MRG.

Figure 2.23: Point clouds with random point dropout on the left, and the accuracy across di erent grouping and training techniques with varying number of sampled points per point cloud [53].

Comments

By taking advantage of the neighboring points and considering di erent search scales, the PointNet++ was able to outperform the MVCNN and the PointNet architectures.

On the other hand, the performance on more sparse point clouds is very poor, as shown in Figure 2.23, probably because the network could not take advantage of the new strategy that considers neighboring points, as there are signi cantly less. This is important because LiDAR sensors produce highly sparse point clouds, sometimes ending up representing an object with very few points. As the previous iteration, PointNet++ does not perform object detection, it only classi es a single point cloud object.

2.4.5 VoxelNet

Previous attempts on Region Proposal Networks for sparse point clouds focused on hand-crafted feature engineering, such as the bird's eye view projection in MV3D.

In contrast, VoxelNet, published by Yin Zhou et al [79], was developed to be a generic framework for 3D object detection, performing both feature extraction and bounding box prediction of sparse point clouds with an end-to-end trainable network.

VoxelNet Architecture

Figure 2.24: VoxelNet architecture diagram [79].

The VoxelNet architecture is subdivided into 3 functional blocks:

1. Feature Learning Network:

   (a) Voxel Partition: this step receives the raw point cloud as input and partitions the 3D space into a 3D voxel grid.

   (b) Grouping of points: points of the point cloud that reside on the same voxel are grouped together. Each voxel will have a variable number of points due to the non-uniform sampling, object occlusion, among other factors related to the sensor's measurements.

   (c) Random Sampling: high resolution LiDAR sensors build point clouds with hundreds of thousands of points, which leads to high memory requirements and computational costs. To avoid these problems, random sampling is applied to every voxel to limit the number of points within.

   (d) Stacked Voxel Feature Encoding: a Voxel Feature Encoding (VFE) layer encodes the interactions between the points within a voxel and concatenates the resulting feature vector with each point-wise input of the layer, forming a point-wise feature, as shown in the Figure 2.25.

Figure 2.25: Architecture of a Voxel feature encoding layer [79].

To capture features of a voxel, the process starts by passing the point-wise input (the group of points within the voxel) to an MLP in order to map each input point vector to a higher dimensional vector, resulting in point-wise features. Then, the resulting feature vectors are combined, using the element-wise maxpool operation, in order to produce a locally aggregated feature vector, which is the signature of the voxel currently being processed. Finally, to allow the chaining of VFE layers, each point feature vector obtained from the MLP is concatenated with the locally aggregated feature vector, resulting in point-wise concatenated features. Taking into account all the voxels of the grid, the result of a VFE layer is a list of voxel feature vectors (one vector per voxel).

   (e) Sparse Tensor Representation: each voxel feature on the list is associated with the spatial coordinates of a non-empty voxel.

   To represent the list of voxel-wise features, a sparse 4D tensor is used, where for

each voxel will be a feature vector. Figure 2.26 illustrates a voxel feature vector within the 4D tensor:

Figure 2.26: Illustration of a voxel feature vector in a 4D tensor. The total number of voxels in the grid is equal to H' W' D'.

2. Convolutional Middle Layers: this functional block contains a sequence of 3D convolution layers, each followed by a Batch Normalization layer and ReLU, to aggregate the voxel-wise features with an expanding lter size, enabling a hierarchical feature extraction at increasing abstraction levels.

3. Region Proposal Network: the RPN used by VoxelNet is a modi ed version of the Faster RCNN, in order to adapt to point cloud input data. It is integrated with the 2 previous blocks to enable end-to-end learning, and outputs a probability score map and a regression map.

Figure 2.27: Region proposal network architecture [79].

Performance Analysis

The VoxelNet is evaluated on the KITTI 3D object detection benchmark, for the Car, Pedestrian and Cyclist categories. The network is trained with only the LiDAR data from KITTI.
To understand the performance advantage of end-to-end learning, a VoxelNet-based model using hand-crafted features was implemented and used as the test's baseline. This architecture was called HC-baseline, and uses the bird's eye view approach described in [12] for the handcrafted feature extraction step.

The Figure 2.28 shows the performance results in the bird's eye view detection, which evaluates the localization of objects in the ground plane (2D plane).

The HC-baseline was able to outperform the MV3D network in the bird's eye view detection, which demonstrated the e ectiveness of the implemented RPN. However, the VoxelNet achieved the greatest AP across all the categories and di culty levels, proving its dominance over the hand-crafted approach used by the baseline network.

The dimensions of the detectable objects is an important factor that contributes to better or worst performance. When a small object is sampled by the LiDAR sensor, it is represented by a small number of points, which hinders the performance of the network.

Figure 2.28: Performance comparison in bird's-eye view detection: AP (in %) on KITTI validation set [79].

Figure 2.29 shows the performance results in 3D object detection, which is a more challenging test that requires localization of objects in 3D space.

Figure 2.29: Performance comparison in 3D object detection using AP (in %) on KITTI validation set, with an IoU of 0.7 for the car category, and 0.5 for the pedestrian and cyclist categories [79].

Even though VoxelNet only uses point cloud data to perform object detection, whereas MV3D uses a combination of point cloud, fornt view image and handcrafted features in form of di erent grid maps, it is capable of outperforming the MV3D architecture on 3D object detection of vehicles with an IoU of 0.7.

Comments

In contrast to MV3D, VoxelNet only makes use of point cloud data to perform object detection in a scene and is capable of detecting cars, pedestrians and cyclist, when MV3D is only capable of detecting cars. VoxelNet also does not rely on handcrafted features, increasing the generalization of the trained model.

The downside of this architecture is the performance. Even though it surpasses previous state of the art, this architecture does not reach high enough performance to be deployed in the real world.

2.4.6 Frustum PointNet

Previous approaches to 3D object localization use the sliding window or the 3D RPN technique, which can be computationally expensive for larger scenes, thus not suitable for real-time applications that require high reaction times.

The Frustum PointNet framework, published by Charles Qi et al, addresses this issue by rst predicting 2D bounding boxes on the front view image of the scene, using already established 2D object detection architectures instead of generating object proposals from the point cloud, reducing the proposal generation time. This is illustrated in Figure 2.30.

Figure 2.30: Depiction of the process performed by Frustum PointNet in order to detect a vehicle in a sample [51].

The inference of amodal bounding boxes, performed by Frustum PointNet, consists on inferring 2D or 3D bounding boxes that enclose the entire objects, even when parts of said objects are missing in the sample data due to object occlusion, re ectivity, among others.

Frustum PointNet Architecture

Figure 2.31: Frustum PointNet architecture diagram [51].

The Frustum PointNet architecture is divided into 3 modules, as illustrated in Figure 2.31:

- Frustum Proposal: the input of the 2D object detection model is the front view RGB image. The authors used the Feature Pyramid Networks (FPN) architecture, but the framework is independent of the 2D object detection algorithm used. This network is pre-trained using ImageNet classi cation and COCO object detection datasets, and is ne-tuned using the KITTI 2D object detection dataset for classi cation and prediction of amodal 2D bounding boxes.

Through a transformation matrix, the 2D bounding boxes can be stretched depth-wise in order to obtain frustums, with a near and far planes speci c to the range of the depth sensor. Each frustum envelopes a group of points of the point cloud, leading to the creation of a set of frustum point clouds.

- 3D Instance Segmentation: each frustum point cloud is given to this module in order to predict a probability score for each point, to distinguish if it belongs to the object, or to the background, within the frustum. \Masking" is the name given to the process of extracting only the points of the frustum point cloud that correspond the detected object.

  This algorithm is implemented using a PointNet-based Network, and uses the object classi cation result obtained from the 2D object detection network to know the class of the object associated to the frustum point cloud.

- 3D Amodal Bounding Box Estimation: this module receives the segmented object points as input and uses a bounding box regression PointNet with a pre-processing transformer network to estimate the amodal oriented 3D bounding box of the object.

Performance Analysis

The authors tested Frustum PointNet using two versions of their previous works:

- Frustum PointNet v1 : uses the PointNet architecture as the backbone for feature extraction.
- Frustum PointNet v2 : uses the PointNet++ architecture as the backbone.

Both versions are rst evaluated using the KITTI test dataset, and the results of the 3D object localization and 3D object detection are shown in Figures 2.32 and 2.33.

The analysis of the bird's eye view detection test results, shown in Figure 2.32, reveals that the Frustum PointNet method was able to outperform state of the art approaches that used 3D CNNs on voxelized point clouds and handcrafted feature extraction through the bird's eye view.

Figure 2.32: 3D object localization AP score (bird's eye view) on KITTI test set [51].

The Figure 2.33 also shows remarkable improvements on on 3D object detection, where both versions accomplish better performance over the state of the art approaches.

As seen in the analysis of the PointNet++ paper, the performance of the network degrades with progressively smaller objects, where the number of points that represent them is scarce.

Figure 2.33: 3D object detection 3D AP on KITTI test set [51].

Even though the Frustum PointNet has no direct sensor fusion or multi-view aggregation steps, it is able to achieve better performance across all object categories.

To compare the performance of the two Frustum PointNet versions with a larger number of architectures, they were also evaluated with the KITTI validation set, producing the results shown in Figure 2.34.

Figure 2.34: 3D object detection and localization using the KITTI val set [51].

The Frustum PointNet architecture, in contrast to previous state of the art approaches, is capable of 3D object detection in both indoor and outdoor scenes. For instance, the performance of approaches like the MV3D, which rely on the separation of objects in bird's eye view, do not extend well to indoor scenes. Other approaches focused on indoor scenes may also not be able to be extended to highly sparse point clouds created from outdoor scenes.

Comments

Frustum PointNet is capable of generating object proposals relatively faster than other architectures by leveraging state of the art 2D object detection architectures, and is able of outperforming the previous architectures in both 3D object detection and localization.

The downside of this architecture are still the performance results, not being high enough to guarantee reliability when deployed in autonomous vehicles, and the fact that it relies on the camera. If the camera is obstructed or malfunctions, the autonomous vehicle would not have a redundancy sensor for object detection to fallback. RaDAR alone is not capable of providing enough information to allow the continuous navigation of the vehicle.

2.4.7 Complex-YOLO

Figure 2.35: Complex-YOLO pipeline [62].

Previous 3D object detection architectures for autonomous driving, mainly the VoxelNet, Frustum PointNet and MV3D, are not suitable for deployment due to their limited inference rate on high-end GPUs, which is around 7 fps. The Complex-YOLO architecture, published by Martin Simon et al [62], is capable of running at 50 fps on a Titan X GPU, and not only predicts the trajectory of surrounding object, using the Euler-Region-Proposal Network (E-RPN), but also simultaneously estimates all classes in one forward path.

Complex-YOLO Architecture

This framework, illustrated in Figure 2.35, is divided into 2 stages:

- Point Cloud Preprocessing stage: the preprocessing stage transforms the point cloud into an RGB image by projecting and discretizing it into a 2D grid, based on the height (G), intensity (B) and density (R) of the points. To capture more detail and reduce the errors of point cloud quantization, the grid cells are $0.08 \quad 0.08m^2$, in contrast to the MV3D's $0.1 \quad 0.1m^2$ grid cells. Only part of the entire point cloud is considered by the establishment of prede ned coordinate ranges (the LiDAR is 1.73m above ground and the considered maximum detection height is 3m). The result of this stage is a bird's eye view RGB map that covers an area of $80 \quad 40m^2$ in front of the origin of the LiDAR sensor, as illustrated in Figure 2.36.

Figure 2.36: Example of a bird's eye view RGB map, disregarding the bounding boxes [62].

- Complex-YOLO Network: the obtained RGB map is supplied to a simpli ed YOLOv2 CNN network [55], extended by Complex Angle Regression, via the E-RPN, to predict ve oriented 3D boxes per grid cell of the output feature map of the CNN (the transition from 2D to 3D is done by a prede ned height based on each object class), including a probability score and class score.

  The E-RPN parses the 3D object position, dimensions, class scores and orientation from the output feature map of the CNN.

Performance Analysis

All the tested architectures were run on a Nvidia Titan X and on a Titan XP, with the exception of the \Complex-YOLO TX2", which was deployed in a dedicated embedded platform (TX2).

The results show that while keeping similar performance levels, the Complex-YOLO architecture is capable of running 5 times faster on accelerated-hardware, compared to its pears.

Figure 2.37: Runtime performance comparison using mAP [62].

It is important to consider that the following results of the Complex-YOLO architecture were achieved through only point cloud data, without the aid of additional input information such as images.

The Complex-YOLO network was tested using the KITTI validation dataset, with a split of 85% for training and 15% for validation, whereas the other architectures used the o cial KITTI test set and the dataset split of 50% for training and 50% for validation.

Its performance was similar to other state of the art approaches, only able to outperform in the Cyclist category by 10%. However these results were achieved while running at 50.4 fps, as shown in Figure 2.38.

Figure 2.38: Performance on bird's eye view detection using AP [62].

The 3D object detection is a more challenging task, resulting in less favorable results for the car category, but reasonable results for the other two categories relative to the other networks, as shown in Figure 2.39.

Figure 2.39: Performance comparison for 3D object detection using AP [62].

Comments

Complex-YOLO is capable of detecting objects signi cantly faster than previous state of the art, reaching 50 fps, and the accuracy is not signi cantly a ected by the low inference time, as its performance closely matches the other architectures. It is only based on point cloud input data.

Aside from the lower performance scores, other downsides of this architecture rely on the fact that it does not operate directly on the input point cloud and applies a pre-processing step that discretizes it, which could limit further improvements to the model.

2.4.8 PointRCNN

Figure 2.40: PointRCNN pipeline [61].

In contrast to previous approaches, that usually start by generating 3D proposals from image views, volumetric representations or bird's eye views, the PointRCNN architecture, published by Shaoshuai Shi et al [61], generates 3D bounding box proposals directly from the input point cloud.

PointRCNN Architecture

The PointRCNN framework is illustrated in Figure 2.40 and is composed of 2 stages:

1. Bottom-up 3D proposal generation:

   (a) Learning point cloud representation: to capture point-wise features from the input point cloud, the PointRCNN uses PointNet ++, with MSG as the backbone network. As previously studied, the PointNet++ is able to capture features from local structures of the point cloud, and the segmentation step is used to obtain the point-wise features.

   (b) Foreground point segmentation: to each point-wise feature is added a segmentation head (a eld for indicating if the point belongs to the foreground or to the background), which is evaluated by a ground truth segmentation mask created from ground truth 3D bounding boxes.

   (c) Bin-based 3D bounding box generation: one box regression head is also added to each point-wise feature for generating 3D proposals. A box regression head is composed of (x, y, z, h, w, l,) values, representing the object's center location in the LiDAR coordinate system, the bounding box dimensions and rotation around the Y-axis (i.e. from the bird's eye view). During training, only the 3D bounding box location values (of the box regression head) are regressed from the foreground points.

The estimation of the center location of an object on the X-Z plane is performed by associating a square area around each foreground point of the object and subdividing it into equally spaced bins, as illustrated in Figure 2.41. Each bin represents a di erent (x, z) location for a possible center of the object.

Figure 2.41: Illustration of bin-based localization, adapted from [61].

The bin-based approach is also used to predict the orientation of the object by dividing the 360 degrees into n bins and performing bin classi cation.

The object dimensions are directly regressed by calculating the residual with respect to the average object size of each class in the training set.

The selection of high-quality proposals is performed by eliminating redundant bounding boxes using Non-Maximum Suppression (NMS), based on the oriented IoU from the bird's eye view.

2. Canonical 3D Box Re nement:

   (a) Point cloud region pooling: The aim of this stage is to re ne the bounding box location, orientation and foreground object con dence. To do so, each 3D box proposal is enlarged to create bigger bounding boxes that contain additional information about local context from the new enveloped points. All the points within the enlarged bounding box become associated to it and proceed for the re nement stage of this box.

   (b) Canonical transformation: The original coordinate system of all the points of the point cloud is the LiDAR coordinate system. To only estimate the residuals of the box parameters of the proposals, a transformation is applied to all the points of each proposal to transform them to the Canonical Coordinate System (CCS) of the correspondent proposal. In the CCS, the origin is at the center of the box proposal, the X and Z-axis are parallel to the ground plane, the X-axis is oriented towards the head direction of the proposal (front of the vehicle) and the Y-axis is the same as that of the LiDAR coordinate system. This is illustrated in Figure 2.42.

   By using the canonical coordinate system, the box re nement stage will be able to

learn better local spatial features for each proposal.

Figure 2.42: Illustration of canonical transformation [61].

(c) Feature learning for box proposal re nement: For the re nement of the bounding box proposals and con dence, a combination of the transformed local spatial points with their global semantic features is performed. However, before the feature concatenation is performed, three feature values are added to the coordinates of the transformed points:

- the distance of point to the LiDAR sensor in order to compensate for the loss of depth information after the transformation to the CCS.
- the correspondent re ection intensity and the segmentation mask.

The resulting concatenated features are supplied to a network with a similar structure to PointNet ++, in order to obtain a feature vector for the following con dence classi cation and box re nement steps.

Performance Evaluation

In Figure 2.43, although most of the previous methods use both RGB images and point cloud as input, the PointRCNN method achieves better performance with an architecture that only uses point clouds as input. For the pedestrian detection, the PointRCNN has similar performance to other LiDAR-only methods but is surpassed by multimodal methods which can take advantage of the detail found in RGB images to better detect pedestrians.

Figure 2.43: 3D object detection performance on KITTI test set using AP [61].

For 3D object detection of cars, the PointRCNN surpassed all the prior approaches at every di culty level of the KITTI dataset, as shown in Figure 2.44.

Figure 2.44: 3D object detection performance on the car class of KITTI validation set [61].

PointRCNN

PointRCNN is also an architecture that directly processes point clouds as input, not relying on the front view image or bird's eye view representations and achieves the best performance accross all car and cyclist detection di culties.

However, the authors don't mention the inference rate of the network, and the performance scores are not high enough to be used in critical applications like in autonomous vehicles.

2.4.9 PointPillars

The PointPillars architecture, published by Alex H. Lang et al [35], is an encoder network that uses PointNets to learn a point cloud representation organized in vertical columns (pillars). The vertical column features are used for the prediction of oriented 3D bounding boxes and allow end-to-end learning by only relying on 2D convolutional layers. The proposed architecture was able to reach an inference rate of 62Hz, and a faster version reached the 105Hz mark.

PointPillars Architecture

Figure 2.45: PointPillars pipeline [35].

The network uses point clouds as the input data and estimates oriented 3D bounding boxes around cars, cyclists and pedestrians. The architecture is subdivided into 3 stages:

1. Pillar Feature Net: It is a feature encoder network that converts a point cloud into a sparse pseudo-image.
   (a) The point cloud is discretized into an equally spaced grid on the XY plane (ground plane).

(b) The points in each pillar are augmented to 9-dimensional vectors $(x, y, z, r, x_c, y_c, z_c, x_p, y_p)$:

- (x, y, z) are the position coordinates of the point in the point cloud.
- r is the reflectance.
- (x_c, y_c, z_c) is the distance to the arithmetic mean of all points in the pillar.
- (x_p, y_p) is the the offset from the pillar's (x,y) center.

The majority of the pillars will be empty due to the sparsity of the LiDAR point cloud, and the non-empty pillars will, in general, not contain a large number of points. The sparsity of the point cloud is handled by establishing limits to the number of non-empty pillars per point cloud sample (P) and the number of points per pillar (N), in order to create a dense tensor with size (D, P, N), where D is the number of dimensions of the points.

(c) A PointNet-based encoder is used to learn a set of features, from the stacked pillars, that can be scattered to create a 2D pseudo-image (with C channels) for the next stage 2D CNN.

2. Backbone (2D CNN): A 2D CNN processes the created pseudo-image and creates a feature map. This stage uses a similar backbone network as used in VoxelNet, and is composed of 2 sub-networks:
   - a top-down network to produce features with decreasing resolutions.
   - a network that performs upsampling and feature concatenation from the different resolutions.
3. Detection Head (SSD): This stage uses the Single Shot Detection (SSD) network to perform 3D bounding box regression over the feature map obtained from the 2D CNN.

Performance Analysis

Figures 2.46 and 2.47 show the mAP results of the BEV and 3D object detection, respectively, on the KITTI test set. PointPillars was able to outperform LiDAR-only methods in almost all categories and difficulties and achieve better performance on the cars and cyclists categories compared to the fusion-based methods.

Figure 2.46: Results on the KITTI test BEV detection benchmark [35].

Figure 2.47: Results on the KITTI test 3D detection benchmark [35].

To evaluate the orientation of the 3D bounding boxes, the authors used Average Orientation Similarity (AOS), which is a method that involves the projection of the 3D bounding boxes into the image, followed by 2D detection matching and the orientation assessment of the matched bounding boxes. The results are compared in Figure 2.48.

Considering that the accuracy of the projected bounding boxes can be a ected by the orientation of their 3D boxes counterparts, PointPillars is generally outperformed by the image-only methods. However, in spite of this fact, it is able to achieve better performance on the cyclist category.

Figure 2.48: Results of Average Orientation Similarity (AOS) [21] detection benchmark [35], on the KITTI test set.

Comments

PointPillars was able to considerably outperform the inference rate of Complex-YOLO, while achieving the highest scores in 3D object detection on the car and cyclist categories. It is another step forward towards high inference rate 3D object detection, which is not only needed to achieve high reliability, but also because the embedded systems tasked to run the Deep Learning architectures in autonomous vehicles will be much cheaper and slower than the GPUs used for training and testing the models.

As stated in the paper, even though the inference rate of 105 Hz is excessive compared to the 20 Hz limit imposed by the sampling rate of the LiDAR, it is important to consider that every evaluation made on the KITTI dataset only takes into consideration the points of the point cloud that are projected onto the front view image. In a real world scenario, the 3D object detection architectures would have to process and detect objects present in the entire point cloud, becoming important the use of highly optimized architectures that are capable of achieving inference rates, on the KITTI dataset, many times faster than the LiDAR's sampling rate.

2.4.10 Pseudo-LiDAR

Figure 2.49: Pseudo-LiDAR 3D object detection pipeline [72].

The majority of the previously studied architectures perform 3D object detection on LiDAR point clouds. In contrast, the Pseudo-LiDAR approach, published by Yan Wang et al [72], focuses its attention to image-based 3D object detection on point clouds generated from monocular or stereo vision.

LiDAR-based approaches for 3D object detection have been dominating the eld with ever increasing performance results, in comparison to image-based techniques. This is in part due to the stereo-based 3D depth estimation error growing quadratically with the distance, whereas the error of time-of- ight sensors like LiDAR grows linearly.

However, in recent years, the accuracy of the depth estimation techniques have greatly improved, with the Deep Ordinal Regression Network (DORN) [15] for monocular and Pyramid Stereo Matching Network (PSMNet) [8] for stereo depth estimation. And recent developments improved the e ciency of these algorithms, allowing for accurate disparity estimation in real-time.

Pseudo-LiDAR Architecture

To take advantage of these new depth estimation techniques, the authors propose a two-step approach by rst estimating the dense pixel depth from stereo (or monocular) vision and back-projecting pixels into a 3D point cloud. By doing so, it is then possible to apply any LiDAR-based 3D object detection architectures to the data. This process is illustrated in Figure 2.49.

The experiments were performed using two multimodal 3D object detection architectures for combining an image view with the correspondent LiDAR point cloud: the AVOD [34] and Frustum PointNet architectures.

Performance Analysis

Figure 2.50 shows that the Pseudo-LiDAR approach has the best performance compared to other monocular and stereo-based approaches.

The stereo-based approaches are also able to obtain similar results, on the car category, as the 3D object detection results obtained from real LiDAR signals, but quickly degrades with the distance of the vehicles and for an IoU of 0.7. This is due to the decreasing accuracy of depth estimation for greater distances and to the low resolution of the camera used (0.4 MP).

Figure 2.50: Pseudo-LiDAR car 3D object detection performance on the KITTI validation set, reported as AP_{BEV} / AP_{3D} (%), IoU = 0.5. Mono stands for monocular. The Pseudo-LiDAR estimated by PSMNET (stereo) or DORN (monocular) are in blue. Methods with LiDAR are in gray [72].

The Pseudo-LiDAR pipeline was also tested for 3D pedestrian and cyclist detection, as shown in Figure 2.51, and resulted in a signi cant performance di erence compared to the models using real LiDAR signals. This is due to the increased di culty of 3D object detection on smaller objects in conjunction to the stereo-based technique.

Figure 2.51: Pseudo-LiDAR pedestrian and cyclist 3D object detection performance on the KITTI validation set, IoU = 0.5, reported as AP_{BEV} / AP_{3D} (%) [72].

Considering the results of the test set on the car category, shown in Figure 2.52, we can conclude that the performance of the architecture is signi cantly inferior when compared to the LiDAR-based pointclouds.

Figure 2.52: Pseudo-LiDAR car 3D object detection performance on the KITTI test set, reported as AP_{BEV} / AP_{3D} (%), with IoU = 0.7 [72].

2.4.11 Pseudo-LiDAR ++

The performance of Pseudo-LiDAR in 3D object detection is heavily a ected by the quality of the depth estimation approaches. Recent algorithms that leverage the power of Deep Learning techniques, like the aforementioned PSMNet, are able to achieve better stereo disparity

estimation, allowing for more accurate 3D point clouds.

As shown in Figures 2.50 and 2.51, the 3D object detection performance of Pseudo-LiDAR decreases signi cantly for objects at greater distances. This problem arises from the fact that stereo depth networks (SDNs) are designed to learn how to minimize the disparity error. Because the pixel disparity is more pronounced at closer distances, the algorithm can give more relevance to nearby objects' disparity errors than to faraway objects'. This is illustrated in Figure 2.53.

Figure 2.53: The disparity-to-depth transform using the KITTI dataset [75].

And, as mentioned in the paper published [75] by Yurong You et al , a single pixel disparity error implies only a 0.1m depth error at a 5m distance, but a 5.8m depth error at a 50m distance.

Pseudo-LiDAR ++ Architecture

To counter this discrepancy between the impact of the errors at di erent distances from the stereo camera, the authors propose a pipeline for accurate depth learning, rather than for disparity learning.

Figure 2.54: Pseudo-LiDAR++ pipeline [75].

However, even after the signi cant improvement on depth estimation, there is a limitation in stereo disparity estimation imposed by the discrete nature of the pixels. Being the disparity the di erence between horizontal coordinates of corresponding pixels, the result is a discretized

value dependent on the image resolution, while the depth is continuous. As higher resolution images require higher computational requirements, the pipeline uses a hybrid approach, named Graph-based Depth Correction (GDC), to combine an inexpensive 4-line LiDAR sensor, that produces highly sparse point clouds, with the generated dense stereo point cloud to correct the bias imposed by the pixel discretization. The entire pipeline is shown in the Figure 2.54.

Performance Analysis

All the entries in blue are from the Pseudo-LiDAR++ with enhanced depth estimation by the SDN and GDC. The results of 3D object detection in both Figures 2.55 and 2.56 show that:

- Pseudo-LiDAR++ is able to improve upon the previous iteration in every category and di culty.
- The depth correction step, performed by the GDC algorithm, signi cantly enhances the object detection performance compared to only using pseudo-lidar point clouds.

Figure 2.55: Pseudo-LiDAR++ object detection results on KITTI validation set of the car category, reported as AP_{BEV} / AP_{3D} (%). The input of the di erent architectures are represented by the following symbols: M : monocular images, S: stereo images, L #: sparse 4-beam LiDAR, L : 64-beam LiDAR and S: stereo images [75].

Figure 2.56: Pseudo-LiDAR++ object detection results of pedestrians (top) and cyclists (bottom) on the KITTI validation set, reported as AP_{BEV} / AP_{3D} (%), at IoU of 0.5 [75].

The performance of the architecture on 3D object detection of the car category, on the test set is shown in Figure 2.57.

Figure 2.57: Pseudo-LiDAR++ object detection results of the car category, on the KITTI test set. Comparison between the models PL++ (blue) and 64-beam LiDAR (gray), using P-RCNN, and report AP_{BEV} / AP_{3D} (%), IoU = 0.7 [75].

Comments

Whereas Pseudo-LiDAR tries to detect objects solely from monocular or stereo vision, resulting in disparity estimation limitations due to the error increasing quadratically with the distance and to pixel discretization, the Pseudo-Lidar++ makes use of a low resolution LiDAR to implement a hybrid approach to achieve better results.

Comparing the Figures 2.52 and 2.57, it is possible to conclude that the performed point cloud correction resulted in signi cantly more precise Pseudo-LiDAR point clouds, as the results approach the ones from LiDAR generated point clouds.

In contrast to the original Pseudo-LiDAR, by relying on a LiDAR sensor, the inference rate of the Pseudo-LiDAR++ approach becomes limited by the sampling rate of the LiDAR and no redundancy is achieved in case of camera failure, obstruction or poor visibility conditions.

2.4.12 Point Voxel-RCNN (PV-RCNN)

In general, 3D object detection approaches can be classi ed as:

- grid-based methods : that transform the irregular point cloud representation into a regular structure using 3D voxels or 2D bird's eye views. These data structures can then be e ciently processed, but su er from information loss due to the discretization of the data.

- point-based methods : that directly capture point features from raw point clouds. They can achieve larger receptive elds through the point set abstraction, capable of encoding point-features from a neighbourhood of any size, but are computationally expensive.

The published PV-RCNN framework, by Shaoshuai Shi et al, [60] incorporates both grid and point-based methods to improve feature learning in order to increase the 3D object detection performance.

PV-RCNN Architecture

Figure 2.58: PV-RCNN architecture [60].

The steps performed by the PV-RCNN, illustrated in Figure 2.58, are as follow:

1. Voxelization : The PV-RCNN pipeline starts by voxelizing the input point cloud into a 3D grid, where the features of the non-empty voxels are calculated by averaging the point-wise features of the points within each voxel. The point-wise features are usually the 3D coordinates and the re ectance intensities.

2. 3D sparse convolutions : The resulting volumetric representation is passed through four 3D sparse convolution layers, generating feature volumes with 1x, 2x, 4x and 8x downsampled sizes.

3. RPN : the 8x downsampled feature volume is converted into multiple 2D bird's eye view feature maps and are used to generate 3D proposals following the anchor-based approach.

4. Keypoints Sampling : the Farthest Point Sampling (FPS) algorithm is used to sample uniformly distributed points for representing the entire scene.

5. Voxel Set Abstraction Module : this module encodes the previously generated multi-scale features to the sampled keypoints. The \Set Abstraction" layer proposed by PointNet++ is used to aggregate the voxel-wise feature volumes.

6. Predicted Keypoint Weighting module : in order to better re ne the box proposals, it is important to give priority to the keypoints belonging to foreground objects. As such, this module re-weighs the keypoint features during training, using the point cloud segmentation created from ground truth 3D boxes. So the entire scene is being summarized into the set of keypoints with multi-scale semantic features.

7. RoI-grid Pooling via the Set Abstraction layer : this layer aggregates the keypoint features into RoI grids for the box proposal re nement step.

8. 3D Proposal Re nement and Con dence Prediction : the proposal re nement network uses the RoI features of each box proposal to learn how to predict the size and location residuals. It uses a 2-layer MLP adn two branches for the con dence prediction and box re nement.

Performance Analysis

The PV-RCNN pipeline outperforms the state of the art approaches to 3D object detection of cars. It achieves better performance in the easy and moderate di culties of bird's eye view detection, with a slight mAP decrease in the hard di culty, as shown in Figure 2.59.

In relation to the cyclist 3D and bird's eye view detection, the mAP surpasses the results from previous approaches in the moderate and hard di culties, by approximately 2%.

Figure 2.59: Performance comparison on the KITTI test set, using mAP with 40 recall positions [60].

Comments

PV-RCNN was able to outperform the state of the art in several object categories and di - culties, with a score surpassing the 90% score in some of the di culties associated with the car 3D and bird's eye view detection. However, the paper does not mention the inference time, which is needed in order to better compare it with Complex-YOLO and PointPillars, which are focused on this goal.

2.4.13 Summary

3D object identi cation is a relatively new eld of study that has been explored, in the last few years, with a variety of methods, from architectures interpreting point clouds as discretized voxel grids and images, to directly extracting features from the raw point clouds, many in the autonomous driving context. The eld has been progressively evolving, with the PV-RCNN architecture, the latest reviewed in this thesis, being capable of reaching the 90% mark in several categories and di culties of the KITTI dataset.

We studied the PointNet and PointNet++ architectures, that are the foundation of many subsequent approaches to 3D object identi cation. Their usage in current state of the art algorithms is motivated by the fact that these architectures directly extract features from the point cloud representation, without pre-processing steps that would discretize the input point cloud and remove potentially important features.

There are also architectures that use a combination of the camera and LiDAR sensors for di erent purposes: Frustum PointNet uses 2D object detection on the front view image in order to increase the speed and accuracy of object proposal generation, and Pseudo-LiDAR for generating a more accurate point cloud from the front-view stereo or monocular images.

However, these approaches are dependent on the camera sensors, which makes them incapable to be applied as a redundancy measure in case of camera failure, obstruction or poor visibility conditions.

It should be noted that all proposed architectures are purely feedforward, which means that no knowledge of the past (frames) is used to their advantage. This is certainly not the case for Human driving.

On the other hand, the performance of 3D object identi cation architectures has yet to reach a high enough accuracy to be deployed in consumer-level autonomous vehicles, as the models still perform rather poorly on the more di cult levels of the KITTI dataset.

In addition, in more recent architectures, the limitations of the KITTI dataset became more apparent. First, the usage of an unbalanced evaluation metric, the 11-point interpolation average precision metric, until properly addressed in 2019, and the incompleteness of the KITTI dataset annotations, which only consider the points of the point cloud that are projected onto the front view image, do not provide a solid ground for evaluating datasets for real-world scenarios. For example, even though the Complex-YOLO and PointPillars architectures are capable of achieving an inference rate of 50fps and 105fps, respectively, in the KITTI dataset, the same results are not guaranteed when taking into account 3D object detection on the entire point cloud, which is the real world scenario in autonomous driving. As mentioned by the authors of the PointPillars architecture [35], the portion of the point cloud considered by KITTI is only about 10% of its total size.

It should also be noted that the performance of an architecture, that is measured with metrics such as average precision, should not be the only aspect to consider when evaluating a model destined to perform critical tasks such as autonomous driving. Other network aspects like explainability [40] and robustness [1], should also be evaluated. For example, as an attempt to better understand what the network has learnt, the exact sequence of steps used by the network to arrive to a certain decision and what situations it is able to e ectively address or not.

The quanti cation of the trustworthiness aspect is to be highlighted for determining if a model is ready to be deployed in autonomous vehicles. This area of interest is very recent and tries to answer the question of how much can a deep neural network be trusted. Recent publications by Alexander Wong et al [73] and Andrew Hryniowski et al [27] explore this eld of study by proposing a set of metrics to quantify the trustworthiness of a model.

2.5 KITTI Objection Detection Dataset in Detail

The KITTI object detection dataset [19], published by Andreas Geiger et al, was the rst publicly available annotated dataset focused on autonomous vehicles. Due to its availability early on, it has become the standard dataset for benchmarking the 3D object identi cation architectures, as to easily compare new approaches to the previous state of the art performance results.

This section provides a detailed review of this dataset due to its selection as the targeted format for the output of the developed dataset generation software, described in Chapter 3. The importance of reproducing the system setup originally used by KITTI and generating datasets with the appropriately formatted data is due to how sensitive trained models are to the original dataset used for training. A single departure from the original system setup or the values, range and format comprising the dataset information, may lead to a decrease in performance of the model. This is also important when considering the use of transfer learning for training the models with the newly generated datasets.

2.5.1 Data acquisition sensors

The sensor setup used to acquire the KITTI dataset is illustrated in Figure 2.60.

Figure 2.60: Vehicle sensor setup [19].

In summary, the system is comprised of 4 front view 1.4 MP cameras, 2 greyscale cameras and the 2 color cameras, a Velodyne HDL-64E with 0.09 degrees angular resolution, vertical FOV of 26.8 degrees, 120 m range and sampling rate of 10 Hz, and a GPS/IMU unit.

The 4 cameras are identi ed by numbers, as follows:

- Cam 0: left greyscale camera. It is the reference camera, i.e. it is the chosen camera for the projection of a point in the Velodyne coordinate system onto an image. This point is then transformed from Cam0 the coordinate systems of the other cameras.
- Cam 1: right greyscale camera
- Cam 2: left color camera

ˆ Cam 3: right color camera

For the purpose of object identi cation, the majority of the studied Deep Learning architectures only use data from the LiDAR sensor and the left color camera (Cam 2), the exception being the Pseudo-LiDAR versions, which use both color cameras as sources for stereo disparity estimation.

The coordinate systems of the three types of sensors are presented in Table 2.2 and illustrated in Figure 2.61. Every coordinate system is right-handed.

Coordinate System	x	y	z
Camera	right	down	forward
Velodyne	forward	left	up
GPS/IMU	forward	left	up

Table 2.2: Sensors coordinate systems comparison

Figure 2.61: Sensors coordinate systems [19].

2.5.2 KITTI dataset le structure

The KITTI object detection dataset, downloaded from [20], uses 0-based indexing to identify all the les corresponding to a given sample, and has the following structure:

```
Dataset/
├── data_object_calib/
│   ├── testing/calib/
│   ├── training/calib/
│   │   └── 00000.txt
├── data_object_label_2/
│   ├── training/label_2/
│   │   └── 00000.txt
└── data_object_image_2/
    ├── testing/image_2/
    ├── training/image_2/
    │   └── 00000.png
```

```
└── data_object_velodyne/
    ├── testing/velodyne/
    └── training/velodyne/
        └── 00000.bin
```

2.5.3 Evaluation on KITTI Object Detection Dataset

KITTI supplies two portions of the dataset: one for training and validation, which is comprised of 7481 samples, and one for testing, constituting 7518 samples. The labels of the testing dataset are not supplied as it is only used by KITTI to perform online evaluation of the submitted models. The standard for training and validating a model using the training dataset, is to perform a 50/50 dataset split. It is also crucial the split must avoid sharing samples between the training and validation samples that are from the same sample sequence.

2.5.4 Sample data analyses

Camera data: every image in the KITTI dataset is given without any compression loss and in the PNG format. The original images are subjected to two post-processing steps, where they are cropped at the top and bottom to remove the sky and car hood, resulting in images with 1382 512 pixel resolution, and are then recti ed in order to remove the distortions applied by the lenses. At the end, the images present in the KITTI dataset have a resolution of 1224 470 pixels.

Label data: The label le contains all the information about the detectable objects in a scene, where each line corresponds to one. It is comprised of 15 properties described in Table 2.3:

# Values	Name	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'Dont-Care'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded, 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image: contains minX, minY, maxX, maxY pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in the camera coordinate system (in meters). It is located at the center of the base of the object.
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating con dence in detection, needed for p/r curves, higher is better.

Table 2.3: Values for each object present in the KITTI label le [47].

Not every architecture requires a dataset that provides all the properties presented in the Table 2.3, and default values should be used when encountering a 'DontCare' entry. The standard default values are listed in Table 2.4.

Name	Default Value
type	'DontCare'
truncated	-1
occluded	-1
alpha	-10
bbox	The only mandatory values
dimensions	(-1, -1, -1)
location	(-1000, -1000, -1000)
rotation_y	-10
score	Skipped when generating or labeling datasets

Table 2.4: KITTI label properties default values.

The 'DontCare' type identifies regions with non-labeled objects, and instructs the Deep Learning architectures to ignore them.

LiDAR data: each Velodyne pointcloud is stored in a binary file, as a float matrix. Each line corresponds to a point, represented by its 3D coordinates, relative to the LiDAR origin, and reflectance (which is the intensity at which the reflected light beam is received). The used Velodyne has a stack of 64 lasers, capturing at each rotation step around its z axis a maximum of 64 vertically aligned points.

Calibration data: A calibration file contains the projection matrices, in row-major order, used to convert from one coordinate system to another. This information is essential because the each sensor has its own coordinate system. This file has the following information:

- P0, P1, P2, P3 : 3 4 projection matrices, used to project a point from the reference rectified coordinate system (Cam0) onto a rectified image of Cam0 (P0 matrix), Cam1 (P1 matrix), Cam2 (P2 matrix) and Cam3 (P3 matrix).
- R0_rect : 3 3 rectifying rotation matrix of the reference camera (Cam0) to make image planes coplanar. It is the transformation matrix used to rectify a non-rectified image of Cam0.
- Tr_velo_to_cam : 3 4 transformation matrix that maps a point in the LiDAR coordinate system to a point in a non-rectified image of Cam0.
- Tr_imu_to_velo : 3 4 transformation matrix that transforms a point in the IMU coordinate system onto a point in the LiDAR coordinate system.

Chapter 3

Dataset Generation for 3D Object Identi cation

The previous chapters highlight the predominant use of the KITTI object detection dataset for benchmarking the 3D object identi cation architectures over the years. It is a way to compare di erent models proposed to the same end. For this reason, we developed a software tool to generate samples according to the KITTI dataset format.

Currently, there are mainly two free alternatives that provide virtual environments required to generate fully synthetic datasets for autonomous vehicles, which are the CARLA open source simulator [13] and the GTA V game. At the moment of writing this thesis, GTA V is the most attractive option due to greater map variety, dynamic character behaviours, population diversity and a more dynamic environment due to its physics engine. As such, GTA V was selected as the target environment for the implementation of synthetic dataset generation. This project is possible through the use of ScriptHookV library and SDK [4], developed by Alexander Blade, that enables the implementation of mods in GTA V.

Figure 3.1: Comparison between colorless point cloud, with added color for visualization of the mod's object classes, and the colored point cloud.