

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Enhancing the Tri-Dimensional Awareness of an Autonomous Surface Vehicle through Machine Learning Algorithms

Michael Neto Rosendo

MASTER'S DEGREE IN ELECTRICAL AND COMPUTERS ENGINEERING

Supervisor: Andry Maykol Gomes Pinto

Co-Supervisor: Pedro Nuno Barbosa Leite

March 29, 2022

Resumo

No mundo moderno atual, o comércio de mercadorias e serviços entre países é importante para a entrega de recursos necessários, que caso contrário não seriam acessíveis a todos. O ambiente marítimo é utilizado como meio de transporte de carga por todo o mundo, representando uma parte significativa do comércio realizado. A utilização de veículos convencionais tripulados muitas vezes põe em risco a sua integridade e a segurança das pessoas, visto que cerca de 75% a 96% de mortes marítimas são causadas por erro humano.

Os Veículos Autônomos de Superfície (ASVs) são navios não tripulados que são capazes de realizar várias tarefas, minimizando os acidentes causados por intervenção humana. Estas tarefas incluem transporte, monitorização do ambiente e a inspeção de estruturas no mar. Para realizar estas tarefas de forma segura, é importante que o navio tenha uma boa compreensão dos seus arredores, aproveitando diferentes sensores que retiram informação do ambiente.

Tendo isto em consideração, esta dissertação propõe um algoritmo de fusão de sensores que tira proveito de informação proveniente de sensores diferentes, de forma a melhorar a percepção tridimensional dos arredores do veículo. A arquitetura desenvolvida usa algoritmos de *machine learning* baseados em redes neuronais convolucionais (CNN), de forma a fundir informação proveniente de câmaras estéreo e de um LiDAR 3D, produzindo um mapa de profundidades denso. As câmaras fornecem informação de cor e textura importantes, enquanto que o LiDAR providencia informação de profundidade. Fundir a informação proveniente destes sensores melhora de uma forma geral o desempenho do problema da previsão de profundidades, aproveitando-se dos pontos fortes de cada sensor e minimizando as suas fraquezas.

A arquitetura desenvolvida foi primeiramente projectada e validada utilizando um *dataset* adquirido utilizando um veículo terrestre, de forma a provar que é capaz de gerar de forma precisa um mapa denso de profundidades, obtendo um RMSE de 3.19m, um SILog de 5.73% e 91.66% de pixels corretamente previstos. Dados marítimos reais foram adquiridos utilizando um ASV, onde a arquitetura desenvolvida foi então treinada e testada neste ambiente.

Esta dissertação contribui com uma comparação de resultados obtidos utilizando diferentes arquiteturas e processos de treino, chegando a uma arquitetura que é capaz de fundir informação de profundidade obtida por diferentes algoritmos de *feature matching* e imagens RGB. Este trabalho também contribui para uma melhor compreensão das diferenças entre ambientes terrestres e marítimos e o seu impacto nos sistemas de percepção de veículos autônomos.

Abstract

In the modern world, trading of goods and services between countries is important to deliver necessary resources that would otherwise not be accessible to everyone. The maritime environment is used as a means of transportation of cargo around the world, representing a significant part of the world's trade. Using conventional manned vehicles often leads to the endangerment of the vehicle's integrity and the safety of people, as about 75% to 96% of maritime casualties are caused by human error.

Autonomous Surface Vehicles (ASV) are unmanned vessels that are able to perform various tasks, minimizing the accidents caused by human intervention. These tasks include transportation, environmental monitoring and the inspection of off-shore structures. To safely perform them, it is important that the vessel has a good understanding of its surroundings, by taking advantage of different sensors that collect information from the environment.

Taking this into consideration, this dissertation proposes a sensor fusion algorithm that takes advantage of data provided by different sensors to improve the tridimensional awareness of the vehicle's surroundings. The developed architecture uses machine learning algorithms based on convolutional neural networks (CNN) to fuse the information provided by stereo cameras and a 3D LiDAR, producing a dense depth map. The cameras give important color and texture information, while the LiDAR provides sparse depth cues. Fusing the information provided by these sensors improves the overall performance of the depth prediction problem, by taking advantage of the strengths of each sensor, while minimizing their weaknesses.

The developed architecture was first projected and validated using an existing dataset acquired using a land vehicle, in order to prove that it is capable of accurately generating a dense depth map, reaching a RMSE of 3.19m, a SILog of 5.73% and 91.66% of correctly predicted pixels. Real maritime data was then acquired using an ASV, where the developed architecture was trained and tested in this environment.

This dissertation contributed with a comparison of results obtained using different architectures and training processes, reaching an architecture that is capable of fusing depth information calculated through different feature matching algorithms and RGB images. This work also contributed to a better understanding of the differences between land and maritime environments, and their impact on the perception systems of autonomous vehicles.

Agradecimentos

This work is partially financed by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia within project POCI-01-0145-FEDER-030010.

Ao Prof. Andry Pinto, pela oportunidade de trabalhar neste tema e pelo apoio prestado durante a realização da dissertação.

Ao co-orientador Pedro Leite pela ajuda fornecida ao longo do desenvolvimento deste trabalho e por todas as contribuições para a escrita do documento.

A todos os amigos que fiz durante os últimos 5 anos, que sempre me apoiaram e me fizeram ultrapassar os momentos mais difíceis, em especial à Ana Cruz, Beatriz Pinto, Hugo Pires, Nuno Lobo, Bruno Gonçalves, Leonardo Ribeiro, Tiago Couto, Eduardo Fonseca, João Pires e Carolina Correia.

À minha família, em especial aos meus pais por me darem a possibilidade de seguir este percurso e pelo apoio dado ao longo dos anos.

Michael Neto Rosendo

*“Our greatest glory is not in never falling,
But in rising every time we fall.”*

Oliver Goldsmith

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Objectives	3
1.4	Structure	3
2	State of the Art	5
2.1	Introduction	5
2.2	Situational Awareness	5
2.2.1	Existing Sensors	6
2.3	Sensor Fusion	10
2.3.1	Conventional Approaches	12
2.3.2	Machine Learning Approaches	14
2.4	Critical Review	26
3	A Sensor Fusion Architecture	29
3.1	Data Acquisition and Processing	30
3.1.1	Feature-Based Stereo	31
3.1.2	LiDAR Point Cloud Projection	36
3.2	Developed Architecture	39
3.2.1	Neural Network Data Preprocessing	39
3.2.2	Developed Encoder-Decoder Models	40
3.3	Network Optimization and Implementation details	46
3.3.1	Loss Function	46
3.3.2	Optimizers	47
4	Experiments and Results	51
4.1	KITTI Dataset	52
4.1.1	Evolution of the RMSE in different Architectures	54
4.1.2	Dense Depth Maps	57
4.2	Maritime Dataset - ROAM@CRAS	60
5	Conclusions and Future Work	65
5.1	Future Work	66
	References	67

List of Figures

2.1	Images of traffic signs acquired in different conditions, the first row represents a sunny day and the second a foggy day with snow	8
2.2	Simple example of how a 3D rotating LiDAR operates	9
2.3	Basic structure of a convolutional neural network	16
2.4	Architecture of LeNet-5, used for digits recognition	17
2.5	Structure of the Inception module.	18
2.6	A building block of the residual learning	18
2.7	Basic structure of a 5-layer dense block	19
2.8	The bottleneck residual block, transforming from k channels to k' channels, expansion factor t and stride s	20
2.9	Framework of the solution proposed by Wouter Gansbeke <i>et al.</i>	21
2.10	Framework of the solution proposed by E. Romera <i>et al.</i>	22
2.11	Encoder-decoder architecture proposed by J. Tang <i>et al.</i>	23
2.12	Depth completion architecture proposed by Jinsun Park <i>et al.</i>	24
2.13	Depth completion architecture proposed by Xinjing Cheng <i>et al.</i>	24
2.14	Depth completion architecture proposed by Shanshan Zhao <i>et al.</i>	25
2.15	Depth completion network based on 2D-3D fuse blocks proposed by Yun Chen <i>et al.</i>	26
2.16	Depth prediction network proposed by Pedro Leite and Andry Pinto.	26
3.1	Diagram representing an overview of the sensor fusion process.	30
3.2	Diagram representative of the process of acquiring and processing the data used to train the machine learning model.	31
3.3	Example of an image pair obtained by the left (a) and right (b) cameras, from the KITTI dataset.	32
3.4	Keypoints obtained using SIFT feature detection algorithm.	32
3.5	Example of matches obtained using SIFT descriptors and FLANN based matcher.	33
3.6	Keypoints obtained using ORB feature detection algorithm.	34
3.7	Example of matches obtained using ORB descriptors.	34
3.8	Depth points obtained using ORB (first image), SIFT (second image) and the combination of ORB and SIFT (third image), where red points represent higher depth values and dark blue points represent low depth values.	35
3.9	Example of region growing applied to the depth map previously obtained, the first image presents the depth points before growing and the second image presents the results of the region growing.	36
3.10	Representation of the 3D LiDAR and left color camera (cropped image obtained from the KITTI dataset).	37
3.11	The pinhole camera model	37

3.12	Principal point represented in the image (x,y) and camera (x_{cam}, y_{cam}) coordinate systems.	38
3.13	Example of a point cloud before projecting to 2D plane	38
3.14	Example of the projection of 3D points acquired by the LiDAR onto the left camera, where red points represent higher depth values and dark blue points represent low depth values.	39
3.15	Diagram representing the training process of the sensor fusion algorithm.	39
3.16	Example of a cropped image taking into consideration the region of interest, where red points represent higher depth values and dark blue points represent low depth values.. . . .	40
3.17	General Encoder-Decoder architecture.	41
3.18	Adapted ResNet-18 neural network.	41
3.19	DenseNet architecture used for classification.	42
3.20	MobileNetV2 architecture used for classification.	42
3.21	Simple Up-Convolution block.	43
3.22	Up-Projection block as proposed by C. Fu <i>et al.</i>	43
3.23	Up-Projection block as proposed by Iro Laina <i>et al.</i>	43
3.24	Encoder: adapted ResNet-18 (in green), Decoder: multiple Up-Projection blocks (in blue).	44
3.25	Encoder: adapted ResNet-18 (in green), Decoder: multiple Up-Projection blocks (in blue) with skip connections from the encoder blocks.	45
3.26	Encoder-Decoder cascade architecture.	45
4.1	INESC TECs' SENSE vehicle and its sensor configuration.	52
4.2	DenseNet RMSE evolution (Architectures number 1 and 2).	54
4.3	MobileNetV2 RMSE evolution (Architectures number 3 to 7).	55
4.4	ResNet RMSE evolution (Architectures number 8 to 11).	56
4.5	ResNet RMSE evolution (Architectures number 12 to 17).	57
4.6	Dense depth map obtained using the architecture number 2 - DenseNet, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.	58
4.7	Dense depth map obtained using the architecture number 13 - ResNet18 cascade, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.	59
4.8	Dense depth map obtained using the architecture number 13 - ResNet18 cascade, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.	59
4.9	Dense depth map obtained using the architecture number 13 - ResNet18 cascade, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.	60
4.10	Dense depth map obtained using architecture A - DenseNet; B - MobileNetV2 and C - ResNet18, where the 6 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the 3 predicted dense depth map and the ground truth, displayed in this order starting from the top.	62

- 4.11 Dense depth map obtained using direct inference (left column) and dense depth map after using transfer learning (right column), using architecture number 9 from table 4.1, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top. 63

List of Tables

2.1	Comparison of performances of different cameras during the night time with fog	8
2.2	Direct comparison of different sensors used in perception systems	11
4.1	Summary of the results obtained using the KITTI dataset. Higher values of δ_1 , δ_2 and δ_1 represent better results, while lower values of <i>RMSE</i> and <i>SILog</i> are better.	53
4.2	Summary of the results obtained using the maritime dataset. Higher values of δ_1 , δ_2 and δ_1 represent better results, while lower values of <i>RMSE</i> and <i>SILog</i> are better.	61

Acronyms and Symbols

Adam	Adaptive Moment Estimation
ADC	Autonomous Driving Car
AGV	Automated Guided Vehicle
ASV	Autonomous Surface Vehicle
CCVNorm	Conditional Cost Volume Norm
CGPM	Co-Attention Guided Graph Propagation
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSPN	Convolutional Spacial Propagation Network
FC	Fully Connected
FLANN	Fast Library for Approximate Nearest Neighbors
FOV	Field of View
GPR	Gaussian Process Regression
GPU	Graphics Processing Unit
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science.
IR	Infrared
LiDAR	Light Detection And Ranging
LSTM	Long short-term memory
LWIR	Long-wave infrared
ML	Machine Learning
MMW	MilliMeter-Wave
MSE	Mean Square Error
MWIR	Medium-wave infrared
NIR	Near infrared
NGP	Navigation Guided Pilot
ORB	Oriented Fast and Rotated Brief
RNN	Recurrent Neural Network
ROI	Region of interest
ROS	Robot Operating System
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SGFM	Symmetric Gated Fusion
SWIR	Short-wave infrared
ToF	Time-of-flight
Vis-NIR	Visible-near infrared

Chapter 1

Introduction

1.1 Context

Autonomous systems and autonomous mobile robots are relevant topics that have gained a lot of attention and sparked a special interest in the scientific community due to their potential and their vast and growing fields of application. These applications include freight transportation, automotive, medicine, space exploration, entertainment, environmental monitoring, military and education [1]. During the past few decades, there has been a significant evolution in autonomous systems and robots due to the developments made in microprocessors, the reduced cost of hardware and the emergence of novel algorithms. Recent developments in relevant sensors used by autonomous vehicles, such as Light Detection and Ranging (LiDAR) and cameras also contributed to their fast growth. These autonomous vehicles are commonly used for land applications due to their usefulness in the industry and our everyday life, however, there are a lot of advantages in using this kind of technology in maritime related vehicles and missions.

In the modern world, global trade is an important topic, as it is used to trade goods and services between countries, such as necessary everyday life resources that otherwise would not be accessible to everyone. Since ocean freight is one of the most used means of transportation of these goods, it is relevant to explore autonomous solutions that can improve this activity, potentially reducing costs and risks associated with human error. There are different autonomous vehicles that have been developed over the years, including maritime related vessels that further expand this activity, along with other applications. ASVs are unmanned vessels that can perform different tasks without the intervention of humans. These tasks often require moving through rough environments where avoiding collisions is of utmost importance in order to keep the vessel's integrity and to preserve the safety of people. Studies show that a significant amount of marine casualties, about 75% to 96%, are caused by human error [2]. These accidents include collisions, allisions, fires and explosions, towing operations and tanker accidents [2]. ASVs are a great alternative approach to different applications where humans are required to be in dangerous environments, being capable of preventing these accidents caused by human error. These applications include the collection of oceanographic data, inspection of off-shore structures, environmental monitoring, replacement of

regular weather boats, as well as other tasks that are dangerous for humans [2].

With the purpose of performing all these tasks with limited or no human interaction, it is important that the vehicle has a good understanding of its surroundings and is integrated with perception and navigation techniques that increase their autonomy. The development and improvement of these perception techniques are of utmost importance as they are a key factor when the vehicle is navigating, avoiding obstacles, or even when doing complex missions such as inspection of structures that vary in shape, or even the inspection of the environment itself. Using perception and recognition in the ocean can even be more challenging, in some respects, than their land counterparts, in the sense that ocean structures and vehicles have a big variety of shapes and there are less points of reference to take into consideration.

1.2 Motivation

The maritime environment is inherently rough due to the constantly moving water and the often adverse weather conditions such as wind. Over the years, different sensors and methods have been used and researched, however, these methods are mostly focused on land environments, resulting in a reduced amount of literature on maritime related solutions. These sensors have strengths and weaknesses, performing better in some situations and failing to provide the expected results in others [3]. For example, LiDAR is a sensor that directly gives tridimensional information in the form of a point cloud, but it is very limited in terms of distance and resolution, resulting in a sparse point cloud that has limited information [4]. When using these point clouds, it is important that a reasonable amount of points are collected, as well as points with accurate information. In various situations, the lack of 3D points limits the object recognition problem and degrades the quality of depth information. On the other hand, there are other sensors such as cameras that capture images that are able to translate information about color and texture. Cameras cover a broader area and can detect objects that are further away from the sensor, but they have the disadvantage that they cannot directly give depth information [4]. Therefore, it is important that the ASV does not solely rely in the information provided by one type of sensor. Taking this into consideration, creating an architecture that can extract information and help ASVs operate in the maritime environment is relevant, as it has to be able to perform well even in situations where the conditions are not favourable to collect data from sensors.

To get the best out of the aforementioned sensors, the information derived from them can be combined, e.g. using color information from the camera and sparse depth cues from the LiDAR - this is called multimodal sensor fusion. This technique has the main objective of making one sensor compensate for the lack of information given by the other sensor, so that the result is a more reliable and accurate representation of the environment [5]. In recent years, different machine learning approaches and techniques have been developed to fuse multimodal information. When compared to conventional approaches, machine learning has the advantage of learning correlations between data acquired by different sensors, as well as the ability to automatically recognize patterns and features in that data. However, these approaches tend to require high computational

power, big and expensive amounts of data, and need to be carefully designed to operate in real-time situations. In general, machine learning is a topic that has been getting a lot of attention in the past few years. Machine learning algorithms are dynamic, receive data as an input and make predictions and decisions based on this data without being specifically programmed to do such task. To learn, these algorithms use training data, adjusting internal parameters so that their predictions get better and better during the training phase. In the past few years, there have been a lot of advances in machine learning due to the fast growth in computer power, the availability of datasets and the progress made in new and better algorithms [6].

Although there is a vast amount of proposed architectures that use machine learning algorithms to combine information provided by different sensors, these methods are typically developed for land vehicles. As mentioned before, there are a lot of advantages in using maritime related autonomous vehicles, as such, it is important that these methods are further explored and developed for this environment.

1.3 Objectives

The main goal of this dissertation is to study and develop a machine learning architecture that is capable of enhancing the tri-dimensional awareness of an ASV by fusing the data obtained by multimodal sensors to generate a dense depth map. To achieve this, the general objectives of this work are:

- Development of an architecture that uses convolutional neural networks to combine multimodal information to ensure that the ASV has an understanding of the surrounding environment in different scenarios and weather conditions. This information includes 3D point clouds provided by a LiDAR and RGB images provided by a stereo camera pair.
- Extensive set of experiments of the developed architecture and its variations using the KITTI dataset [7] for quantitative evaluation.
- Develop a transfer learning methodology and assess the reliability of the developed approach in real-world maritime data previously collected at the *Leixões*' port using the SENSE ASV [8].

1.4 Structure

In addition to the introductory chapter 1, this document is comprised of a state of the art review (chapter 2) which is divided in four sections. The first section (2.1) is a small introduction, section 2.2 presents a comparison between relevant sensors that are commonly used by autonomous vehicles to understand their surroundings in different environments and weather conditions, as well as some of their applications and section 2.3 discusses how the information from different sensor modalities can be fused to achieve better results, comparing conventional approaches with machine learning approaches, presenting relevant works that use these approaches to obtain a dense

depth map. The state of the art chapter ends with section 2.4, where some of the most important ideas and conclusions taken from the conducted research are presented. Chapter 3 presents the acquirement, processing and organization of the data used by the neural network, along with the multiple developed machine learning architectures and their variations. Chapter 4 presents the results obtained using the developed architectures using the proposed metrics, as well as the final dense depth maps. Finally, chapter 5 presents a review of the developed work and the conclusions derived from it, along with the future work.

Chapter 2

State of the Art

The objective of this chapter is to present an overview of existing state-of-the-art sensors and fusion algorithms. Commonly used and thoroughly investigated sensor technologies will be further discussed and compared in subsection 2.2.1, giving a good understanding of the strengths and weaknesses of each sensor. Different state-of-the-art sensor fusion setups and methodologies to exploit the advantages and disadvantages of each sensor will be presented in subsection 2.3. In recent years, machine learning has been used to create and improve different systems, including the fusion of sensor information, different approaches to this problematic will be presented and discussed in subsection 2.3.2.1.

2.1 Introduction

Autonomous systems proved to be relevant in different areas, so much so that research regarding mobile robots is being conducted all over the world, resulting in increasingly better state-of-the-art technologies, algorithms and sensors. The aforementioned research is focused on improving already existing technologies in order to help humans perform tasks that would otherwise be impossible or dangerous or even tasks that are repetitive and prone to human error. ASVs are no exception, as they are more and more used in order to perform tasks related to the maritime environment. Recent technological advancements, the growing demand for unmanned vessels as a safer alternative and the increasing interest in using autonomous transportation are important factors that stimulated the design and development of new larger commercial vessels [9] along with other state-of-the-art technologies.

2.2 Situational Awareness

Autonomous mobile robots have very restricted or even nonexistent human intervention when performing tasks or simply moving through the environment, as such, these are very limited to the information given by the sensors mounted on them. If the robot is not capable of observing its surroundings correctly, it will fail to perform the desired task such as the estimation of an object

position [10]. Choosing the right number of sensors and what sensor modalities to use in mobile robots is a very complex task [11] and it is a topic that is widely researched. Although ASVs are the main focus of this dissertation, in addition to maritime related research, general mobile robots perception technologies and approaches to this problem will be discussed in this section, as these are more developed and extensively investigated in other types of vehicles, such as self driving cars.

2.2.1 Existing Sensors

Generally, state-of-the-art autonomous vehicles use the information provided by different sensors. Each of these sensors have their own advantages and disadvantages which can be complemented using sensor fusion techniques that will be further discussed in section 2.3. These sensor fusion techniques can use information provided by sensors of the same modality (homogeneous sensors), or by sensors of different modalities (heterogeneous sensors). For different objectives, it is important to use heterogeneous sensors such as vision, laser, radar and sonar [12], as each of these modalities provide different information.

Perception sensors can be separated in two different categories: proprioceptive and exteroceptive [11]. Proprioceptive sensors give information relative to the robot's internal state such as wheel position, motor speed and battery level and can be potentiometers, encoders, gyroscopes, inclinometers, compasses, among others [11]. Exteroceptive sensors are the most relevant to this work and they gather information relative to the environment and can be ultrasonic sensors, sonar sensors, IR sensitive sensors, distance sensors and GPS [11]. Sensors can also be separated in passive, when they measure the energy that naturally occur in the environment and active, when they provide their own energy [11].

When choosing what sensors should be used for a specific use case, there are many factors that should be taken into consideration such as: measurement range, resolution, linearity, bandwidth, error characterization, accuracy, repeatability, robustness to environmental changes, price, size, weight and power consumption. The most commonly used perception sensors in mobile robots include LiDARs, cameras, radars and sonars. There are many advantages and disadvantages in using these sensors and their performance is heavily related to the environment where the robot is navigating, weather conditions, light and time of the day. ASVs navigate in an aggressive environment where their stability is compromised due to the inherent unstable nature of the water and where rough weather conditions are a common occurrence. These factors, along with the ones already mentioned, should be taken into consideration when selecting which sensors the ASV should be equipped with.

During this section, the most relevant sensors in mobile robots will be presented and discussed, along with their advantages, disadvantages and fields of application.

A lot of the perception systems used in situational awareness include camera sensors because of their numerous advantages. Images that are captured by a visible spectrum color camera cover a broad area and capture information from objects that are far away from the sensor [4]. These

cameras can give important information about the appearance of the objects such as color, brightness and other relevant features such as texture can be extracted [13]. Other significant advantages include their low price when compared to LiDARs and their small size. Although the presented advantages are very useful when trying to make sense of the surrounding environment, just like the human eye, visible light systems completely depend on the light that is reflected from the objects, failing to operate as expected under certain conditions where the illumination is not enough, such as at night time or under rough weather. Outdoor mobile robots are heavily affected by these drawbacks, as poor illumination is a common occurrence, specially in maritime environments where wind and high waves are common. To compensate for these flaws, camera information can be fused with other sensors, such as LiDARs, in order to acquire better and more reliable information from the environment [4], as mentioned before, this topic, along with different sensor fusion methodologies will be further discussed in subsection 2.3. Other types of cameras, infrared (IR) cameras, are more reliable when operating in the aforementioned conditions as some of them are able to capture infrared information independently of the lighting conditions. IR cameras can be classified depending on the wavelength that they are designed to capture. Long-wave infrared (LWIR) cameras capture wavelengths between $8\mu\text{m}$ and $14\mu\text{m}$ [14], which allows them to capture thermal information, that is, the heat naturally emitted by objects. This thermal information does not depend on illumination, thus allowing them to detect and extract information even at night time. Short-wave infrared (SWIR) cameras capture wavelengths between $1\mu\text{m}$ and $2.5\mu\text{m}$ [14] and have the advantage that they have high sensitivity and wide dynamic range. The information provided by these two different IR cameras can be combined and “*results illustrate the complementary nature of the different spectral regions, provide insight into the strengths and pitfalls of joint versus modal analyses, and demonstrate improved hyperspectral mapping capabilities using the combined data*” [15]. Other IR cameras include near-infrared (NIR), medium-wave infrared (MWIR) and visible-near infrared (Vis-NIR) [16]. There are many systems that use cameras as an important sensor to capture information from the environment, these include: mobile robots in general, self driving cars, collaborative robots, automated guided vehicles (AGVs), ASVs, drones, medical imaging systems, mapping. Cameras are one of the most relevant sensors to perceive the environment and are used in well-known applications such as the Tesla Autopilot and full self-driving systems, XPeng’s Navigation Guided Pilot (NGP) and RAC Intellibus.

The different types of cameras presented were discussed and compared by N. Pinchon *et al.* (2016) [17]. The experiments were conducted during adverse weather and visibility conditions, such as rain, fog and snow, in order to compare the performance of the different cameras. Results demonstrated that LWIR cameras perform well when detecting objects and shapes even at night time with fog, but perform poorly when detecting lights, traffic signs and road marks. On the other hand, SWIR, NIR and Visible cameras cannot recognize vehicle’s shape under these circumstances but perform better than LWIR when detecting lights, traffic signs and road marks. These results are presented in figure 2.2.

Table 2.1: Comparison of performances of different cameras during the night time with fog [17].

ADAS FUNCTION	Camera Spectral band Night Fog, using headlights			
	VIS	NIR	SWIR	LWIR
Pedestrian, Bycycles, Animals, detection	+	++	++	++++
Vechile shape recognition	-	-	--	++
Vechile light detection	+	++	+++	--
Traffic signs recognition	+	++	-	--
Road marking detection	+	++	++	-

One specific scenario that was analysed was the identification of traffic signs. The results are interesting as they give a clear idea of some of the advantages and disadvantages of each camera, and the type of image that should be expected from them. From the table 2.1 it is possible to see that the SWIR sensor is not able to identify the traffic sign, even in good conditions, due to the similar reflectivity of the letters and the background. The NIR sensor obtained a better result than the visible camera and was able to identify the sign even with bad visibility. The LWIR camera was not able to identify the sign as the whole sign has the same temperature and the letters cannot be segmented from the background [17].

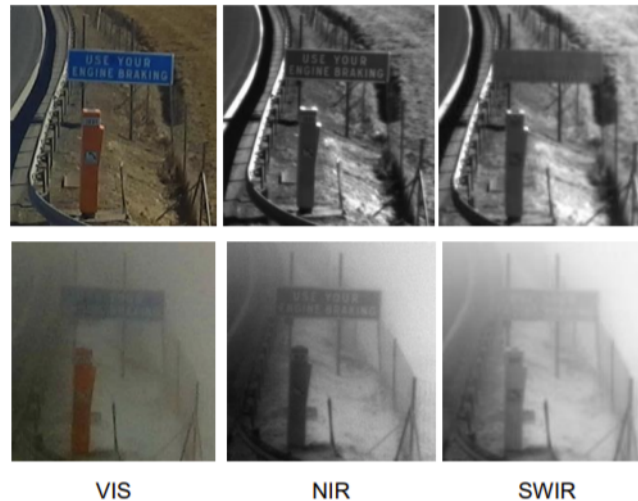


Figure 2.1: Images of traffic signs acquired in different conditions, the first row represents a sunny day and the second a foggy day with snow [17].

LiDAR is an active sensor that is often used to complement other sensors such as cameras, radars and ultrasonic sensors [18]. LiDARs can be classified depending on the type of information that they extract from the environment, 2D or 3D, and depending on the way that they are

constructed, rotary or solid state [19]. In order to gather information, LiDARs emit a laser to the environment and measure the distance to objects by calculating the time it takes for the light to reflect off objects and return to the photodetector. This laser is generated by a diode that emits light at near infrared wavelengths [20]. After scanning its surroundings, a point cloud is generated, which gives relevant 3D information about the environment. This information can be used in many fields of application such as autonomous navigation, terrain mapping, topographic surveying, emergency mapping and military [21]. The basic operation of a typical LiDAR system is represented in figure 2.2. LiDARs have numerous advantages, such as their rich and detailed spatial information, resulting in being, in most cases, the most accurate when solving spatial based problems, such as localization of a robot [22] or identifying the position of a certain object. Despite these advantages, LiDARs have a major drawback which is their high price, however, over the last few years their price has been reduced significantly, making them more accessible to the general public and car manufacturers. Other disadvantages include their low resolution, specially in cheaper LiDARs, which consequently gather sparser information from the environment.

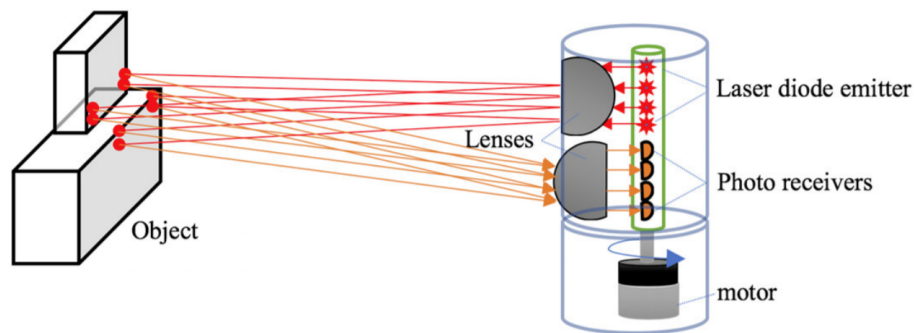


Figure 2.2: Simple example of how a 3D rotating LiDAR operates [19].

One of the most researched and used field of application is autonomous driving for obstacle detection and environmental perception. Some well known vehicles that use this technology include the Google self-driving car and Groupe RENAULT's car developed for the EVAPS project [20]. In 2007, the DARPA Grand Challenge proved the big potential of using LiDAR perception systems in autonomous driving, as the teams that placed in first and second place used a Velodyne HDL64 as their central sensor [20]. In most real-time systems, including Autonomous Driving Cars (ADCs), the information given by the sensors must be processed under a tight time constraint, as decisions made using that information will determine, for instance, if the car is able to avoid collision with a certain obstacle. LiDAR sensors typically produce a large amount of data, which should be processed efficiently to be used in real-time systems, this proves to be challenging and methods that use downsampling or feature extraction are used to mitigate this problem [22]. A downsampling solution was proposed in 2019, by O. Ervan and H. Temeltas [23], inspired by tensor voting [24], where the latter was adapted to downsample 3D LiDAR point clouds.

There are other approaches to extract and process information given by LiDARs that are used and vary depending on the desired output. The 3D point clouds information is often processed

using machine learning algorithms, such as the work developed by E. Maltezos *et al.* (2019) [25], where LiDAR information is used in deep convolutional neural networks in order to classify buildings. Other machine learning applications and methods will be discussed in section 2.3.2.1 focusing on methods to fuse information from LiDARs and other sensors. Just like cameras, LiDARs face challenging problems when it comes to environmental conditions such as bright sun, fog, rain, dirt and spray [18]. According to Peynot's *et al.* [26] results obtained in 2009, LiDAR sensors suffer a considerable attenuation when compared to radar sensors in such adverse conditions. These results also showed that heavy dust particles are detected in mid-air, which prevents the sensor from detecting the points of interest behind the dust cloud. This is a very researched topic and there is a significant amount of studies where the performance of LiDAR sensors is tested in different weather conditions such as rain, by A. Filgueira *et al.* (2017) [27] and fog, by S. Hasirlioglu *et al.* (2017) [28]. In maritime environments, just like autonomous driving cars, LiDARs can be used in order to identify objects such as other vessels, docks, floating obstacles or distances to the environment itself. Taking into consideration the rich information provided by these sensors, along with all the aforementioned advantages, this is a sensor that will be used, fused with the information provided by cameras, in order to accomplish the objective of this dissertation.

Radar sensor is a relatively old technology that has been used in maritime related missions for a long time, as well as other applications such as aerial or terrestrial threat detection systems [19]. These sensors measure the distance between the emitter and objects by calculating the time of flight of the emitted and received signal. They allow the detection of the distance to objects, as well as the speed and direction of the objects [19]. As a fundamental part in navigation, radars are used to identify, track and position vessels in order to safely move across the sea. These sensors are classified in the X-band (10 GHz) and in the S-band (3GHz). X-band based radars have a better resolution and usually produce a sharper image, whereas S-band based radars are used in adverse weather conditions such as rain or fog. Ka- and W-band radars operate in higher frequencies and are used in civilian, automotive and military applications, however, they are only used for short and medium ranged operations [29]. There are various systems that use this kind of technology such as Yara Birkeland autonomous container ship and, in more recent years, the aforementioned Tesla Autopilot system.

2.3 Sensor Fusion

As mentioned and discussed in section 2.2.1, there is a vast variety of sensors that are used for perception and situational awareness in state-of-the-art systems. These sensors perform well in specific situations and setups, but fail to produce the desired outputs when faced with unfavourable conditions. To achieve the best situational awareness reliability, the capabilities of different sensor modalities can be combined, complementing each sensor's weaknesses and errors, preventing the occurrence of false positives and false negatives, leading to a better overall performance of the perception system [30]. Taking into consideration the advantages and disadvantages of the sensors discussed in section 2.2.1, the different types of sensors that are commonly used in autonomous

systems and maritime related applications are directly compared in table 2.2. In 2019, Francisca Rosique *et al.* [19] quantified the different features of each sensor on four scores, being: 0-none, 1-low, 2-medium and 3-high. The desired score for the first seven features is 3, while the last four are preferably 0. As can be concluded from the table 2.2, there are characteristics from the rotating 3D LiDAR and the VIS camera that complement each other, such as resolution and color perception.

Table 2.2: Direct comparison of different sensors used in perception systems [19].

	RADAR	3D LiDAR		Cameras		
		Rotating	Solid State	VIS	IR	ToF
Field of View	2	3	2	3	3	2
Range	3	3	3	2	3	2
Accuracy	2	3	3	3	2	2
Frame rate	2	2	2	2	3	3
Resolution	1	2	2	3	1	1
Colour perception	0	1	2	3	1	1
Visibility	1	3	2	2	2	2
Size	1	2	1	1	1	1
Weather affections	1	2	2	3	1	3
Maintenance	1	2	1	2	2	2
Price	2	3	1	1	3	2

In 1997, D. L. Hall and J. Llinas [31] stated that “*data fusion techniques combine data from multiple sensors, and related information from associated databases, to achieve improved accuracies and more specific inferences than could be achieved by the use of a single sensor alone*”, when talking about sensor fusion, this means higher quality and more relevant information that allows a better perception of the environment. Data fusion is a multidisciplinary area and there are many different data fusion techniques that were researched and developed over the years. In a review conducted by Federico Castanedo [32] in 2013, these data fusion methods were classified based on different criteria: (1) relations between the input data sources which can be defined as complementary, redundant or cooperative data; (2) according to the input/output data types and nature; (3) abstraction level of the employed data which can be raw measurements, signals and characteristics or decisions; (4) based on different data fusion levels defined by the JDL; (5) depending on the architecture type, which can be centralized, decentralized or distributed. The different approaches presented in 2.3.1 can be classified according to the presented criteria, having different types of input data depending on the sensors used, different output data types depending on the objective of the solution, for example a depth map or a classification. These approaches also present different types of architectures, where some are more centralized oriented and others more distributed.

The fusion of different radars and visual sensors, including HD and IR thermal cameras proved to be a good solution in maritime environments, as “*radars enable easy tracking of target distance and can provide the required tolerance to bad weather*” [30], and cameras augment the data provided by radars by giving different and detailed information of the detected objects, thus increasing

the detection robustness. On the other hand, IR cameras can detect objects even in poor lighting conditions, while visual HD cameras can easily segment objects and obstacles present in the water.

Taking into consideration the advantages and disadvantages of each sensor, different state-of-the-art methods and approaches to sensor fusion for different applications such as autonomous driving, object classification and depth estimation will be discussed and analyzed in this section. Machine learning based algorithms to fuse sensor information will be extensively discussed in section 2.3.2.1, as it is a focus in this dissertation.

2.3.1 Conventional Approaches

In this section, various works that fuse data from different sensors using a conventional approach, that is, methods that don't use machine learning techniques to help process the information will be presented. These conventional approaches are often based on probabilistic methods that deal with the inherent imperfection of data, relying on probabilistic distribution functions to model data uncertainty [33].

A method that fuses information from a millimeter-wave (MMW) radar and a single camera, with the objective of detecting vehicles, was presented by X. Wang *et al.* [34] in 2016. This work presents a collaborative fusion approach in order to achieve a good balance between computational efficiency and vehicle detection accuracy. The MMW radar starts by detecting on-road vehicles, transmitting the size and location of the region of interest (ROI) to the images captured by the camera. Based on this region of interest, the vision processing module creates a square boundary in the image frame and applies an active contour method to detect vehicles within the boundary. This information is then used to generate a trajectory which is compared with the trajectory generated by the radar processing unit to confirm whether the detection and tracking are valid. This method resulted in a 92.36% detection rate and a 0% false alarm rate, using real-world experimental data.

Fusing data provided by LiDAR and camera sensors is a common practice as LiDARs can provide a 3D but sparse point cloud of the environment, which can be used for obstacle detection, however, it lacks relevant information such as color and texture, information that can easily be extracted from normal cameras. There are numerous works that combine the information provided by these two sensors, applied to a vast number of applications with different objectives and outputs. In 2019, L. Stanislas and M. Dunbabin. [35] presented a work regarding sensor fusion using a LiDAR, a camera and a radar. The work developed used data collected from the 2014 and 2016 Maritime RobotX Challenge, which is an international robotics competition where ASVs perform different and complex marine related tasks, simulating real-world scenarios. In this work, the obstacle classification module consists of a LiDAR and a camera. As stated in this work, "*LiDAR intensity maximum, mean, and standard deviation values are appearance features computed statistically for each map cell based on the set of sensor points contained in that cell*" [35]. These features were used in order to differentiate the object's materials. Other structural features were used such as height, width and aspect ratio, in order to differentiate objects of different shapes and sizes. A classification was performed on these features, using the Random Forest algorithm [36]. As these features lack color, the information provided by the camera was used in order to create

bounding boxes around high-confidence object areas, labeling them. These labels were then fused with transformed 3D points provided by the LiDAR, resulting in a complete semantic representation on the various obstacles.

Although a significant amount of the applications are object classification and scene parsing, LiDAR and camera sensor fusion can be used in order to produce a dense depth map which gives information about the distances of the surfaces of objects from a certain viewpoint. Machine learning approaches are often used to achieve this, and will be discussed in section 2.3.2.1. In 2017, H. Courtois and N. Aouf [37] proposed a method that fuses information from a stereo camera and a LiDAR to produce a depth map. This method consists on using the stereo image and the LiDAR scanner to acquire a sparse point cloud of known depths, while the rest of the image's depths are estimated using a modified bilateral filter in a Bayesian framework [38] to interpolate the missing disparities. The stereo image is then used to improve each estimation.

A different approach was presented in 2019 by Rayyan Akhtar *et al.* [39] where a Velodyne VLP-16 LiDAR and a monocular camera were used to solve the same problem. In this work, the color information provided by the monocular camera was integrated with the LiDAR data in order to obtain a 3D model where depth information was assigned to the image pixels, resulting in a depth map. Since the information from the LiDAR is very sparse and the resolution of the images captured by the camera is bigger than the LiDAR data, Rayyan “*proposed a method to interpolate the 2D depth map data to completely fill the empty locations in the depth map*” [39]. This method is based on the covariance matrix, where depth values associated with empty locations in the depth map are obtained taking into consideration the value of the covariance function in the covariance matrix. This method produced better results than their previous approach that used Gaussian Process Regression (GPR), obtaining a better run time and interpolation result, allowing this method to be used in real-time applications.

In 2019, Zhenshan Bao *et al.* [40] proposed a method where the information from a stereo and a ToF camera were fused in order to produce a highly accurate depth map. The first step consists of using an energy cost function with the objective of using the depth map obtained by the ToF camera to guide the stereo matching. The energy cost function is slightly different from other approaches, as it is applied to each ToF depth pixel, obtaining lower values if the estimated depth is closer to the correct depth and obtaining higher values otherwise. The second step consists on designing a weight function for pixel-level fusion and according to their credibility functions. Experiments showed that the this approach was able to achieve good results with high accuracy.

Current LiDAR and stereo camera based solutions typically face limited range issues. In an attempt to bypass this range limitation, K. Zhang *et al.* (2020) [41] proposed a method that takes advantage of a three-camera system that utilizes small field of view cameras and outputs a dense depth map with acceptable accuracy for objects at long distances that are usually not covered by a significant amount of commercial LiDARs. This approach not only allows long range dense depth sensing but also is a more cost-effective solution, as LiDARs are a very expensive alternative. The method developed also attempts to resolve geometrical ambiguities typically present in stereo systems that only use two small field of view cameras, by adding a third camera and using a novel

depth disambiguation algorithm. The method consists on three modules where the first one is a pseudo-rectification that transforms the image provided by the left and right camera with affine warps in order to form a pseudo-stereo pair. After this step, a standard stereo matching algorithm is used to form a disparity map with an unknown constant offset cause by the pseudo-rectification. This ambiguity is then resolved using the image provided by the third camera, converting the disparity map to a depth map.

A different method to obtain a dense depth map was proposed in 2020, by X. Zhao *et al.* [42]. In order to obtain the depth map, Xiaoming fused the information provided by a LiDAR, a time of flight camera and a binocular camera. The first step was to calibrate the sensors to a uniform coordinate system, this was done by using a calibration method [43] to calibrate the LiDAR and the binocular camera, as well as a new calibration method developed to calibrate the ToF camera and the binocular camera. In this work, the depth information of the binocular camera was calculated by binocular stereo vision from the left-right RGB images and was fused with the sparse but accurate depth data from the LiDAR and with the dense and relatively reliable data from the time of flight camera. This depth fusion was done based on gradient priors which are more reliable than the gaussian distribution counterpart, as they are more consistent with the depth distribution and image structure features of the real scene.

In 2020, H. H. Cholakkal *et al.* [44] proposed a LiDAR and stereo camera fusion with a novel post-processing method for depth estimation, based on a patch-wise depth correction method hoping to preserve object's boundaries and edges in different scenarios. The general idea of this solution is to use the sparse but accurate 3D information from the LiDAR as a reference to improve the depth estimation obtained by the stereo camera. A Bilateral Filter is then applied in order to restore the continuity of the depth map. Just like the previous work [42], as expected from a sensor fusion algorithm, a calibration method was used to determine the sensor's relative pose. In this case, a calibration solution based on [45] was implemented, where a planar board with four circular holes is used. The first step of the depth correction algorithm consists on projecting the 3D point cloud information acquired from the LiDAR into the left camera frame, using the transformation parameters that were obtained from the calibration. These points were then filtered and the points that were occluded in the camera line of vision were removed. The second step consists of merging the sparse depth image obtained in the first step with the depth map provided by the stereo camera, following a patch-wise correction approach. The final step consists on creating a dense depth map using a bilateral filter. The results of this method were tested using the well-known KITTI dataset [7], demonstrating the algorithm's accuracy.

2.3.2 Machine Learning Approaches

As discussed in section 2.3, there are many advantages in using sensor fusion, as one sensor compensates for the weaknesses of others, improving the overall performance and reliability of the result. There are many studies and methods regarding sensor fusion that use a machine learning approach. In this section, the fundamentals of machine learning and some well-known neural

networks will be presented, along with important works that focus on sensor fusion techniques that predict dense depth maps [2.3.2.1](#).

Machine learning (ML) is a part of artificial intelligence and is a relevant topic that has been getting a lot of attention in the past few years. Machine learning algorithms are dynamic and learn from data, improving their performance over time. These algorithms receive data as input and make decisions and predictions based on patterns and features that they learned to recognize, without being manually programmed to do such task. In the training phase, these algorithms take training data as input and adjust their internal parameters in order to get better predictions. Machine learning algorithms also help interpret and process data in a way that would otherwise be impossible for a human, as they can find patterns and output predictions in sizable amounts of data. In recent years, machine learning based algorithms are getting a lot of attention due to the increasing availability of datasets, fast growth in computer power and new and better algorithms [\[6\]](#). These algorithms are used in a vast set of applications such as speech recognition, text analysis, pattern recognition, planning, image captioning, web search, spam filtering, fraud detection [\[46\]](#), medical diagnosis, military and image recognition. During this section, different approaches, commonly used learning methods and some of the most well-known neural network architectures will be discussed.

Machine learning methods can be categorised in three main categories, depending on how they learn: supervised, unsupervised and semi-supervised machine learning. Supervised machine learning use labeled data in order to train itself. For instance, a supervised machine learning algorithm designed to discriminate dogs from cats, receive images as input along with a label that says whether a specific image is a dog or a cat. After processing an image, the algorithm outputs a prediction which is then compared with the desired output in order to adapt its internal parameters to make better predictions in the future. Compared to other methods, supervised machine learning requires less data to learn, however, labeled data is more expensive and harder to get [\[47\]](#). On the other hand, unsupervised machine learning receives a large amount of unlabeled data and find patterns and relationships, labeling or classifying data that would take too long for humans. Semi-supervised learning uses a small amount of labeled data to help classify and interpret a larger amount of data that is not labeled, mitigating the problem of not having enough labeled data.

A neural network is a machine learning method that is inspired by how the brain works and is formed by neurons and connections between neurons that perform computations on their inputs and transmit the output to the next neuron [\[47\]](#). A neural network structure is composed by different layers where the first layer receives the input data, transmits this information to the intermediary hidden layers which in turn transmits the information to the output layer that outputs the result [\[47\]](#). Neurons determine their outputs by calculating the weighted sum of their inputs and applying an activation function that transforms this sum to a number that falls between a lower limit and an upper limit. There are many different activation functions that are used, including sigmoid that transforms the input to a value between 0 and 1; *tanh* that transforms the input to a value between -1 and 1 and ReLU that is 0 when the input is a negative number and leaves the input as it is when it is a positive number.

During the training phase, in order to know how close the network's prediction is to the desired value, a loss function is calculated taking into consideration how far the output is to the ground truth. Based on the result of the loss function, the weights of the network are updated in the direction that minimizes the loss. There are different commonly used loss functions and they are chosen based on the type of problem that the network is trying to solve, they include mean square error (MSE), typically used for regression problems, and cross-entropy that is used for classification problems. There is a vast number of well-known neural network architectures already implemented that apply some of the principles previously mentioned, some of these architectures will be presented and briefly discussed below, starting with general neural network structures such as convolutional neural networks and recurrent neural networks, ending with specific network architectures developed over the years.

A Convolutional Neural Network (CNN) is a deep learning model that is inspired by how the animal visual cortex works and is typically used for data that is organized in a grid pattern, such as images [48]. It is specifically designed to learn spatial features and patterns that are present in the input data, allowing, for instance, to recognize shapes in images. CNNs are typically built using three different types of layers: convolution, pooling and fully connected (FC) layers [48]. A typical CNN structure, represented in figure 2.3, starts with convolutional layers that use a kernel that is applied to each position of the image, scanning a small number of pixels at a time and extracting features that could be present anywhere in the image. Convolutional layers are usually followed by a pooling layer which reduces the amount of information present in each feature. Just like the convolutional layer, they use kernels that are applied to different parts of the image, however, the operations are different and focus on determining a single value out of the pixels present in each iteration, thus reducing the size of the image. The most common pooling methods include average pooling and max pooling, where the average value and the max value of the pixels is chosen respectively. The final step is to take the feature map produced by the convolutional layers and map them to an output, such as classification [48], which is done by using a fully connected layer.

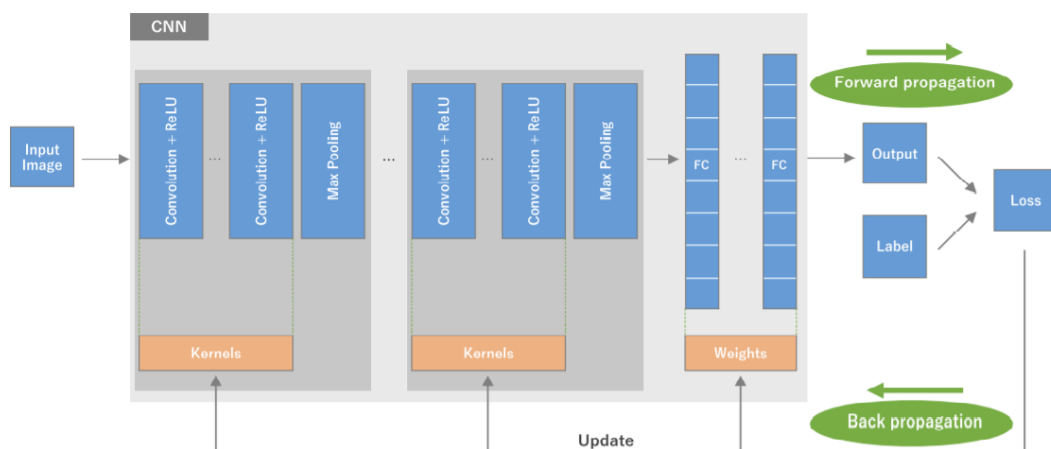


Figure 2.3: Basic structure of a convolutional neural network [48].

LeNet is one of the first convolutional neural networks, proposed in 1989, by Yann Lecun *et al.* [49]. Figure 2.4 represents the basic structure of LeNet-5 that was used for digits recognition.

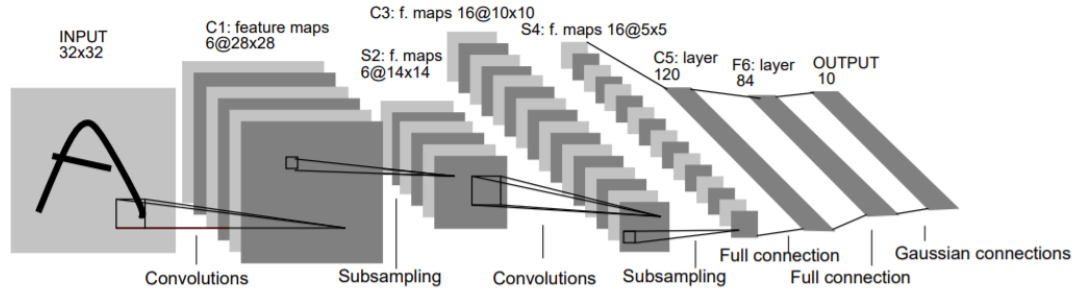


Figure 2.4: Architecture of LeNet-5, used for digits recognition [50].

As an architecture that was developed in the early days of machine learning, it helped understand how image features are present across images and how convolutional layers are useful to extract features using a small number of learnable parameters. The structure of LeNet-5 is similar to the general convolutional neural network structure previously discussed in this section, using convolutional layers to extract spatial features, average pooling layers to subsample the data and obtain averages of feature maps and fully connected layers to obtain the final classification.

VGG [51] is a deep convolutional neural network that uses all the already discussed layers present in CNNs, and uses a large amount of convolutional layers when compared to LeNet. These convolutional layers use a sequence of multiple small 3x3 filters that extract similar features to the ones extracted by larger filters, with the advantage of having less weights. By having a smaller filter size of 3x3, it is necessary to stack three of them to have the same result of a single 7x7 filter, resulting in the need of more non-linear rectification layers, which in turn “*makes the decision function more discriminative*” [51]. As a deep convolutional neural network with a significant amount of small convolutional layers, one of the drawbacks of using this architecture is the very heavy training process. The general ideas implemented in VGG are also used in other modern network architectures that will be presented in this section: Inception and ResNet.

In 2015, Christian Szegedy *et al.* [52] proposed a deep convolution neural network architecture named Inception. The main objective of this architecture was to increase the depth and width of the network while keeping the computer processing power as low as possible [52] and achieving state-of-the-art performance. The idea behind the Inception architecture, represented in figure 2.5, is “*based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components.*” [52].

This structure is a combination of parallel convolutions of sizes 1x1, 3x3 and 5x5. As seen in 2.5, before each expensive 3x3 and 5x5 convolution blocks, a 1x1 convolution is used in order to reduce the size of the features, thus reducing the computational power needed to perform the bigger convolutions.

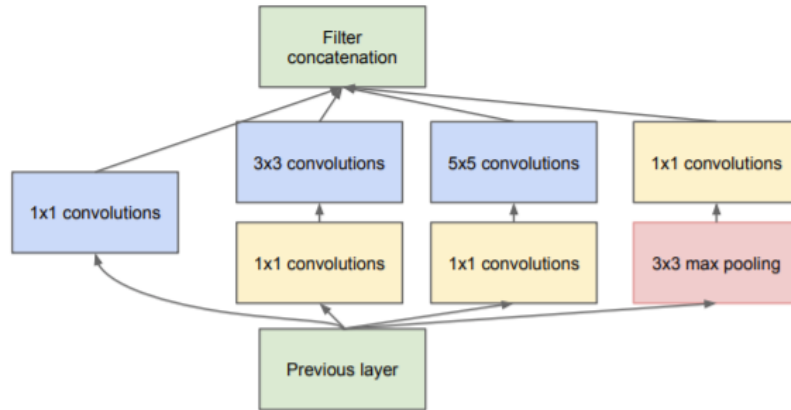


Figure 2.5: Structure of the Inception module. [52].

Generally speaking, an Inception network is composed by multiple modules stacked on top of each other, with some max-pooling layers with stride 2 to reduce the resolution of the grid to half [52]. This network design is also very useful because *“it aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously”* [52]. This concept proved to be very successful, achieving state-of-the-art results in well-known datasets, so much so that these ideas are used in more recent architectures, such as ResNet. In 2015, Christian Szedy *et al.* [53], the same authors, proposed a newer version of their previous work, where just like VGG, 5x5 convolutions were replaced by multiple 3x3 convolutions. Some of the general design principles of this approach include the balancement of the number of filters used on each stage and the increase of the number of features and width of the layer when depth is increased.

In 2016, Kaiming He *et al.* [54] presented a *“residual learning framework to ease the training of networks that are substantially deeper than those used previously”* [54].

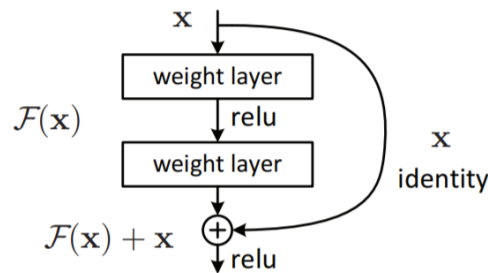


Figure 2.6: A building block of the residual learning [54].

This network is built based on simple blocks, shown in figure 2.6, that use a special connection that skips convolutions and directly feeds the input to the next residual block. This type of network starts with a 7x7 convolutional layer, followed by a pooling layer that reduces the size, which in turn is followed by multiple blocks similar to the one presented in figure 2.6, ending with an average pool and a fully connected layer with softmax function. For deeper networks, the building

block is changed and uses the same principle as Inception, by applying a 1×1 convolution that reduces the size, followed by a 3×3 convolution, ending with another 1×1 convolution to increase the number of features, thus reducing the computation needed while keeping a good combination of features.

In 2017, G. Huang *et al.* [55] presented the Dense Convolutional Neural Network (DenseNet). In this network, every layer has a connection to all subsequent layers, as such, each layers receives the feature maps from all preceding layers. This allows the features learned from previous layers to be reused, leading to more compact and thinner networks [55]. When compared with other networks such as ResNet or VGG, this approach uses fewer parameters, resulting in a higher computational and memory efficiency, while achieving good results [55].

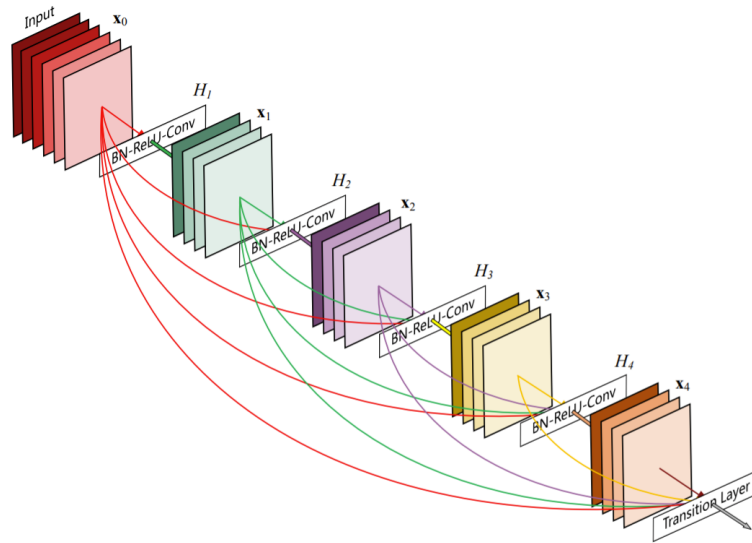


Figure 2.7: Basic structure of a 5-layer dense block [55]

The proposed architecture is composed of multiple dense blocks, figure 2.7, connected by transition layers that are responsible for down-sampling the data. These transition layers consist of a batch normalization layer, followed by a 1×1 convolution layer, ending with an average pooling layer [55].

In 2018, Mark Sandler *et al.* [56] presented the MobileNetV2. This network introduces a novel inverted residual with linear bottleneck, that “takes as an input a low-dimensional compressed representation which is first expanded to high dimension and filtered with a lightweight depthwise convolution.” [56]. The features are then projected back to their low-dimensional representation, using linear convolutions [56]. Figure 3.20 represents the bottleneck structure.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d, ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwse s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Figure 2.8: The bottleneck residual block, transforming from k channels to k' channels, expansion factor t and stride s [56].

The MobileNetV2 network starts with a fully convolutional layer, followed by 19 bottleneck blocks as represented in figure 2.8, ending with a sequence of a convolutional layer, an average-pool layer and a final convolutional layer [56].

2.3.2.1 Machine Learning Architectures

There are a lot of sensor fusion methods and techniques that take advantage of machine learning algorithms in order to improve their performance. During the rest of this section, some of these methods will be presented and discussed.

When talking about sensor fusion, there are three fusion categories to take into consideration: early fusion, late fusion and cross fusion. Late fusion is simple and is one of the most used fusion methods [57]. In this approach, the data is merged using the sum or average, after processing the information from each sensor individually “*by powerful targeted approaches, tailored to the specific properties of the particular modality*” [57]. This approach has a drawback, as it is not able to take advantage of possible correlations between the different modalities [57]. On the other hand, early fusion merges the information provided by different sensor modalities in the beginning of the process. This information can be fused directly without any pre-processing and is fused by merging the individual data into a joint representation. This approach has the advantage of exploring the correlation between data and the advantage of only using one learning phase, while having the disadvantage of the difficulty to combine the information into a single representation [58]. Cross fusion consists on initially processing the information from each sensor individually, followed by a merge of this information, where the features from one sensor are fused with the features from the other sensor and vice-versa.

In 2019, C. Fu *et al.* [59] adopted a sensor fusion technique where information from a LiDAR and a monocular RGB camera were used to estimate a dense depth map, using a deep neural network. In this work, the early-fusion technique was used, where the RGB image and the sparse point cloud were combined in an early stage. The proposed network architecture uses an encoder-decoder approach where, in this case, the well-known ResNet50 model was tuned and used as the feature encoder. The proposed architecture has 4 channels in the input layer of the network, which is the result of the early data fusion. In this architecture, three skip connections are used to pass feature tensors from the encoding residual blocks to the decoding blocks, which improves the prediction by propagating more context information to higher resolution layers [59]. In addition, these skip connections “*also forward missing detailed features such as object boundaries to*

higher-resolution decoding blocks, which mirror the input” [59]. Another important part of this architecture is the proposed Residual Up-Projection Block which is present in the decoder network. Common Up-Pulling layers usually increase the spatial resolution taking into consideration the features from nearby patches, using bilinear interpolation or nearest neighbour interpolation mechanisms for semantic segmentation [59]. However, when facing a depth completion problem, this Up-Sampling layer is not enough because it does not consider important information such as geometry and semantic information [59]. The proposed block in this work is based on the de-convolution layer in [60], which up-samples the features by using Up-Pooling kernels, with the addition of a residual into the block [59]. To validate this architecture, it was tested on the KITTY [7] and NYUDepthV2 [61] datasets, obtaining positive results when compared to other state-of-the-art methods at that time.

Wouter Gansbeke *et al.* (2019) [62] also proposed a solution using a LiDAR and an RGB camera. Unlike C. Fu *et al.* [59], this work proposes a combination of early and late fusion. The early fusion “*takes the form of a guidance map for our local network extracted from global information*” [62]. The proposed method takes the 3D point cloud provided by the LiDAR and projects them to a 2D plane, and addresses the depth completion problem by interpreting it as a regression problem. This approach uses the information from the ground truth to train the CNN, encoding the missing LiDAR values with zeros [62]. The target values are calculated using temporal information and semi-global matching [63], obtaining a semi-sparse ground truth. To overcome the difficulties that arise from having a sparse input and a semi-sparse ground truth, the CNN takes advantage of global guidance information in order to correct artifacts and to properly upsample the input data [62].

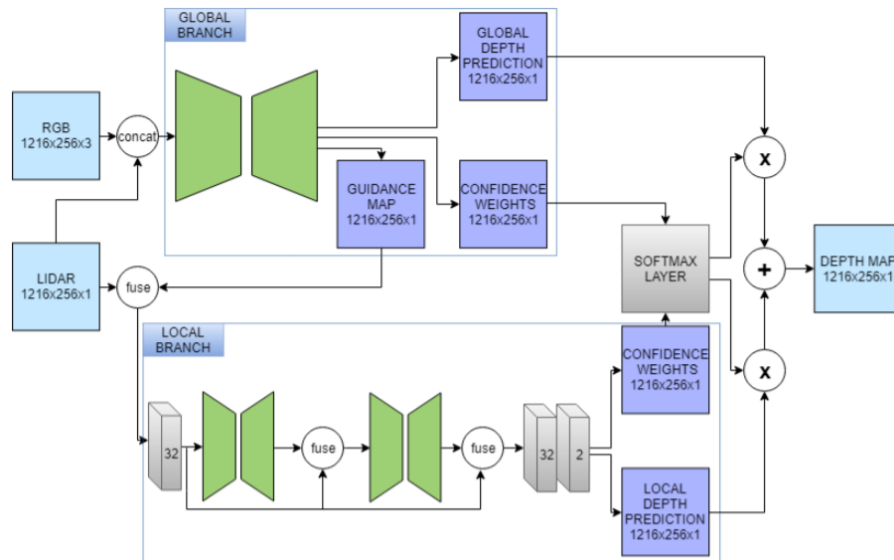


Figure 2.9: Framework of the solution proposed by Wouter Gansbeke *et al.* [62].

The architecture of this solution, presented in figure 2.9, consists of two branches: the global branch and the local branch. The objective of the global branch is to extract global information

to compensate for mistakes made in the LiDAR input frames, by using that information to help the local network detect these errors and reconstruct more accurately the sparse information [62]. In addition, the global branch detects objects and structures in the frame that are likely to have the same depth and uses the RGB data to prevent sudden variations in the LiDAR input, taking into consideration that color information can be used to detect borders of objects. On the other hand, the local branch is responsible for examining the information provided by the LiDAR and perform local up-sampling. Assuming that the LiDAR data can be noisy, in order to mitigate this problem, the LiDAR map is fused with a guidance map provided by the global branch which is then the input of the local branch. The network used in the global branch is an encoder-decoder approach, based on ERFNet [64] while the network used in the local branch is a stacked hourglass network. The ERFNet-based network was decided as it can be used in real-time situations and performs well in city landscapes, which is the target of this architecture. As seen in figure 2.9, the global guidance map provided by the global branch was fused with the sparse LiDAR point cloud, taking advantage of global information. This approach can be seen as an early fusion which guides the local branch [62]. Besides the guidance map, the global branch also outputs a global depth prediction map and a confidence map. On the other hand, the local branch only outputs a local depth prediction map and a confidence map. The final depth map prediction is constructed by “multiplying the confidence map with its depth map and adding the predictions from both networks” [62].

In 2019, E. Romera *et al.* [65] proposed a different approach where the information from a LiDAR and a stereo camera were fused in order to produce a depth map. Stereo cameras rely on geometric correspondences between two images in different viewing angles, producing less ambiguous depth estimates, in addition, information from the stereo camera can be well aligned with the 3D LiDAR measurements [65]. In this work, the sparse 3D LiDAR data is fused into a stereo matching network using Input Fusion and Conditional Cost Volume Norm (CCVNorm).

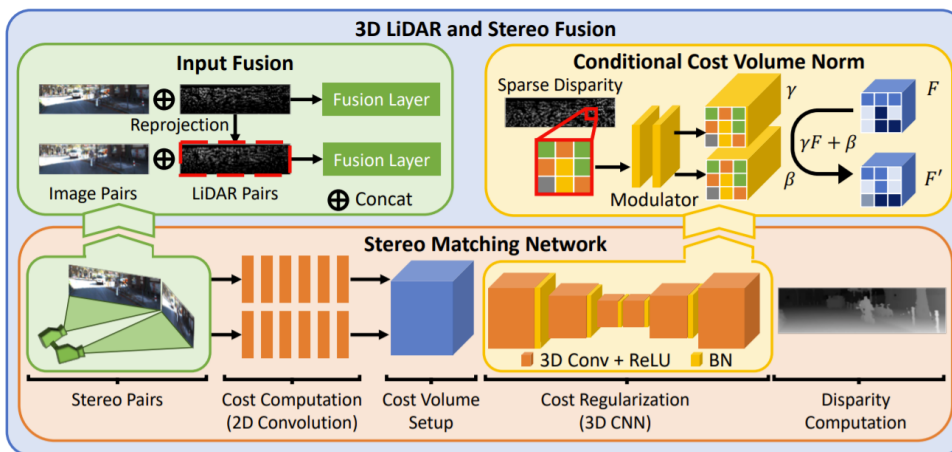


Figure 2.10: Framework of the solution proposed by E. Romera *et al.* [65].

This stereo matching network, along with the general architecture of this approach can be seen in figure 2.10. The Input Fusion technique is used to combine the geometric information

from sparse LiDAR points with the RGB images, serving as input for the Cost Computation phase, which is used to learn joint feature representations [65]. On the other hand, the CCVNorm is used to encode the LiDAR information into features of the 4D cost volume, which is formed by “aggregating the deep features extracted from the left-image with their corresponding ones from the right-image across each disparity level, where the size of cost volume is 4-dimensional $C \times H \times W \times D$ (i.e., feature size $\times$ height $\times$ width $\times$ disparity)” [65]. The proposed architecture starts with a block that learns the deep feature extracted from the rectified left-right stereo pair, in order to compute the cost of stereo matching [65]. The cost volume is then calculated by the aforementioned method, followed by a sequence of 3D convolutions that regularizes the cost volume, ending with a final disparity computation where “the final disparity estimation is carried out by regression with respect to the output volume of 3D-CNN along the D dimension” [65].

In 2021, J. Tang *et al.* [66] proposed another method that fuses information from a LiDAR and an RGB camera. In this approach, an encoder-decoder architecture, figure 2.11, was also adopted. The dense depth map is produced by first creating a sparse depth map from projecting the LiDAR points to the image plane with the help of calibration parameters and then, with an RGB image as guidance reference, the sparse map is upgraded to a dense map. [66]. A novel guided network inspired by guided image filtering was designed in order to predict kernel weights with the help of the guidance image. These kernels are applied to extract relevant depth image features, generating content-dependant and spatially-variant kernels that are used for multi-modal feature fusion [66]. A convolution factorization method was also developed in order to reduce the required computational power and memory usage.

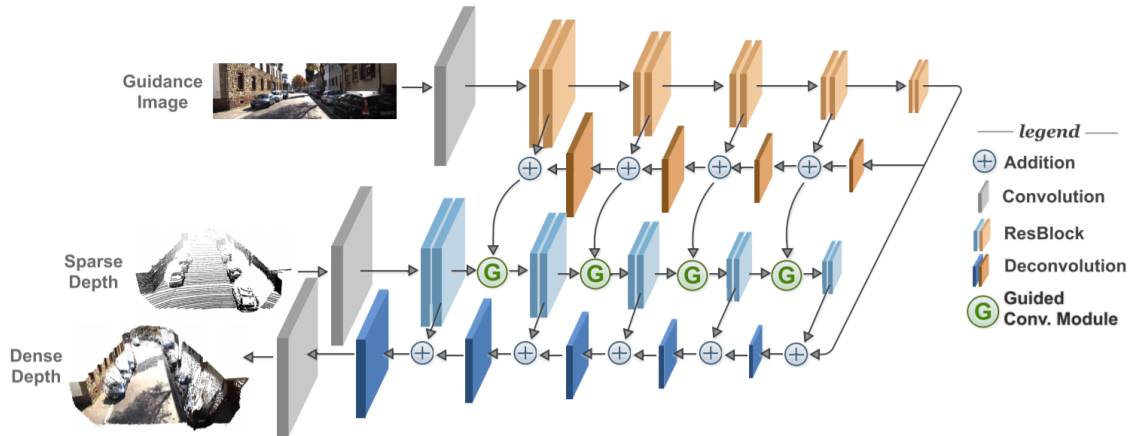


Figure 2.11: Encoder-decoder architecture proposed by J. Tang *et al.* [66].

The proposed architecture is divided in two encoder-decoder sub-networks present in figure 2.11, where GuideNet is represented in orange and DepthNet is represented in blue. The GuideNet is responsible for learning “hierarchical feature representations with both low-level and high-level information from RGB image” [66]. These image features are used to automatically generate the aforementioned kernels. The DepthNet takes the LiDAR depth image and fuses it with the hierarchical features extracted from the guided convolution module. The encoders of both GuideNet

and DepthNet use ResNet blocks.

In 2020, Jinsun Park *et al.* [67] also proposed a method to fuse the information provided by a LiDAR and an RGB camera to create a depth map. In this work, the proposed network initially estimates non-local neighbours and their affinities of each pixel using the input RGB and sparse images, as well as an initial depth map with pixel-wise confidences [67]. This initial depth prediction is then refined using its confidences and non-local spatial propagation procedure that is based on the previously estimated non-local neighbors and their affinities [67]. This makes the proposed algorithm specially robust when it comes to the mixed-depth problem that occurs on depth boundaries, which is a common occurrence in other depth estimation algorithms.

The proposed architecture, presented in figure 2.12, uses an encoder-decoder approach based on the residual learning method. This block is responsible for extracting high-level features from the RGB and sparse depth images. On the other hand, the iterative non-local spatial propagation block is responsible for estimating missing values and refining less confident values by propagation neighbors using their corresponding similarities.

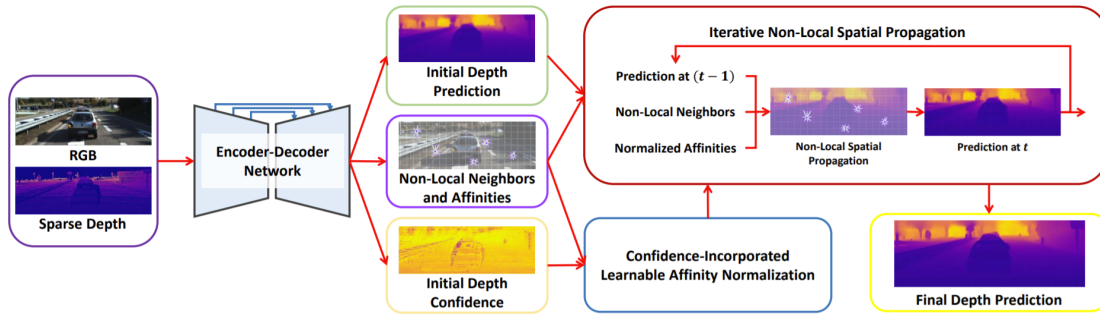


Figure 2.12: Depth completion architecture proposed by Jinsun Park *et al.* [67].

In 2020, Xinjing Cheng *et al.* [68] proposed another method that predicts a dense depth map from an RGB image and a sparse depth map. In this work, an adapted version of the Convolutional Spatial Propagation Network (CSPN) was developed.

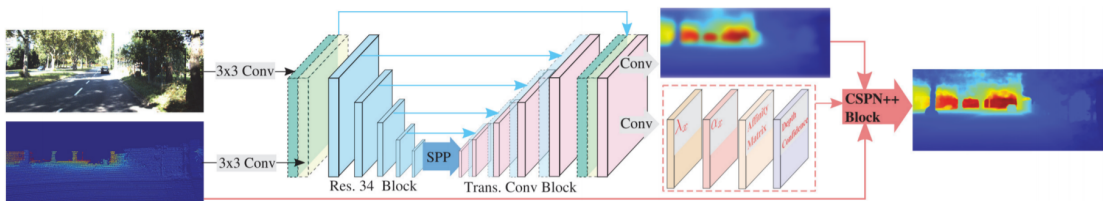


Figure 2.13: Depth completion architecture proposed by Xinjing Cheng *et al.* [68].

This network recovers structural details of the scene and the adapted version improves its efficiency and effectiveness by learning adaptive convolutional kernel sizes and number of iterations for the propagation [68]. This allows a dynamic context and computational resources needed for each pixel. These two hyper-parameters are learned and solve an architecture selection problem where different predefined configurations of kernel sizes and normal of iterations can be chosen from. The proposed architecture is presented in figure 2.13.

Shanshan Zhao *et al.* (2020) [69] proposed a different method to solve the same problem. In this work, a graph propagation method was adopted to capture the observed spatial contexts. From the observed pixels, multiple graphs at different scales were formed. Due to the different graph structures that vary from sample to sample, an attention mechanism was applied on the propagation in order to model the contextual information adaptively [69]. Since the input data is multi-modal, the graph propagation was exploited on each modality to extract multi-modal representations. The proposed architecture, represented in figure 2.14, is based on an encoder-decoder structure.

In the encoding stage, multi-scale multimodal features are extracted using multiple Co-Attention Guided Graph Propagation (CGPM), represented in blue, composed of residual connections and a co-attention guided path propagation module [69]. Furthermore, this stage is responsible for learning adaptive attention weights from depth features, RGB features and spatial locations [69]. In the decoding stage, multimodal features are fused progressively by using multiple Symmetric Gated Fusion (SGFM) blocks represented in red. The output is then calculated using the dense maps and confidence maps that were produced by the decoder or predicted using the intermediate fused feature maps [69].

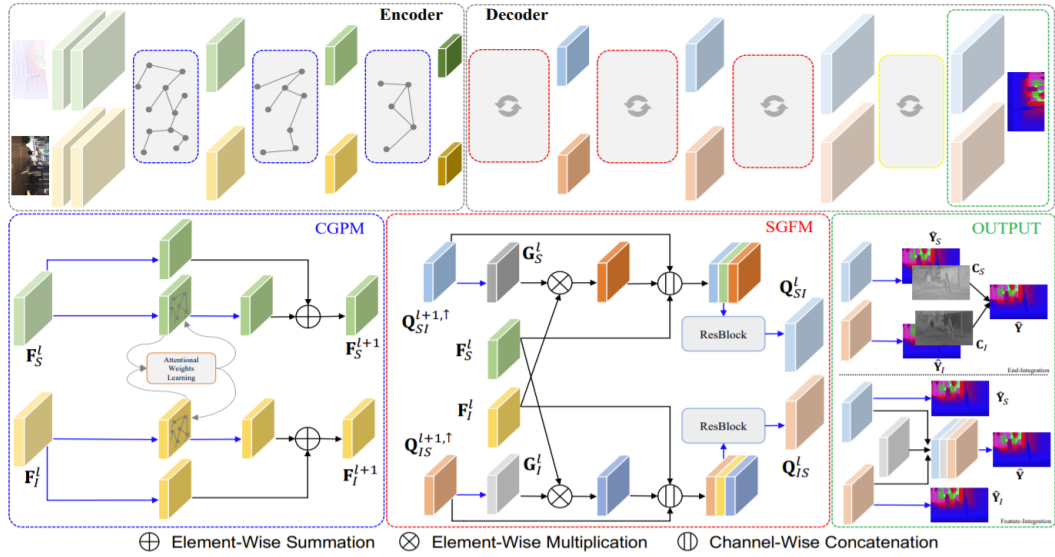


Figure 2.14: Depth completion architecture proposed by Shanshan Zhao *et al.* [69].

In 2019, Yun Chen *et al.* [70] proposed a method that produces a dense depth map from RGBD data. In this work, a neural network block was designed to learn to extract joint 2D and 3D features. This block uses two domain-specific networks that apply continuous convolutions on 3D points and 2D convolutions on image pixels, fusing their outputs in image space [70]. The overall network, presented in figure 2.15, is built by stacking the proposed block, learning hierarchical representations fused between 2D and 3D spaces [70]. In this network, the 2D-3D fuse block takes image and sparse depth as inputs and outputs a dense depth map. The 2D convolution network is divided in two branches, which are used to extract multi-scale appearance features. The first branch applies a simple 3x3 convolution, whereas the second branch applies two 3x3 convolutions, followed by an upsample block. The 3D continuous convolution network applies

continuous convolutions on the 3D points in order to learn geometric features [70]. The output feature maps provided by both 2D and 3D convolution networks are fused by simple element-wise summation as they have the same shape, followed by a 3x3 convolution.

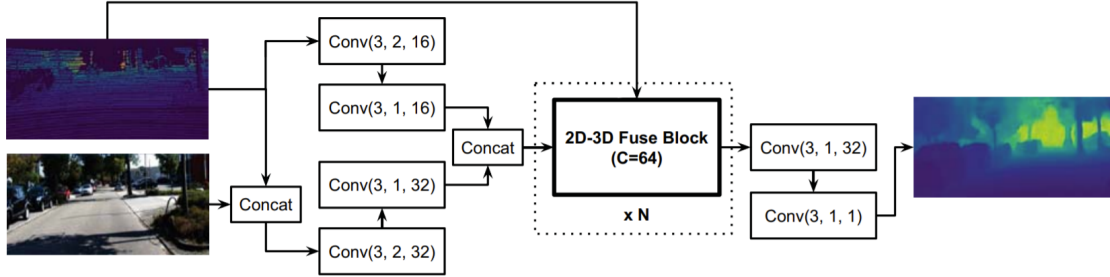


Figure 2.15: Depth completion network based on 2D-3D fuse blocks proposed by Yun Chen *et al.* [70].

In 2021, Pedro Leite and Andry Pinto [71] presented a fusion architecture that produces a dense depth map from a pair of RGB images. In this work, depth information was obtained by taking advantage of the Motion-Parallax effect, where closer objects to the camera present a higher variation of position when compared to objects further away. This depth information is then fused with the texture and color information provided by a RGB camera. The encoder-decoder model was adopted, where ResNet18 was used as the feature encoder and BLConv5 as the decoder. Figure 2.16 represents the developed encoder-decoder architecture.

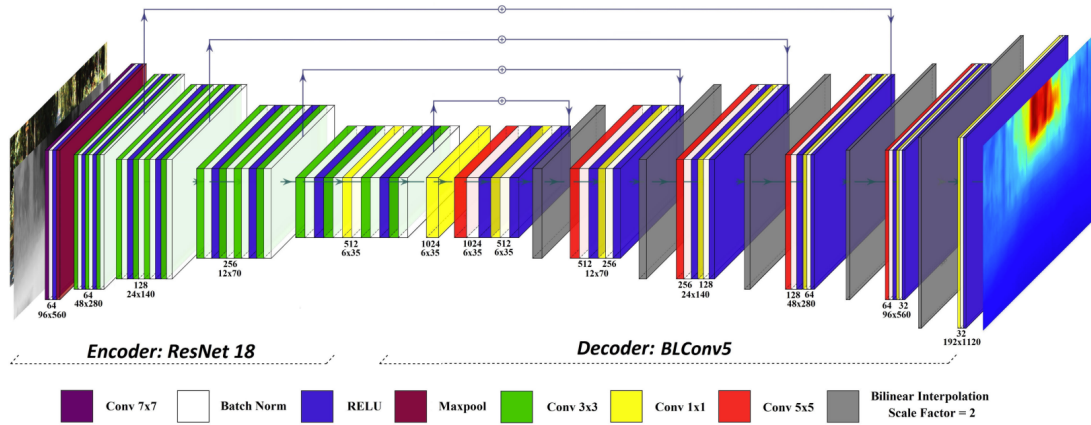


Figure 2.16: Depth prediction network proposed by Pedro Leite and Andry Pinto [71].

2.4 Critical Review

During this section, some of the most important ideas and conclusions taken from the conducted research on the state-of-the-art will be presented. In order to perform various tasks, ASVs need to be able to navigate and locate themselves, as such, it is important that they have a good perception of their surroundings. There are different sensors that can be used to achieve this, including

LiDARs, different types of cameras and radars. However, relying on just one of these sensors to gather information from the environment is not the best solution, as each sensor has limitations and there are situations where one performs better than the other. LiDARs provide important but sparse 3D information of the environment, with low resolution. On the other hand, RGB cameras provide rich color and texture information while covering a broader area and having more range. From the conducted research on these sensors and perception systems, it is possible to conclude that they complement each other very well and are often used in autonomous vehicles. As such, the proposed solution will take advantage of each of these sensors' strengths by fusing the information provided by them.

There are different methods and architectures that are designed to fuse information provided by different sensors, these can be categorised in two different approaches: conventional approach and machine learning approach. Conventional approaches are typically based on probabilistic methods that deal with the imperfection of data, while machine learning approaches take advantage of machine learning algorithms that learn to recognize patterns and important features in the input data. Generally speaking, machine learning approaches achieve better results, with the disadvantage of requiring high computational power and expensive data to train the networks. Convolutional neural networks are the most used networks when solving this problem, as they excel at extracting spatial features from data that is organized in a grid pattern, such as RGB images provided by cameras. The most interesting researched architectures involve light-weight networks that focus on reducing the number of parameters and computational power while still achieving good results. These architectures often use an encoder-decoder approach and different methods that use CNNs, skip connections and residual blocks.

Chapter 3

A Sensor Fusion Architecture

In this chapter, the machine learning sensor fusion architecture will be presented, along with each of the variations that were projected and tested. It is divided in three main sections: section 3.1 where the data acquisition and processing methods are presented; section 3.2.2 where the different developed encoder-decoder models are discussed; and section 3.3 where different loss functions and optimizers are introduced.

This dissertation focuses on combining information provided by sensors of different modalities, a LiDAR and a stereo pair of cameras, to overcome the issues that arise from working in the maritime environment. This information is fused by using machine learning algorithms that avoid relying on probabilistic models and the need to hand-craft features, typical of conventional approaches [71]. The output of the system is a dense depth map, giving the ASV relevant information about the distance of objects and surfaces, which can be used to perform various tasks such as obstacle avoidance ([72], [73]) and docking ([74], [75]). To solve this problem, a dense depth map will be obtained by fusing a sparse depth map obtained using feature matching, and color information provided by stereo RGB cameras.

The developed machine learning algorithms are strongly based on convolutional neural networks, as they perform well when the input data is organized in a grid pattern, such as images. These networks are designed to learn and extract spatial features and patterns, which is useful in this case because of the nature of the input data. An encoder-decoder architecture is a common approach to this problem, as concluded from the conducted state of the art research, as such, it was adopted as the architecture to be used in this dissertation. The encoder uses a residual learning approach, that can be the ResNet model or other approaches such as the MobileNetV2 model or the Densenet model. These architectures will be tested and their results will be compared to determine which produces better results. The decoder block is then responsible for up-sampling the feature map using unpooling kernels to recover details of the image and finally output a depth map. In this dissertation, different decoders will be explored, such as UpProj ([76] and [59]) and UpConv. Different training and optimization methods will also be explored, such as different loss functions, e.g. L1 loss and BerHu loss, and different optimizers, e.g. SGD and Adam.

The results and the performance of the implemented solution and its alternatives can be tested

and evaluated, taking into consideration different criteria, including root mean square error (RMSE) 3.1, scale invariant logarithmic error (SILog) 3.2 and the percentage of predicted pixels with a relative error that is within a threshold (δ_n) 3.3.

$$RMSE = \sqrt{\frac{1}{n} \times \sum_i (y_i - \hat{y}_i)^2} \quad (3.1)$$

$$SILog = \frac{1}{n} \times \sum_i d_i^2 - \frac{1}{n^2} \times \left(\sum_i d_i \right)^2 \quad (3.2)$$

where $d_i = \log y_i - \log \hat{y}_i$.

$$\delta_n = \frac{\text{card}(\{\hat{y} : \max\{\frac{\hat{y}_i}{y_i}, \frac{y_i}{\hat{y}_i}\} < 1.25\})}{\text{card}(\{y_i\})} < 1.25^n \quad (3.3)$$

where y_i is the ground truth, $\hat{y}_i$ is the prediction and card is the cardinality of a set.

The RMSE gives information on the metric error, in this work, presented in meters. The SILog error metric defined by Eigen *et al.* [77], measures the relation between points regardless of the absolute global scale and δ_n gives information on the visual quality of the prediction.

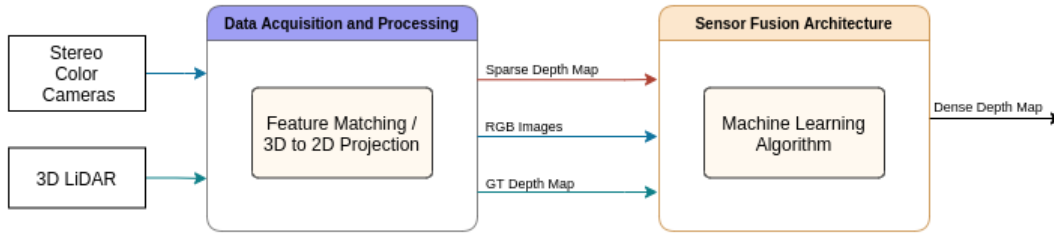


Figure 3.1: Diagram representing an overview of the sensor fusion process.

Figure 3.1 represents the overall sensor fusion process, starting from the acquisition and processing of the data provided by the sensors and ending with the neural network architecture that uses the data to generate a dense depth map.

3.1 Data Acquisition and Processing

When implementing and training the proposed machine learning architecture, it is essential that the input data is carefully processed and organized, resulting in a smooth data pipeline and facilitating the training process. During this section, the input data that is going to be used will be presented, along with the methods implemented to generate the sparse depth map from the RGB images obtained by the stereo camera and the depth map obtained by a 3D to 2D projection of the points obtained by the LiDAR.

Figure 3.2 represents an overview of the whole process using a simple diagram. The first step is to acquire information from the environment using the stereo cameras and the 3D LiDAR.

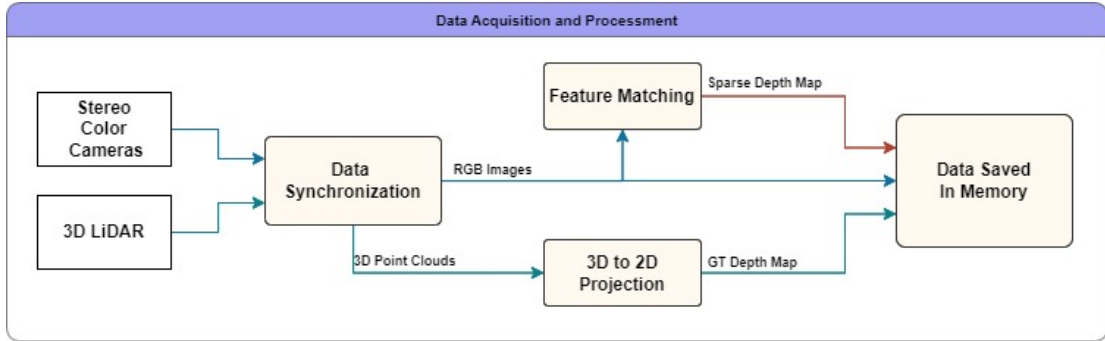


Figure 3.2: Diagram representative of the process of acquiring and processing the data used to train the machine learning model.

After acquiring the data from the color cameras and the 3D LiDAR, the images and the point clouds are then synchronized. This has to be done due to the different acquisition frequencies of each sensor. In order to ensure that both RGB images and the point cloud were acquired around the same time, it is necessary to check the timestamp of each message and discard the information in case there is a big time difference between them. After synchronizing the information provided by the sensors, each point cloud and stereo image pair need to be processed before being ready to be used as the inputs for the neural network. The RGB images are used as both a direct input to the neural network and a way to obtain a sparse depth map using feature matching. This sparse depth map is calculated by taking advantage of the methods that will be expanded upon in section 3.1.1, using relevant information such as the intrinsic camera parameters and its baseline. The 3D point cloud is used to calculate a precise depth map that is obtained by projecting each 3D point from the point cloud onto a 2D plane, so that this information is in the same representation of the rest of the data. This projection is done by using the methods that will be further discussed in 3.1.2, that utilize the camera intrinsic parameters as well as the relation between the LiDAR and the camera's frame.

All the necessary data is then stored in a structured way so that it is easier to load the inputs into the machine learning architecture. In the context of this dissertation, the data was stored using the same structure that the KITTI dataset [7] uses, that is, both calculated depth maps were stored following the directory paths and names already used by the original dataset.

3.1.1 Feature-Based Stereo

The previously mentioned sparse depth map is calculated using feature matching algorithms that consist on finding two sets of features from similar RGB images, figure 3.3, provided by two different cameras that are separated by a known distance (baseline). Each calculated feature present in the left image is then matched with a feature present in the right image, using a matching algorithm based on search distance algorithms. In this subsection, two different methods to calculate

the descriptors and features of two images will be briefly presented, the Scale Invariant Feature Transform [78] (SIFT), and Oriented Fast and Rotated Brief [79] (ORB).

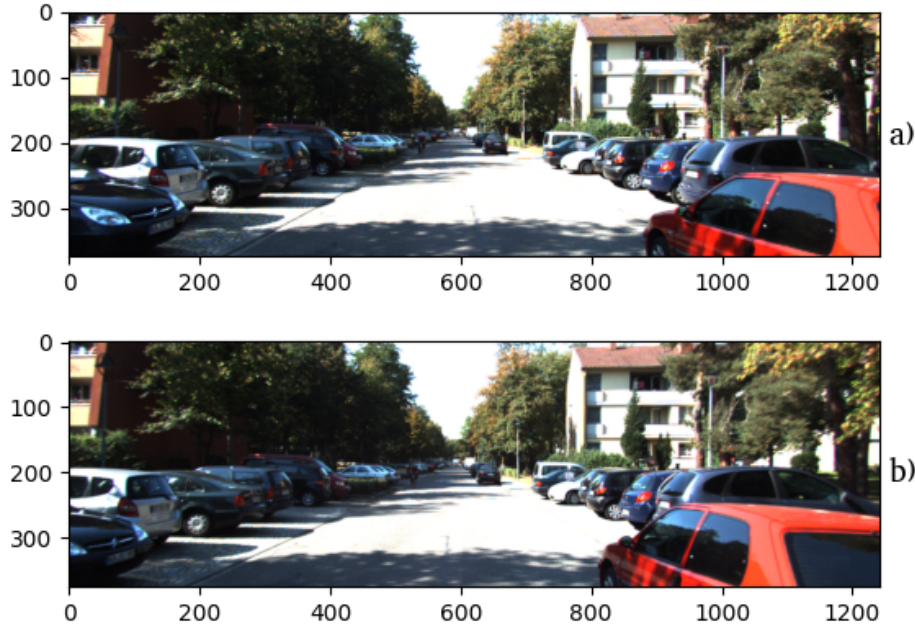


Figure 3.3: Example of an image pair obtained by the left (a) and right (b) cameras, from the KITTI dataset [7].

The SIFT [78] algorithm detects and describes local features present in an image, also known as keypoints, with the advantage that these keypoints are rotation and scale invariant. This allows a feature matching between two images even when one of them is rotated or in a different scale from the other. Figure 3.4 presents the keypoints that were calculated in the image obtained by the left camera, figure 3.3, using the SIFT algorithm.

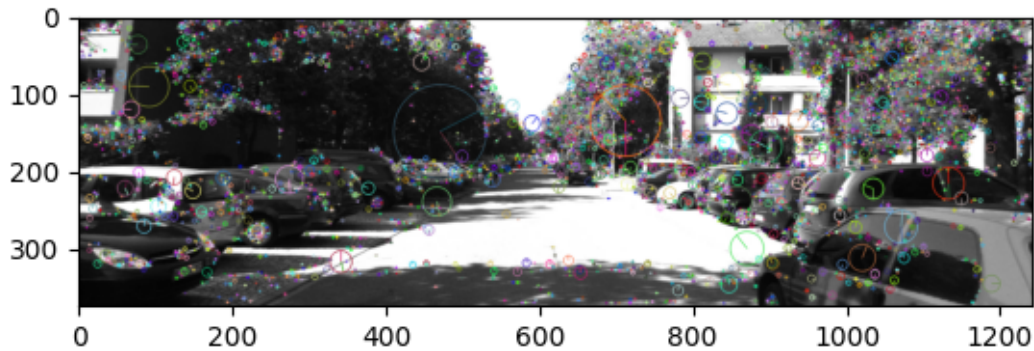


Figure 3.4: Keypoints obtained using SIFT [78] feature detection algorithm.

These calculated keypoints, along with their descriptors, are used by a matching algorithm that finds the correspondences between the two images. In this case, the Fast Library for Approximate Nearest Neighbors (FLANN) based matcher was used with the Kd-tree search algorithm.

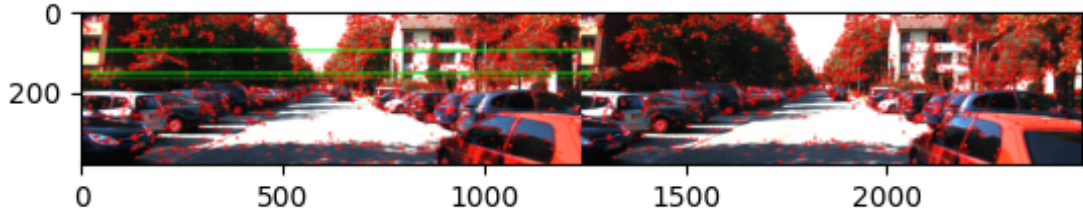


Figure 3.5: Example of matches obtained using SIFT [78] descriptors and FLANN based matcher.

Figure 3.5 presents all the features (in red) that were calculated on both images, as well as some of the matches that were obtained (represented by green lines). For each of the determined keypoints in the left image, there are multiple keypoints that are selected as a match in the right image, however, only the best matches are selected, based on the distance between the calculated keypoints of each image. These matches were then filtered using the ratio test, as proposed by D. Lowe [78], where a keypoint is discarded if the distance between the two best matches is not different enough, based on a selected constant, as represented in figure 3.4. In this case, the constant was set to 0.7, meaning that the distance between the best matched keypoints has to be at least 0.7 times the distance of the second best keypoint. This allows the elimination of possible matches that are not good enough to be used, ultimately resulting in more reliable input data in general.

$$dBest < dSecondBest \times aConstant \quad (3.4)$$

where $dBest$ is the distance between a certain keypoint and its best match, $dSecondBest$ is the distance between a certain keypoint and its second best match and $aConstant$ is the constant that determines the minimum difference allowed between $dBest$ and $dSecondBest$.

After matching and filtering the keypoints obtained, the sparse depth map is obtained by calculating the distance between each matched keypoint, using their coordinates in both the left (x_l, y_l) and the right (x_r, y_r) images, and by using the focal length and baseline of the used cameras, as shown in the equation 3.5.

$$d = \frac{B \times fx}{fmdist} \quad (3.5)$$

where d is the depth, B is the distance between the cameras (baseline), fx is the focal length of the camera, and $fmdist$ is the distance between the matched keypoints of each image.

The ORB algorithm, just like SIFT, is an algorithm that describes and detects features present in a given image, it is resistant to noise and is rotation invariant [80]. Compared to SIFT, it is faster and achieves results that are as good in many different situations. Figure 3.6 presents the keypoints that were calculated in the image obtained by the left camera, figure 3.3, using the ORB algorithm.

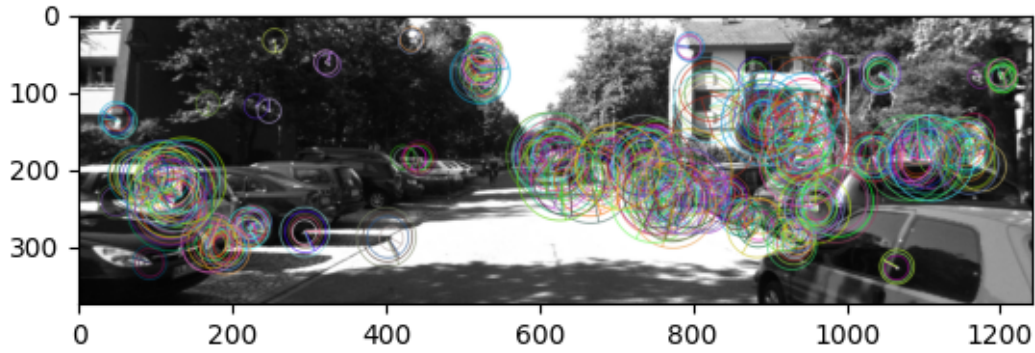


Figure 3.6: Keypoints obtained using ORB [80] feature detection algorithm.

Similarly to the previously discussed approach, these calculated keypoints, along with their descriptors, are used by a matching algorithm in order to find matching pairs of features on both images. In this case, a brute force matcher algorithm was adopted, taking the descriptor of each of the features present on the left image and comparing them to every feature present on the second image, using a distance calculation.

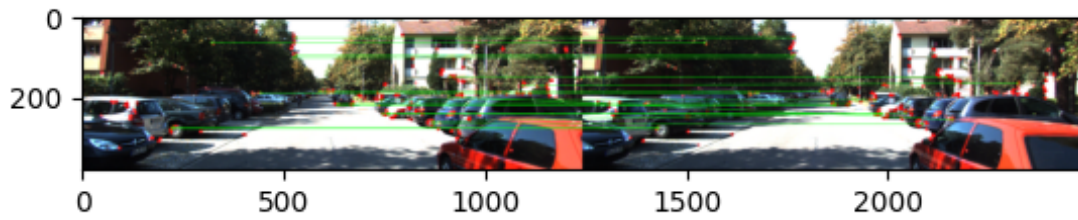


Figure 3.7: Example of matches obtained using ORB descriptors.

Figure 3.7 presents all the features (in red) that were calculated on both images, as well as the matches that were calculated (represented by green lines). This result was obtained by using the previously mentioned brute force algorithm, with a cross-check method that only accepts a pair of matched features if the feature from the right image is the best match possible and vice-versa. This filters the calculated matches, allowing a better and more consistent result, making it a great alternative for the previously mentioned ratio test that was proposed by D. Lowe [78]. The sparse depth map is then obtained using the same method that was previously discussed, which utilizes the equation 3.5.

Combination of SIFT and ORB outputs

In order to achieve a sparse depth map from stereo matching with a considerable amount of points, the results obtained from the two methods previously discussed were combined into a single depth map, providing more information for the input of the machine learning architecture.

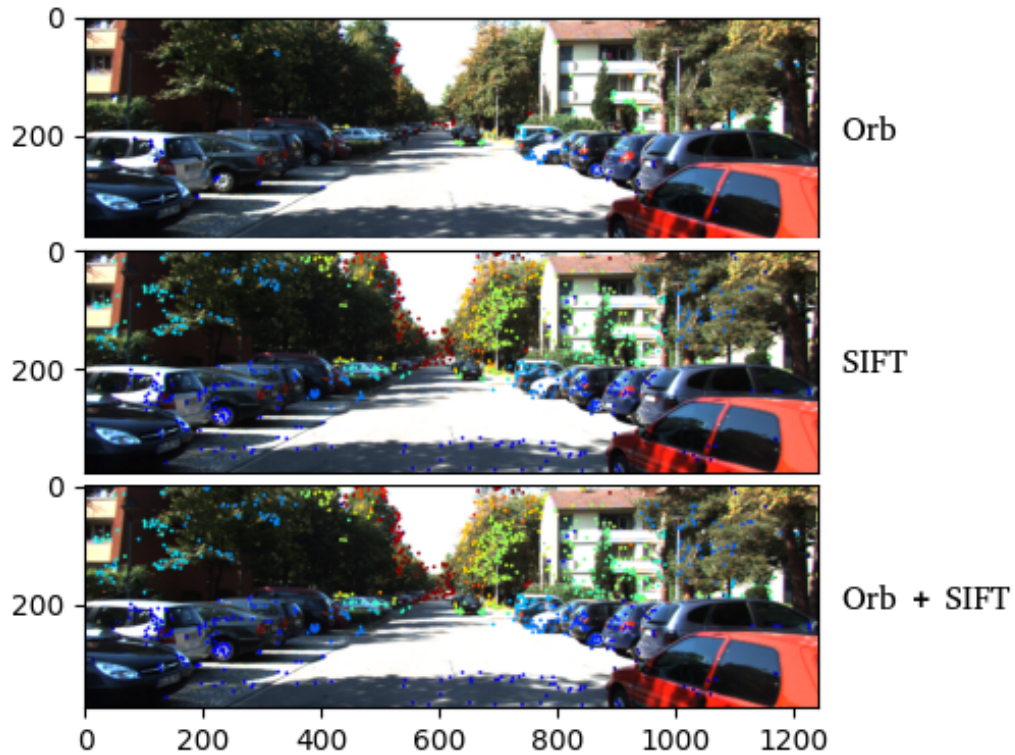


Figure 3.8: Depth points obtained using ORB (first image), SIFT (second image) and the combination of ORB and SIFT (third image), where red points represent higher depth values and dark blue points represent low depth values.

The result of this combination is demonstrated in figure 3.8, where the depth points are shown on top of the left camera image, represented in red. To ensure that the results obtained using these methods are reliable and can be used as the sparse depth input of the neural network, their values were compared with the ground truth depth map provided by the LiDAR. For depth points of objects that are closer to the camera, the error between the calculated depths and the depths provided by the LiDAR is small, meaning that these points are very accurate and contribute to a better final result. From comparing the calculated depths with the ground truth it was also possible to conclude that the farther away the objects are from the camera, the higher the error is. As this feature-based stereo method is heavily dependant on features present in the environment, the sparse depth input obtained in the maritime environment, where in a lot of situations there is a lack of texture information and reference objects, poses a problem in the training process, resulting in

one of the reasons for the results obtained in 4.1.

Region Growing

With the objective of improving and obtaining a sparse depth map with more information than the ones presented before, a simple region growing was applied where the pixels surrounding each depth point were considered to have the same depth value as the already calculated depth. This method was applied to every calculated depth map, obtaining the results presented in figure 3.9.

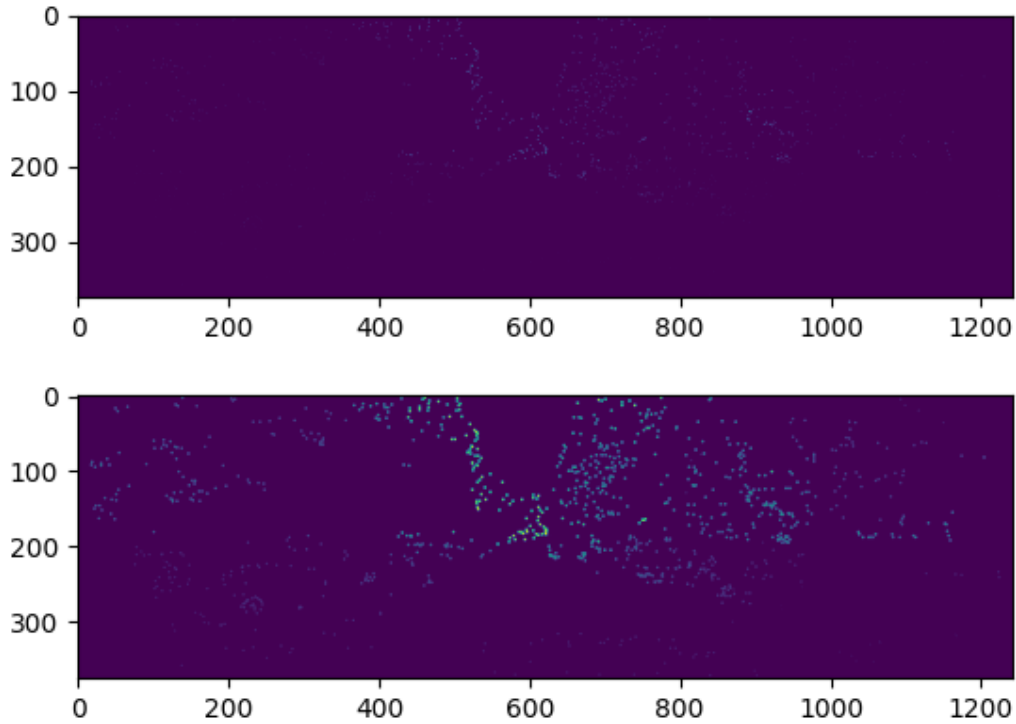


Figure 3.9: Example of region growing applied to the depth map previously obtained, the first image presents the depth points before growing and the second image presents the results of the region growing.

3.1.2 LiDAR Point Cloud Projection

As previously mentioned, for the training phase, the input of the neural network is composed by the RGB image obtained using a color camera and a sparse depth map. These inputs are combined into one single representation with 4 channels, consisting of the three color channels provided by the camera and one depth channel that can either be the calculated sparse depth map using feature matching, or projected points provided by a LiDAR, when a second LiDAR is available. In this machine learning architecture, the ground truth is considered to be the precise points obtained by the 3D LiDAR. To do this, it is necessary to project the 3D point cloud into a 2D plane, so that each point is correctly represented and located in pixel coordinates. The first step is to transform the 3D point cloud from the LiDAR's referential frame to the left camera's frame. This is done using the homogeneous matrix, equation 3.6, that represents the translation and rotation from the LiDAR's

frame to the left camera's frame. Each 3D point is then multiplied by this matrix, obtaining its new coordinates in the left camera's frame, equation 3.7.

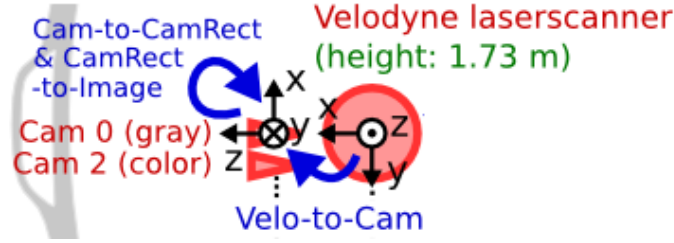


Figure 3.10: Representation of the 3D LiDAR and left color camera (cropped image obtained from the KITTI dataset [7]).

$$H_l^c = \begin{bmatrix} R_l^c & T_l^c \\ 0 & 1 \end{bmatrix} \quad (3.6)$$

$$P^c = \begin{bmatrix} R_l^c & T_l^c \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} P^l \\ 1 \end{bmatrix} \quad (3.7)$$

where H_l^c , R_l^c and T_l^c are the homogeneous, rotation and transformation matrices from the LiDAR frame to the camera frame, respectively, P^c is a point in the camera frame and P^l is a point in the LiDAR frame. The next step is to project each point from their 3D representation $P^{3D} = (X, Y, Z)$ to the 2D image plane, where each point will be represented by a pixel $P^{2D} = (u, v)$. To do this, the camera was approximated by the basic pinhole model, represented in figure 3.11.

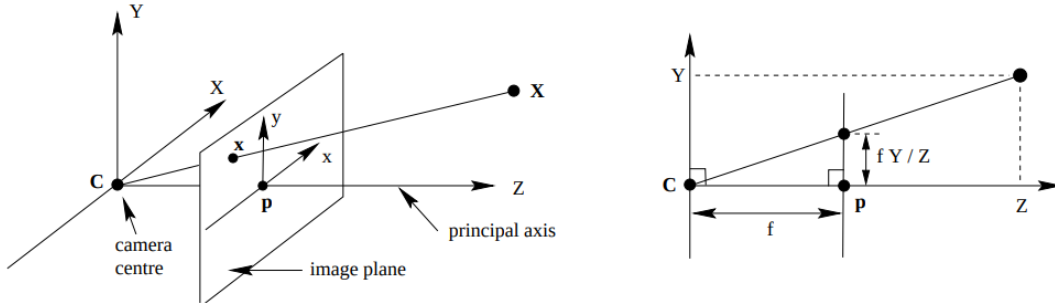


Figure 3.11: The pinhole camera model [81].

As shown in figure 3.11, C is considered to be the center of projection for the points projected from the 3D space into a plane. This point can be representative of the pinhole, and the origin of an Euclidean coordinate system. The plane represented by $Z = f$ is called the image plane, which is the plane that contains the object's projected image. Using this model, the point X in the 3D space can be mapped to a point x on the image plane by drawing a line that connects the point X and the center of projection C, as represented in figure 3.11. The point p, called the principal point, is the intersection of the Z axis (principal axis) and the image plane, and is located at $p = (x_0, y_0)$, as represented in figure 3.12.

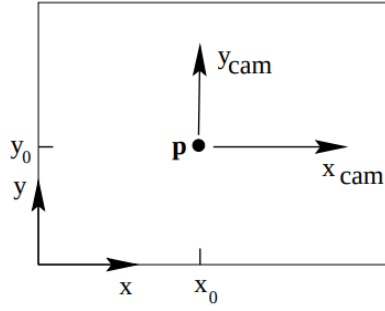


Figure 3.12: Principal point represented in the image (x,y) and camera (x_{cam},y_{cam}) coordinate systems [81].

Using this model and the intrinsic camera parameters such as the focal length and the coordinates of the principal point, it is possible to project a 3D point that is represented in the camera's frame to a 2D point in the image plane by using the equations 3.8 and 3.9 [81].

$$P_u^{2D} = f_x \cdot \frac{P_x^{3D}}{P_z^{3D}} + x_0 \quad (3.8)$$

$$P_v^{2D} = f_y \cdot \frac{P_y^{3D}}{P_z^{3D}} + y_0 \quad (3.9)$$

where P^{2D} is the projected point in the image plane, P^{3D} is the 3D point obtained by the LiDAR, f_x and f_y are the focal lengths, and lastly C_x and C_y are the coordinates of the principal point. Taking this into consideration, every point acquired by the 3D LiDAR was projected into the image, obtaining a pair of pixel coordinates for each point. These points were then filtered in order to exclude points that had their pixel coordinates out of the images captured by the camera. In the case of the KITTI dataset, the images are 1242x375 pixels, meaning that any point that falls out of this range will be discarded.

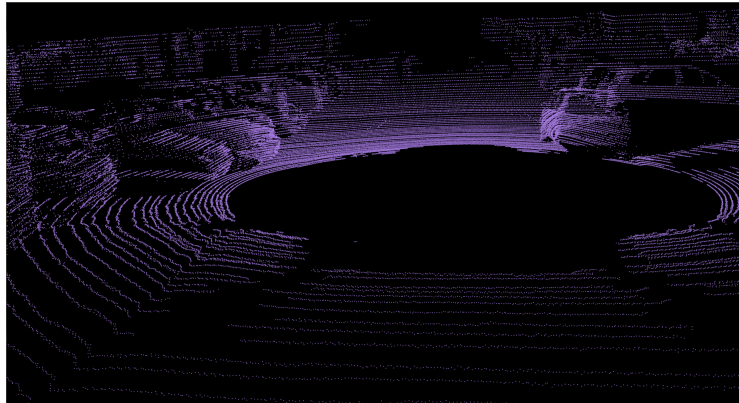


Figure 3.13: Example of a point cloud before projecting to 2D plane

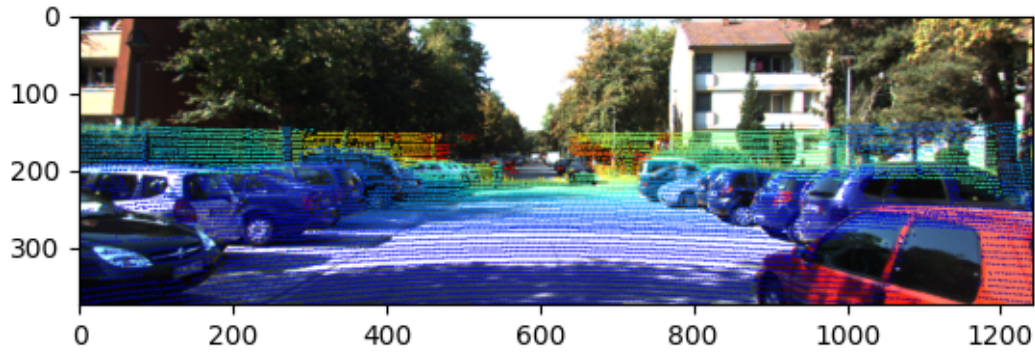


Figure 3.14: Example of the projection of 3D points acquired by the LiDAR onto the left camera, where red points represent higher depth values and dark blue points represent low depth values.

Figure 3.13 represents a point cloud in the 3D space and figure 3.14 represents the projections of those 3D LiDAR points onto the left camera image that was used in the previous examples. This process is applied to every acquired point cloud and the results obtained are stored as images to be used in the neural network.

3.2 Developed Architecture

In this section, the sensor fusion architecture developed and trained will be presented and discussed. Each subsystem and neural network component integrating this architecture will be carefully explained. The diagram presented in figure 3.15 represents the system, starting with the preprocessing of the input data, followed by the sensor fusion model that calculates a dense depth map from these inputs. In this diagram, the overall training process is demonstrated, and each of its components and variations are further explored below.

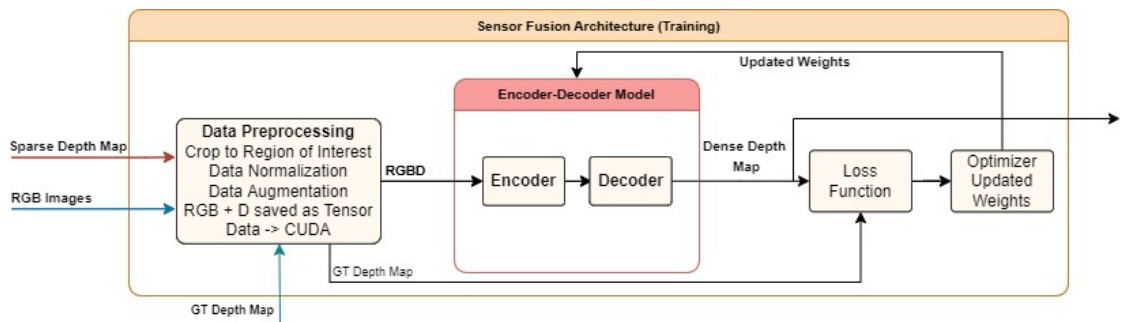


Figure 3.15: Diagram representing the training process of the sensor fusion algorithm.

3.2.1 Neural Network Data Preprocessing

In this architecture, an early fusion method was adopted where the input sparse depth map and the RGB image are combined into a single input with four different channels, represented by RGBD in

figure 3.15. These channels represent the Red, Green and Blue channels from the colored images as well as the Depth channel from the sparse depth maps. The first step is to crop the images according to a region of interest, due to the fact that there is a big area in the color images where the information provided by the LiDAR is minimal or non-existent. This can be easily observed in figure 3.14, where the top region of the image has no depth points provided by the LiDAR, making this region not valuable in this context.

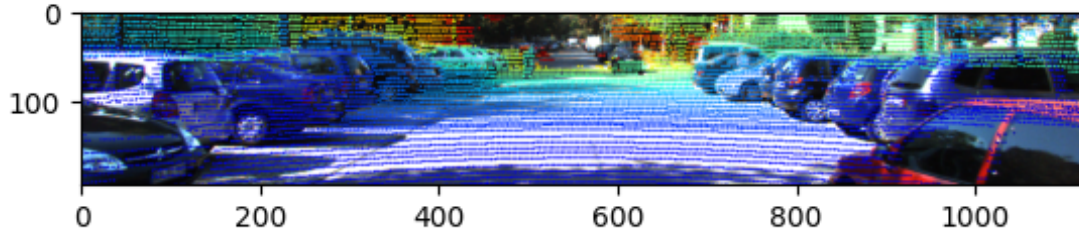


Figure 3.16: Example of a cropped image taking into consideration the region of interest, where red points represent higher depth values and dark blue points represent low depth values..

Figure 3.16 represents the region of interest obtained by applying the aforementioned crop to the image 3.14, where most of the image contains valuable information that is used by the neural network. After cropping the images to the appropriate region of interest, the developed data loader performs a normalization of the input data and the ground truth so that each value is between zero and one, according to the maximum range of the LiDAR. This is done to ensure that the four input channels that will be combined into a single variable are on the same scale.

3.2.2 Developed Encoder-Decoder Models

The developed architectures are based on the Encoder-Decoder model, as it is proven to achieve good results while remaining simple and faster than more complex architectures, as concluded from the conducted state of the art research. In this work, different architectures with different hyperparameters were built and tested. In this subsection, every variation of these encoder-decoder models will be presented. Three different feature encoders were tested: ResNet, DenseNet and MobileNetV2, while using two different decoders: Up-Convolution and Up-Projection based. These architectures were also tested using skip connections from the encoder to the decoder and trained with two different optimizers: Adam and SGD. The results obtained using each of these architectures will be presented and discussed in chapter 4, giving insight on which parameters and models work best using the available and calculated input data.

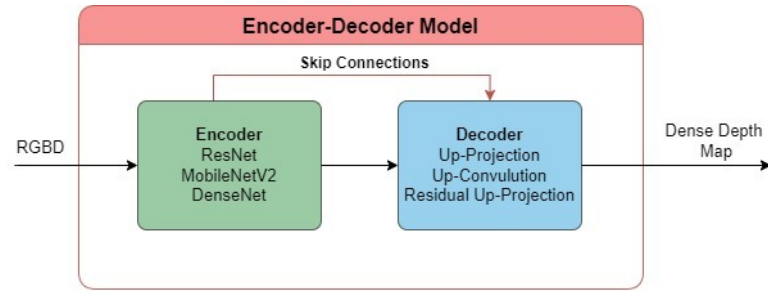


Figure 3.17: General Encoder-Decoder architecture.

The diagram represented by figure 3.17 represents the general Encoder-Decoder model. As explained before, the input of the network is a combination of the RGB image and the sparse depth map, forming a 4 channel tensor from which the encoder captures features, resulting in an early fusion. The encoder is responsible for understanding and extracting important features from the provided input, transforming that input data into increasingly smaller encoded features that contain valuable information. The decoder is then used to interpret the encoded information and recreate the desired output.

Encoder: Adapted ResNet Network

Using ResNet as a feature encoder is a common solution, as it provides good results while remaining relatively simple. In this work, the well-known ResNet model that is used for image classification was used and adapted to the input data and the final fully connected layer was removed, as shown in figure 3.18. In the original ResNet model, this layer is used to interpret the extracted features and classify the input image from 1000 different objects [54]. However, in this work, the objective is to use the extracted features as the input for the decoder network and as such, the fully connected layer is replaced by a convolutional layer that connects the encoder and the decoder. The first convolution was changed to accept an input with 4 different channels.

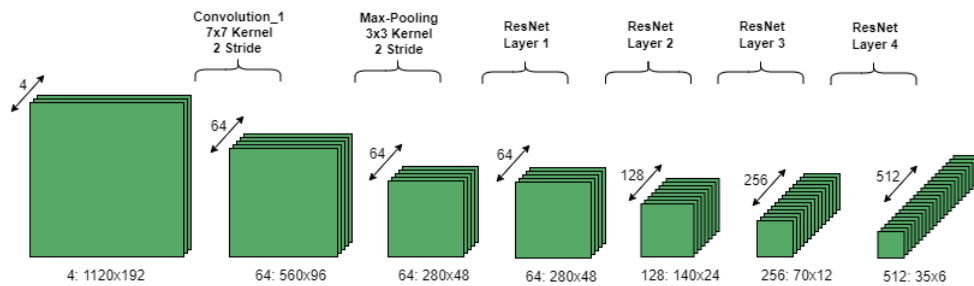


Figure 3.18: Adapted ResNet-18 neural network.

As mentioned before, the initial image size is always halved 5 times by the encoder network, in this case by *Convolution 1*, *Max-Pooling*, *ResNet Layer 2*, *ResNet Layer 3* and *ResNet Layer 4*, as shown in figure 3.18. The represented model is the 18 layer version of the ResNet model, resulting in 512 different features with a width of 35 pixels and a height of 8 pixels, in this case, where the input is 1120 pixels by 192 pixels. Each *ResNet Layer* is composed of a series of four

3x3 convolutions, following the structure of the residual block represented in figure 2.6.

Encoder: Adapted DenseNet Network

Similarly to the case of the ResNet, the DenseNet architecture, figure 3.19, developed for image classification was adapted and used as a feature encoder. To do this, the first convolutional layer was modified to reflect the number of inputs used (4 input channels to 64 output channels), and the classification layer was removed and replaced by the decoder model.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Figure 3.19: DenseNet architecture used for classification [55].

The DenseNet-121 version was used due to hardware limitations and the increased speed in the training process. This network is built based on the structure represented by figure 2.7. In this architecture, just like the ResNet, the size of the input image is reduced 5 times by a factor of 2, meaning that the same input image size and decoders can be used.

Encoder: Adapted MobileNetV2 Network

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Figure 3.20: MobileNetV2 architecture used for classification [56].

The final encoder that was used and tested is an adapted version of the MobileNetV2 architecture [56], represented in figure 3.20. Once again, the input layer was changed to reflect the number of channels of the input data and the last two layers were removed and replaced by the decoder.

This architecture takes advantage of the bottleneck block represented in figure 2.8, by using them as a building block for the final model.

Decoder Block: Up-Convolution

To increase the feature size and interpret the information derived from the encoder, the decoder part of the model is composed of multiple blocks that perform up-sampling. Figure 3.21 represents a simple Up-Convolution block that uses an unpooling layer with a stride of 2, followed by a 5x5 convolution.

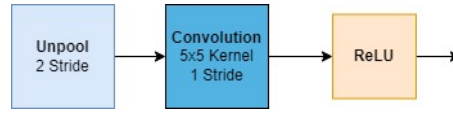


Figure 3.21: Simple Up-Convolution block.

Decoder Block: Up-Projection

The Up-Projection block uses up-pooling kernels that interpret and recover important details in the image. The Up-Projection models used in this architecture take advantage of this, and in addition, use residual connections as shown in figures 3.22 and 3.23.

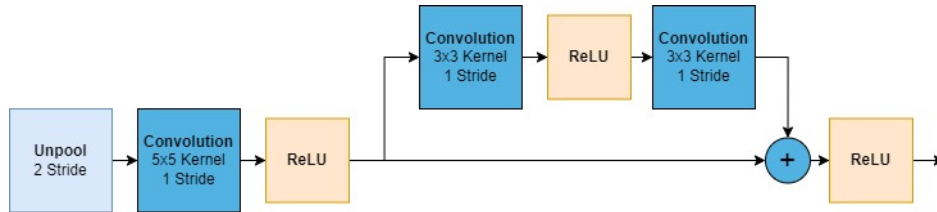


Figure 3.22: Up-Projection block as proposed by C. Fu *et al.* [59].

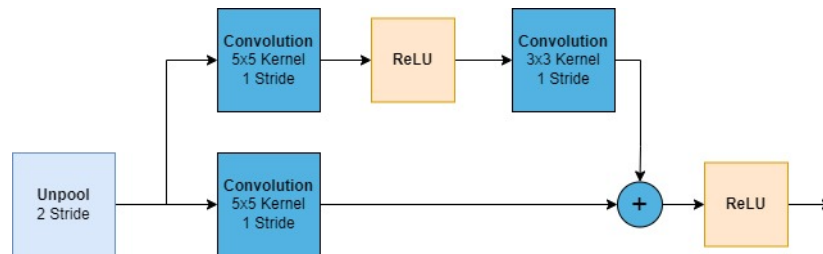


Figure 3.23: Up-Projection block as proposed by Iro Laina *et al.* [76].

Encoder-Decoder: ResNet-18 and Up-Projection

Figure 3.24 represents the general developed Encoder-Decoder architecture, using the ResNet-18 model presented in 3.18 and the Up-Projection blocks presented in 3.23. In this architecture, a

convolutional layer is used to connect the encoder (represented in green) and the decoder (represented in blue). The decoder represented here is composed of a series of four Up-Projection blocks that double the feature size, followed by a convolution that congregates the information present in the 64 features obtained by the previous layer into a single feature. The final dense depth-map is then obtained by applying a bilinear upsampling block. As previously mentioned, the encoder here represented in green, was also tested using the DenseNet and the MobileNetV2 models.

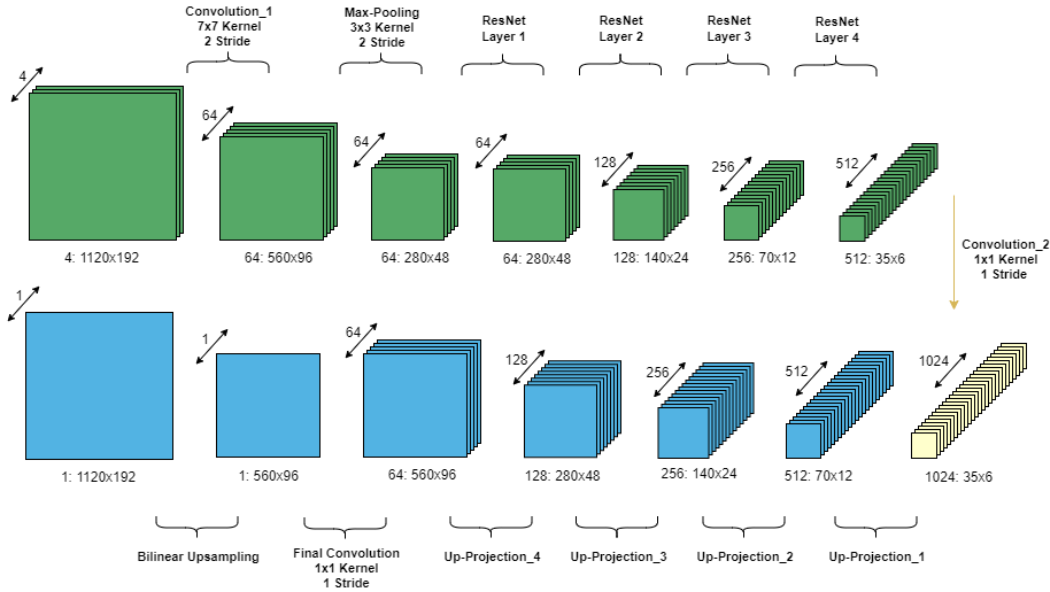


Figure 3.24: Encoder: adapted ResNet-18 (in green), Decoder: multiple Up-Projection blocks (in blue).

The structure of the decoder was also modified and its variations were implemented and tested. These variations include the usage of different Up-Projection blocks and the usage of five Up-Projection blocks instead of four, resulting in 32 features with the final image size, followed by a final convolution that transforms these features into the desired depth map. Different versions of the architectures were also implemented and tested, such as the ResNet-34 which is similar to the ResNet-18 but with more residual layers, resulting in more parameters to learn and slowing the training process.

Encoder-Decoder: Skip Connections

Compared to the architecture presented in 3.24, multiple skip connections were added, where features obtained by the residual encoding blocks are concatenated with the features obtained by the decoding blocks. This improves the overall result by propagating texture and contextual information from the encoder to the decoder, as well as object boundaries and other information that may be lost during the encoding-decoding process. Figure 3.25 represents these connections, represented in red, where the output features from the Resnet Layer 1, 2 and 3 are concatenated with the output features from the Up-Projection 1, 2 and 3 blocks.

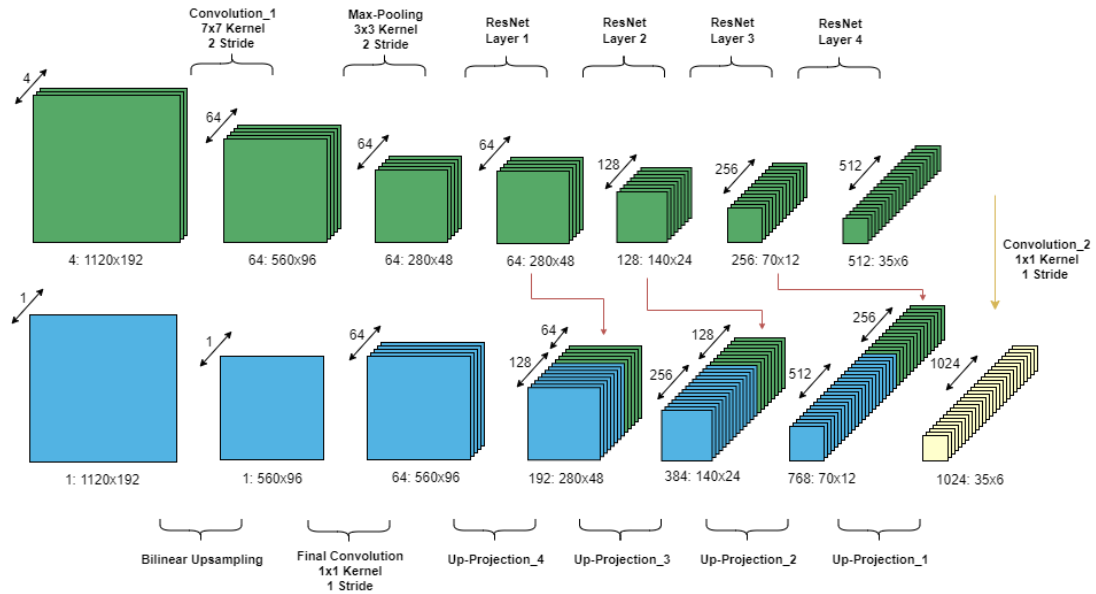


Figure 3.25: Encoder: adapted ResNet-18 (in green), Decoder: multiple Up-Projection blocks (in blue) with skip connections from the encoder blocks.

The usage of these skip connections resulted in a better depth prediction in general and were used in most of the implemented and tested architectures. Three skip connections were also used in the architecture where the MobileNetV2 is used as the feature encoder.

Encoder-Decoder: Cascade Architecture

A cascade of multiple Encoder-Decoder models was also implemented, where the dense depth map calculated by the first model is combined with the same RGB image used as its input, forming a 4 channel tensor that is used as the input of the second model, as represented in figure 3.26.

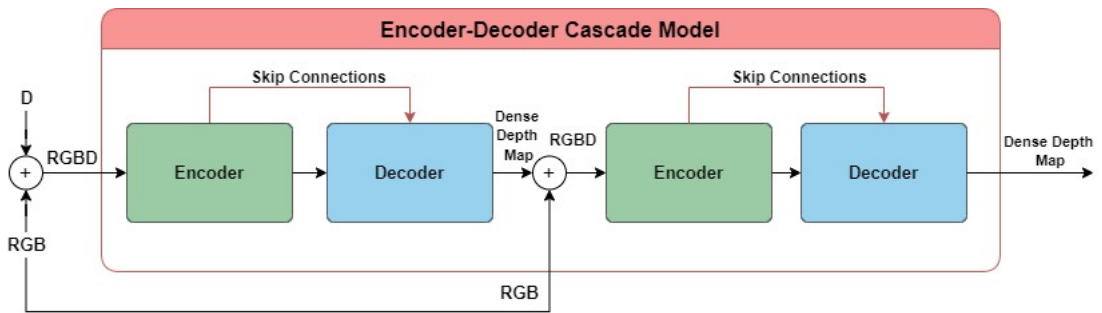


Figure 3.26: Encoder-Decoder cascade architecture.

This aims to improve the prediction provided by the first model and obtain a more accurate dense depth map. This architecture is heavier than the single encoder-decoder architecture, slowing down the training process and increasing the required hardware capabilities. Different approaches to this architecture were implemented and tested. Different encoder-decoder models

were used as the first block of the cascade. In some cases, a previously trained model with its parameters frozen was used, with the advantage of training one model at a time and with the objective of taking the output of the first model and refine it. The overall performance of this architecture will be presented and discussed in the result chapter.

3.3 Network Optimization and Implementation details

In this section, different loss functions and optimizers will be presented, along with important parameters used in the training process of the neural network.

3.3.1 Loss Function

The loss function plays an important role in the training process of the neural network model. It is used as a means of evaluating the performance of the model and the learned parameters by comparing the output of the neural network to the desired value, in this case, the depth map used as ground truth. The loss function measures how far the calculated output is from the desired value, and the objective of the training process is to minimize this function in order to obtain more accurate results. In the context of the problem at hands, different loss functions commonly used for regression problems will be presented and used in the training process.

L1 Loss Function

The L1 loss function, also called the least absolute deviation (LAD), consists on calculating the sum of the absolute differences between the predicted value and the target value, as represented in equation 3.10.

$$L1_{Loss} = \sum_{i=1}^n |y_{GT} - y_{pred}| \quad (3.10)$$

where y_{GT} is the target value and y_{pred} is the estimated value.

L2 Loss Function

The L2 loss function, also called the least squared error (LSE), consists on calculating the sum of the squared differences between the predicted value and the target value, as represented in equation 3.11

$$L2_{Loss} = \sum_{i=1}^n (y_{GT} - y_{pred})^2 \quad (3.11)$$

where y_{GT} is the target value and y_{pred} is the estimated value. When compared, the L1 loss function is more robust than the L2 loss function due to its resistance to outliers present in the data. Since the L2 loss squares the error, in case of an outlier where the error value assumes a high value, a significantly larger error will be produced, making the L2 loss function more sensitive to this situation. On the other hand, the L2 loss function is more stable, producing smaller changes

in the regression line when faced with small adjustments of a data point.

BerHu Loss Function

The BerHu loss function [82] is a reverse of the Huber loss that produces better results than using the L1 or the L2 loss functions in depth estimation problems, as concluded from [76] and the results of this dissertation. The BerHu function uses two different methods to calculate the loss for a specific pixel, depending on the absolute value of the difference between the the estimated depth value and the ground truth. When this value is lower than the value c , the L1 loss function is applied to that pixel and when it is higher than the value c , a continuous version of the L2 loss function is applied, as shown in 3.12.

$$B(z) = \begin{cases} |z|, & |z| \leq c, \\ \frac{z^2 + c^2}{2c}, & |z| > c \end{cases} \quad (3.12)$$

$$c = 0.2 \cdot \max(|z|) \quad (3.13)$$

where B is the BerHu loss function, z is the difference between the predicted value and the target value, c is a positive real number. Using the BerHu loss, it is obtained a balance between both norms, where L1 accentuates the impact of small residual values, while L2, in opposition to other loss functions that are robust, gives high value to pixels with high residuals. The value c is calculated by finding the maximum residual value in a given batch of images and multiplying it by 0.2, as shown in 3.13.

3.3.2 Optimizers

In the training process of a neural network, the optimizer is responsible for improving the performance of the model. It takes into consideration the loss function and has the objective of minimizing its value, which is done by updating the weights of the neural network. The rate at which the weights are updated depends on the value of the hyperparameter learning rate. This parameter determines the size of the step taken into the direction that minimizes the loss function, taking into consideration the calculated gradient. Choosing the value of the learning rate is important as it determines how fast the training process is. A small value will result in a long training process that may not achieve an optimal set of weights, while a high value may result in an unstable training process. A learning rate schedule was used where the value of the learning rate decreases every few epochs. This helps the learning process by starting with a higher learning rate in order to find a good set of weights faster, and then reduce its value to fine-tune the weights. Different learning rate values and schedules were tested and their results will be presented.

Another important hyperparameter used by the optimizer that needs to be adjusted is the weight decay. It is considered a regularization method that prevents the exploding gradient problem as well as the overfitting of the model to the training data. It consists on adding an extra

penalty term to the cost function. The L2 penalty, as presented in 3.14 was used, with a weight decay value of 0.1.

$$C = L + \frac{\lambda}{2n} \sum_{i=1}^n w_i^2 \quad (3.14)$$

where L is the loss function, λ is the weight decay parameter and w is the weight.

Two different optimizers were used and compared: the Stochastic Gradient Descent (SGD) and the Adaptive Moment Estimation (Adam) [83].

Stochastic Gradient Descent

The Gradient Descent is an optimization algorithm that uses the first order derivative of the loss function. It takes advantage of the gradient of that function, which gives information on how it changes when a small variation is applied to its inputs. The Back-Propagation algorithm is used in order to calculate the gradient for each individual weight present in the neural network, allowing the optimization algorithm to update their values according to the value of the gradient. Equation 3.15 represents how the SGD algorithm updates each weight, without taking into consideration the momentum.

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t} \quad (3.15)$$

where w is the weight, α is the learning rate and L is the loss function.

However, instead of only using the current gradient, the SGD with momentum was adopted where the exponential moving average of the past gradients, up to time t , are used, 3.16 and 3.17. This alternative results in a faster learning process by improving the speed of convergence, while dampening oscillations.

$$w_{t+1} = w_t - \alpha m_t \quad (3.16)$$

$$m_t = \beta m_{t-1} + (1 - \beta) \frac{\partial L}{\partial w_t} \quad (3.17)$$

where w is the weight, α is the learning rate, L is the loss function, m is the momentum and β is the momentum hyperparameter.

Adam

The Adaptive Moment Estimation [83] is a stochastic optimization algorithm that calculates individual adaptive learning rates, using first and second moments of the gradients. It has the advantage of working with sparse gradients, the parameter updates are not dependant on rescaling of

the gradient, the stepsize is bounded by the learning rate and carries out a step size annealing [83]. Equation 3.18 represents how the parameters are updated using this optimizer.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m} \quad (3.18)$$

Bias-corrected first and second moment estimate:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (3.19)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (3.20)$$

Biased first and second moment estimate:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t} \quad (3.21)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\partial L}{\partial w_t} \right]^2 \quad (3.22)$$

The values of each parameter of this algorithm were chosen based on the default values suggested by the author [83]: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$. Just like SGD, in this work, a weight decay using the L2 penalty was applied to this optimizer.

Weight Initialization

Weight initialization is a method that consists on setting the weights of a neural network to random values, defining the initial point for the training of the network. In this work, the initialization proposed by Kaiming *et al.* [49] was adopted, where the initial values are set according to a zero-mean Gaussian distribution with a standard deviation defined by equation 3.23.

$$std = \sqrt{\frac{2}{n_t}} \quad (3.23)$$

where std is the standard deviation, n is the number of connections of a response.

Chapter 4

Experiments and Results

In this chapter, the results obtained using the architectures and methods explained in chapter 3 will be presented and discussed. Each architectural variation will be compared to each other using the metrics previously established in 3, along with visual inspection of the predicted depth map. With the objective of developing and validating the machine learning architecture, the real data acquired in a land environment, provided by the KITTI dataset [7] was first used. This approach was adopted, as land datasets are easier to acquire and are more common in the scientific community, as opposed to the lack of categorized data using ASVs. Furthermore, this allows to understand if the developed network is correctly learning to interpret the input data, since depth information is extracted from the RGB images by exploring their textures, shadows and occlusions [71], regardless of the acquisition environment. The vehicle used to acquire the KITTI dataset [7] is equipped with different sensors that collect information of its surroundings, however, in the context of this dissertation, only the two color cameras and the 3D LiDAR will be used. The specifications

The specifications of the relevant sensors used in this work are [7]:

- **Camera** - PointGray Flea2 color cameras (FL2-14S3C-C), 1.4 Megapixels, 1/2" Sony ICX-267 CCD, global shutter.
- **LiDAR** - Velodyne HDL-64E rotating 3D laser scanner, 10 Hz, 64 beams, 0.09 degree angular resolution, 2 cm distance accuracy, collecting 1.3 million points/second, field of view: 360° horizontal, 26.8° vertical, range: 120 m.

On the other hand, the sensor configuration, figure 4.1, represents the relevant sensors that were used by the ASV developed at INESC TEC, the SENSE [8], giving a general idea of the location of each sensor and the relative position between them. The relevant sensors to take into consideration for this work is the LiDAR (Velodyne VLP-16) and the stereo camera (Mynt Eye D).

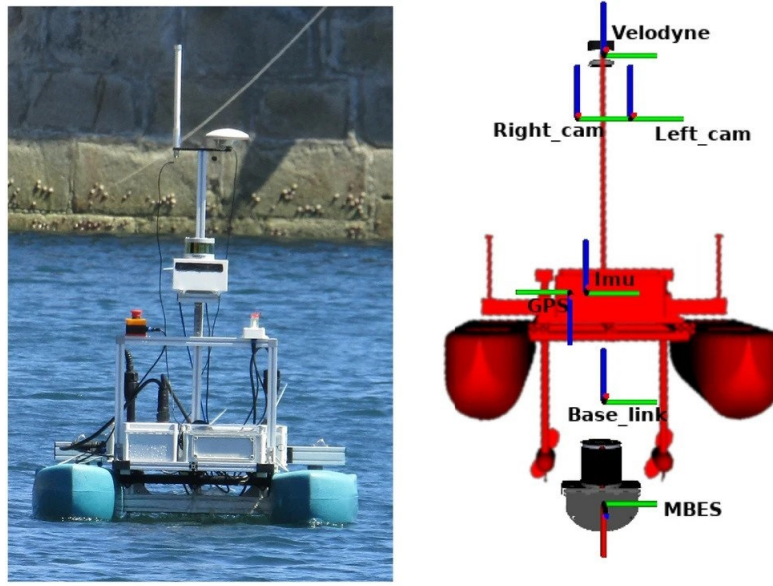


Figure 4.1: INESC TECs' SENSE vehicle and its sensor configuration [8].

The experiments presented in this chapter include direct comparison of similar architectures to determine which optimizer performs better. Architectures using the same encoder with different decoder models were compared, allowing to determine the best decoder option. Other experiments include comparisons between: pretrained models and random weight initialization; cascade and single encoder-decoder models; cascade using transfer learning and default cascade.

4.1 KITTI Dataset

In this section, the results obtained by training the neural networks using the KITTI dataset [7] will be presented and discussed. During this process, the data was separated into training data and validation data, using a similar split to the one proposed by Eigen [77], where 23158 images were used for training and 652 images were used for validation. The results presented using the aforementioned metrics only take into consideration the batch of images selected for validation. For most architectures a batch size of 4 was used, with the exception of the cascade models, where a batch size of 2 was used due to hardware limitations. The system running the neural networks has a GeForce RTX 2060 graphics card with 6GB of memory and an Intel i7-10750H CPU.

Table 4.1 presents a summary of the results obtained by each of the proposed architectures. The *Skip* column gives information on which architectures use skip connections from the encoder to the decoder, the *Cascade* column gives information on which architectures use the previously mentioned cascade architecture where multiple encoder-decoder models are used and the *Extra* column gives extra information relevant to each architecture. The column *Epoch* represents in which training epoch the best result was obtained for each architecture, and the overall best value for each of the used metrics is presented in bold.

As previously mentioned, the value of the learning rate is important as it determines how fast the neural network learns the value of each parameter in order to minimize the loss function. Different learning rates were tested, arriving at the conclusion that in this context, a starting learning rate of 0.01 or higher results in an unstable learning process, as the value of each parameter is updated using a big step, ultimately resulting in a network that does not converge. A learning rate with a value lower than 0.001 results in a slow learning process. Taking this into consideration, it was possible to conclude that a learning rate with a value of 0.001 is a good starting point and every architecture here presented was trained using this value. A learning rate schedule was used, where in most cases the value of the learning rate was reduced by a factor of 10 every 3 training epochs, with the objective of progressively fine tune the parameters by reducing their changing rate.

Both BerHu and L1 loss functions were tested, arriving at the conclusion that the BerHu loss is the better option in this context, as it was also previously deducted from the performed literature review. Using this information, every architecture here presented uses the BerHu loss function with the L2 penalty defined in subsection 3.3.1.

Table 4.1: Summary of the results obtained using the KITTI dataset [7]. Higher values of $\delta 1$, $\delta 2$ and $\delta 3$ represent better results, while lower values of $RMSE$ and $SILog$ are better.

No	Encoder	Decoder	Skip	Optim.	Cascade	Extra	Epoch	RMSE (m)	$\delta 1$ (%)	$\delta 2$ (%)	$\delta 3$ (%)	SILog (%)
1	DenseNet	5 UpProj	False	Adam			4	5.73	71.66	90.46	96.71	10.06
2	DenseNet	5 UpProj	False	SGD			6	4.72	80.41	93.51	97.82	8.76
3	MobileNet	5 UpConv	True	Adam			3	4.90	76.72	92.97	97.20	9.32
4	MobileNet	5 UpConv	True	SGD			8	4.25	83.10	94.56	98.14	8.06
5	MobileNet	5 UpProj	True	SGD			4	4.58	81.57	93.89	97.90	8.47
6	MobileNet	4 UpProj	False	SGD			5	4.53	81.67	93.76	97.67	8.53
7	MobileNet	5 UpConv	True	SGD	True		3	6.76	70.76	87.47	94.81	11.48
8	ResNet18	4 UpProj	True	Adam			6	4.53	84.13	94.03	97.53	8.51
9	ResNet18	4 UpProj	True	SGD			13	3.21	92.07	97.85	99.26	5.72
10	ResNet18	4 UpConv	True	SGD			12	3.95	86.90	96.03	98.56	7.25
11	ResNet18	4 UpProj	True	SGD		Pretrained	7	4.47	81.23	93.72	97.49	8.34
12	ResNet18	4 Upconv	True	SGD	True		5	3.82	85.72	96.01	98.68	7.23
13	ResNet18	4 UpProj	True	SGD	True	Frozen	3	3.19	91.66	97.88	99.28	5.73
14	ResNet18	4 UpProj	False	SGD	True	G+D	4	3.21	91.76	97.82	99.25	5.86
15	ResNet18	4 UpProj	True	SGD	True	RGB+D+G	8	3.26	91.25	97.49	99.13	6.05
16	ResNet18	4 UpProj	True	SGD	True	Pretrained	8	3.92	85.05	95.49	98.47	7.36
17	ResNet34	4 UpProj	True	SGD	True		2	4.23	82.34	93.71	97.29	8.38

When comparing results of similar architectures, the networks were trained for 5 epochs in order to determine which variation produces better results, with the exception of architectures number 8 through 11, where 10 training epochs were first used, due to their faster training process, in order to better understand which architecture was the best. The better performing architectures, were then trained for more epochs in order to determine the best result that can be achieved using them, such as architecture number 9.

As previously mentioned, two optimizers were used and tested, the SGD and Adam. In order to understand which optimizer works better in this context, both were tested in pairs using the same architectures. When using the DenseNet as the encoder (architecture number 1 and 2), the SGD optimizer outperformed the Adam optimizer in every metric, achieving a RMSE of 5.73m at epoch number 4, whereas the Adam optimizer achieved the best RMSE of 4.72m at epoch number 6. The same conclusion can be obtained in the case where the encoder is the MobileNetV2 (architectures 3 and 4) and where the encoder is the ResNet (architectures 8 and 9). Taking this into consideration,

most architectures here presented use the SGD as their optimizer, including the architectures that obtained the best results.

Regarding the decoder, the Up-Projection block presented in 3.23 was mostly used by the architectures that use the ResNet network as the feature encoder, as it was verified that it is a better option than the Up-Convolution block. This can be seen by comparing the results obtained by the architectures number 9 and 10, where the first shows a considerably better result on every metric used, achieving a RMSE of 3.21m at epoch number 13 as opposed to a RMSE of 3.95m at epoch number 12. As for the architectures that use the MobileNetV2 as the encoder, the results obtained using the Up-Convolution block presented in 3.21 obtained similar results to the Up-Projection block and both were used in different architectures.

Transfer learning methods were also used and tested, where the parameters of the ResNet network trained for classification were chosen as the starting values for the weights of the decoder. This has the objective of taking advantage of the feature extracting capabilities that were previously trained. However, in this context, using this method did not lead to better results, as seen when comparing the architectures number 9 and 11, due to the environmental and context differences between the datasets used. Architecture number 9 obtained a RMSE of 3.21m, as opposed to a RMSE of 4.47m achieved by architecture 11. As a result, the parameters of every other architecture were initialized following the initialization method proposed by Kaiming *et al.* [49], as previously mentioned in 3.3.2.

4.1.1 Evolution of the RMSE in different Architectures

DenseNet

To better compare the developed architectures, the evolution of the RMSE metric during the training process will be presented and discussed, where the values of the RMSE error calculated at the end of each epoch were combined into a single graph.

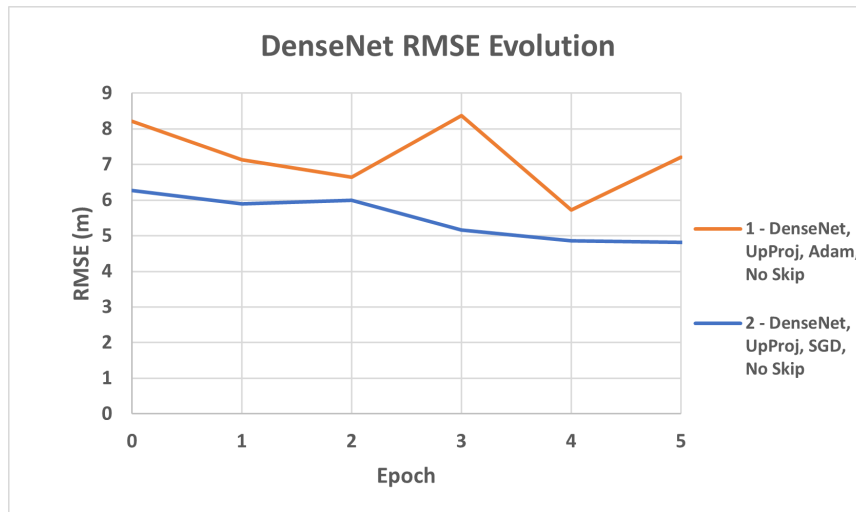


Figure 4.2: DenseNet RMSE evolution (Architectures number 1 and 2).

The graph presented in figure 4.2 shows the evolution of the RMSE error for the architectures number 1 and 2 where the DenseNet-121 is used as the feature encoder. As previously concluded, the architecture number 2 shows a better overall result, achieving a RMSE value of 4.72m, a δ_1 of 80.41% and a SILog of 8.76% as opposed to 5.73m, 71.66% and 10.06% respectively, obtained using architecture number 1, making SGD the best option. When comparing the results obtained using the DenseNet-121 as the encoder to the MobileNetV2 or the ResNet, the former presents worse results, both using the metrics and visual inspection of the predicted dense depth map.

Taking this into consideration, for the type of data used as the input of the neural network in this context, the DenseNet-121 was considered to be an inferior option and was discarded as an option for the best architecture to be used.

MobileNetV2

The graph presented in figure 4.3 shows the evolution of the RMSE error for the architectures number 3 through 7, where the MobileNetV2 is used as the feature encoder. Similarly to the case of the DenseNet, as previously mentioned, the architecture number 4 that uses SGD as the optimizer, achieves better results than the architecture number 3 that uses Adam. In the first 5 epochs, the first achieves the best RMSE value of 4.41m, while the latter achieves the best RMSE value of 4.53m.

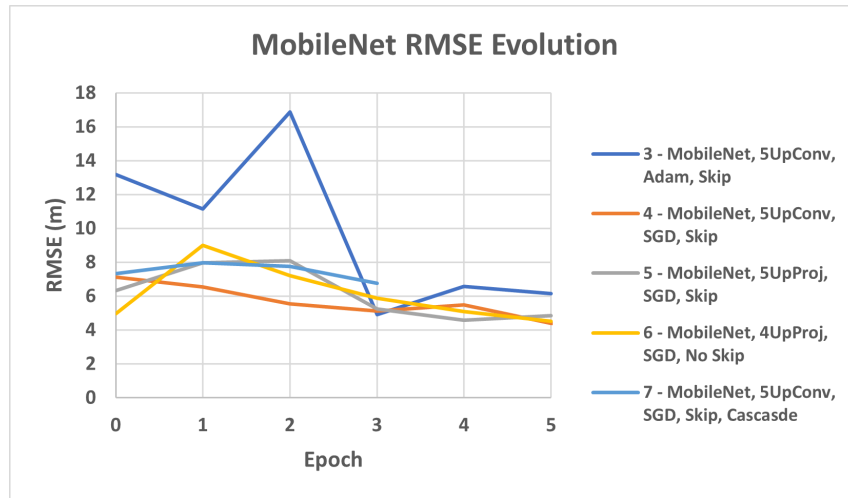


Figure 4.3: MobileNetV2 RMSE evolution (Architectures number 3 to 7).

Between all the architectures that use the MobileNetV2 as the encoder, the architecture number 4 that uses 5 Up-Convolution blocks as the decoder with skip connections, achieved the overall best results in every metric used, obtaining a RMSE value of 4.41m in the first 6 epochs and a best RMSE value of 4.25m. Even though the architecture number 7 uses the cascade model presented in 3.26, in addition to taking a significantly higher computational power and training time due to more parameters, the results obtained were not satisfactory, achieving the worse quantitative results between these architectures, with a RMSE value of 6.76m and δ_1 of 70.76%, while also performing the worse when capturing the global scale of the image, with a SILog of 11.48%. In

general, the architectures that use MobileNetV2 as the encoder performed better than the ones that use DenseNet, but achieved worse results than the ones that use ResNet.

ResNet

The graph presented in figure 4.4 shows the evolution of the RMSE error for the architectures 8 through 11, where the ResNet18 network is used as the feature encoder. These represent the architectures that do not use the cascade model that will be later presented in figure 4.5. Similarly to the DenseNet and MobileNetV2 architectures, when using the ResNet as the encoder, the SGD optimizer obtains better results than the Adam, as can be concluded by comparing the architectures 8 and 9, where the only difference between them is the optimizer used. Architecture 9 is one of the best performing models, achieving a RMSE value of 4,01m in the first 7 epochs, as opposed to a RMSE value of 4.53m achieved by architecture 8 that uses Adam optimizer. Out of all 17 presented architectures, it also achieved the best $\delta 1$ and $SILog$, with a value of 92.07% and 5.72% respectively, and the second best RMSE value. It is also interesting to compare architectures 9 and 10, where the Up-Projection and Up-Convolution blocks are used as the decoder, respectively. The model that uses the Up-Convolution block achieved, in the first 10 epochs, results that are considerably worse (RMSE value of 4,02m) than the ones achieved by the Up-Projection (RMSE value of 3,42m), making the latter the preferable option.

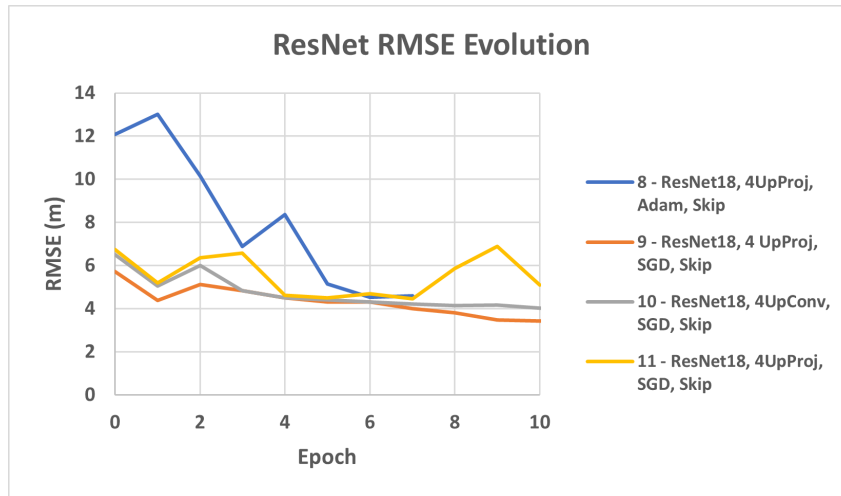


Figure 4.4: ResNet RMSE evolution (Architectures number 8 to 11).

The graph presented in figure 4.5 shows the evolution of the RMSE error for the architectures 12 through 17, where the ResNet18 neural network is used as the feature encoder and the cascade architecture presented in 3.26 is adopted. Out of these, in the first 5 training epochs, architecture 13 achieved the best RMSE with a value of 3,19m at epoch 3, which also corresponds to the overall best value obtained. This architecture uses model number 9 as the first encoder-decoder model of the cascade, with its parameters frozen. This was done with the objective of improving the results obtained by the first encoder-decoder model previously trained, by using its output as the input of the second model. This proved to improve the final results, achieving, in addition to the RMSE,

the overall best δ_2 and δ_3 , with a value of 97.88% and 99.28% respectively, making it the overall best architecture. However, these values are close to the ones obtained using architecture number 9, thus both can be considered equally the best models.

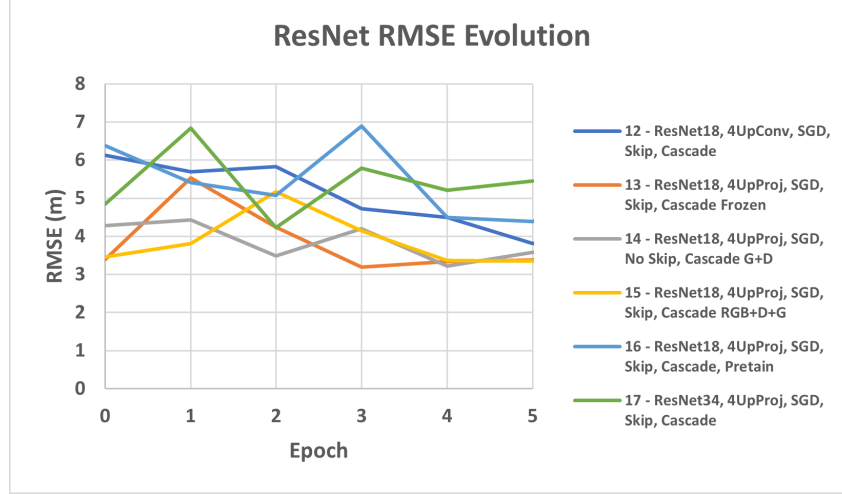


Figure 4.5: ResNet RMSE evolution (Architectures number 12 to 17).

However, as this architecture has double the parameters than the simple encoder-decoder model, the training process is slower due to longer prediction times and requires higher computational power. Architecture 16 uses a similar method with the difference that the first model's parameters are not frozen, achieving considerably worse results than the frozen counterpart. Architecture 14 uses the output of the first model combined with the grayscale version of the original RGB image as the input for the second encoder-decoder model. This approach achieved results similar to the ones obtained by the best architectures, with a RMSE value of 3.22m. Architecture 17, uses a ResNet-34 network which has more layers and consequently more trainable parameters than the ResNet-18. As can be observed in figure 4.5, after 5 training epochs, the best achieved RMSE has a value of 4.23m, which is significantly lower than the other architectures. It is possible to conclude that more trainable parameters in this case does not result in a better output.

4.1.2 Dense Depth Maps

In this section, the final dense depth maps obtained by the better performing architectures are presented and discussed. Each figure contains different images: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top. In the predicted dense depth map and ground truth images, dark blue represents low depth values, while red represents points with a high depth value. Figure 4.6 represents the results obtained by using the architecture number 2, 4 and 9. From visual inspection of the calculated dense depth map of architecture 2, it is possible to conclude that the object boundaries are not clear, and each object's depth values are mixed with each other. This problem is minimized in the architectures that obtained better results, where one of the reasons for the improvement is due to the skip connections from the encoder to

the decoder that propagate texture information, as well as object boundaries and other information that may be lost during the encoding-decoding process.

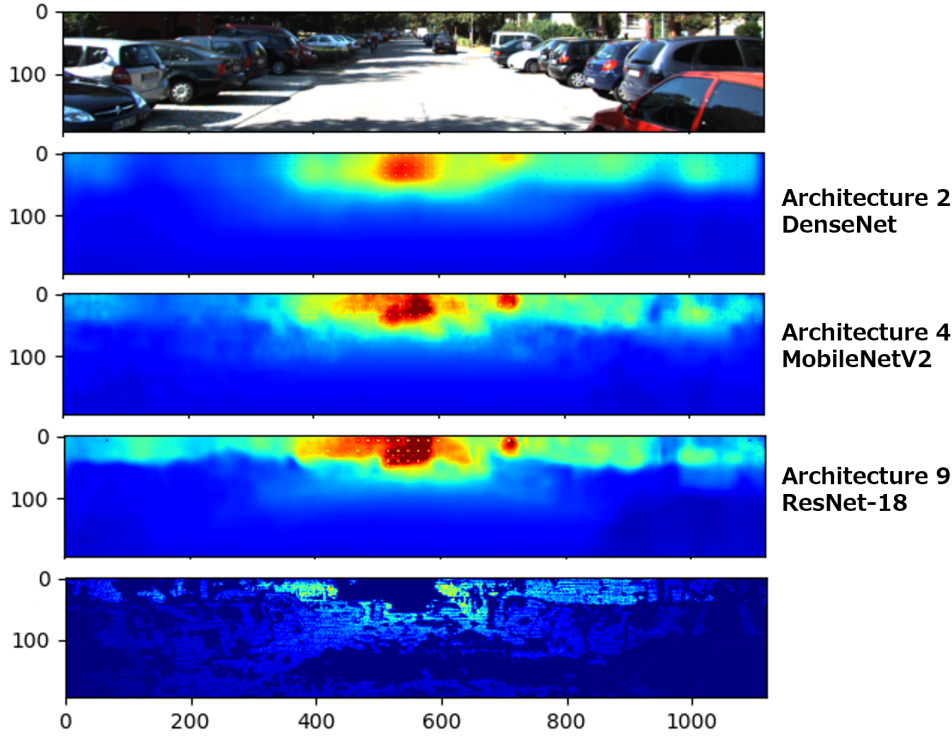


Figure 4.6: Dense depth map obtained using the architecture number 2 - DenseNet, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.

When compared to the DenseNet, it is possible to conclude that the results obtained using architecture number 4 are more accurate, and are visually closer to the ground truth. When evaluating architecture number 9, where the ResNet18 model is used as the feature encoder, it is possible to conclude from visual inspection that the object boundaries are clearer than the previous examples and the overall depth map is closer to the ground truth.

Figures 4.7, 4.8 and 4.9 represent some of the results obtained using the architecture number 13, where architecture 9 is used as the first encoder-decoder model present in the cascade architecture. The results here obtained are similar to the ones obtained using the single encoder-decoder model used by architecture 9. This was expected, as the latter was used as a basis for the former one. Both architectures present similar metric values with the cascade model having an improvement of the RMSE by 0.02m, δ_2 by 0.03% and δ_3 by 0.02%, while having a worse δ_1 by 0.03% and SILog by 0.01%. These values do not justify the heavier cascade architecture, and the single encoder-decoder was deemed the overall best alternative.

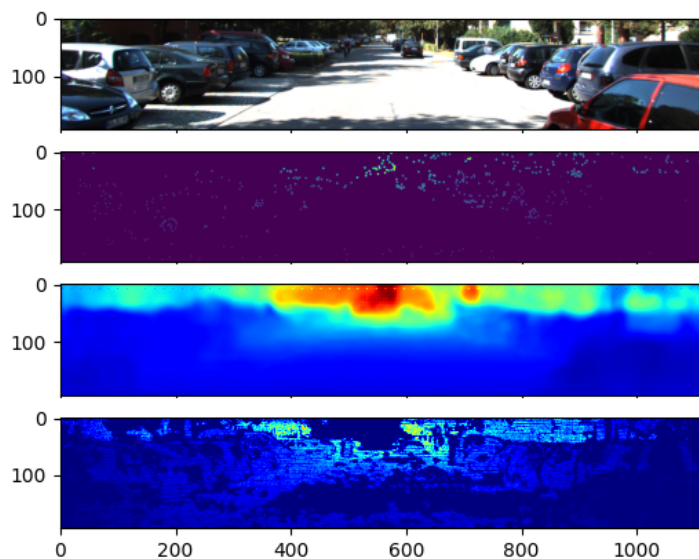


Figure 4.7: Dense depth map obtained using the architecture number 13 - ResNet18 cascade, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.

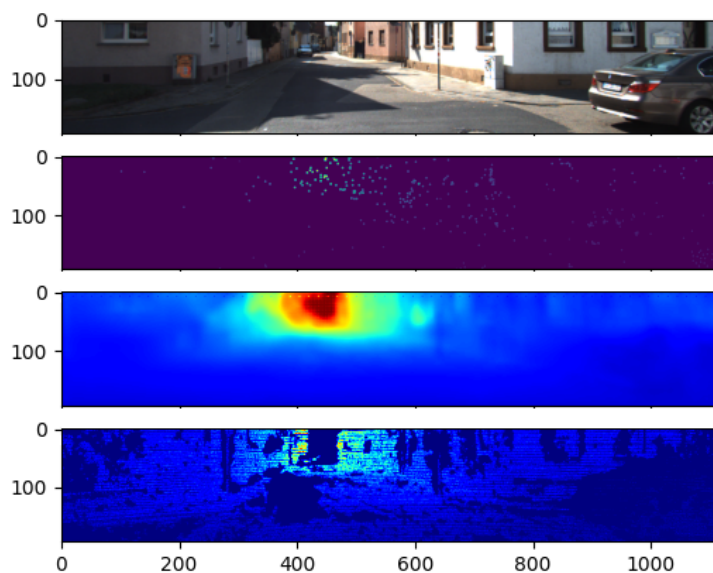


Figure 4.8: Dense depth map obtained using the architecture number 13 - ResNet18 cascade, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.

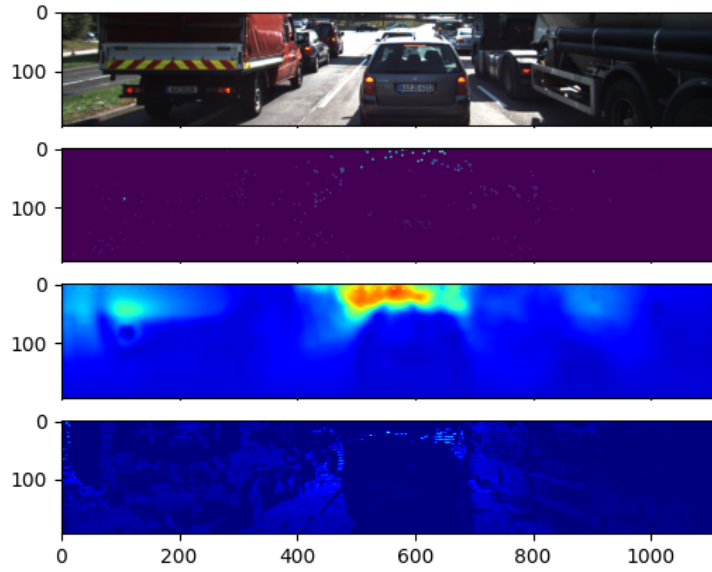


Figure 4.9: Dense depth map obtained using the architecture number 13 - ResNet18 cascade, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.

Conclusions

From the results obtained using the different models and methods, the developed architectures were proved to be successful in calculating a dense depth map from sparse depth points and RGB images. It is possible to conclude that in the context of this work, the ResNet-18 is the feature encoder that produces the best results. As for the decoder, a sequence of 4 Up-Projection blocks followed by an unpool layer proved to generally be the best option. Comparisons between models also proved that using the BerHu loss function and the SGD as the optimizer achieves better results than the tested counterparts. Skip connections between the encoder and the decoder resulted in better results according to the used metrics and better overall results by visual inspection of the generated dense depth maps. Finally, it is possible to conclude that in general, the cascade architecture produces good results, with the cost of slower learning process and higher computational power.

4.2 Maritime Dataset - ROAM@CRAS¹

As previously mentioned, the projected architectures were initially developed and trained using the KITTI dataset [7] to validate their performance. These architectures were then used in the maritime environment using real data collected by the SENSE ASV [8]. In this section, the results obtained using this dataset are presented and discussed.

¹<https://rdm.inesctec.pt/vi/dataset/nis-2020-002>

The data was separated using a similar ratio of test and train images that was used in the previous section, using 14767 images for the training process and 455 images for the validation process.

Three different procedures were used and tested: the architectures that obtained the best results (one from each different encoder) using the KITTI dataset [7] were trained, starting from random weights, following the initialization proposed by Kaiming *et al.* [49] (architectures number 2, 4 and 9 from table 4.1); direct inference using the architecture number 9 from table 4.1, where the model and its trained weights were imported and directly used to obtain results using the maritime dataset; and finally, the transfer learning method was used for the architecture number 9, where the model was imported and retrained using the maritime dataset, using the previously trained weights as the starting point. Table 4.2 presents a summary of the results obtained using the aforementioned architectures, where A through C represent the architectures that did not use transfer learning, architecture D represents the results obtained directly using the architecture number 9 from table 4.1 and architecture E represents the same model, retrained using the maritime dataset. The models were trained using the BerHu loss function and a batch size of 1, due to hardware limitations.

Table 4.2: Summary of the results obtained using the maritime dataset. Higher values of $\delta 1$, $\delta 2$ and $\delta 3$ represent better results, while lower values of $RMSE$ and $SILog$ are better.

ID	Encoder	Decoder	Skip	Optim.	Extra	Epoch	RMSE (m)	$\delta 1$ (%)	$\delta 2$ (%)	$\delta 3$ (%)	SILog (%)
A	DenseNet	5 UpProj	False	SGD		4	15.26	31.25	59.53	77.61	18.57
B	MobileNet	5 UpConv	True	SGD		4	36.77	4.74	8.88	13.13	25.00
C	ResNet18	4 UpProj	True	SGD		3	11.89	49.42	76.92	86.08	16.62
D	ResNet18	4 UpProj	True	—	Direct Inference	—	24.49	5.96	14.67	28.49	18.51
E	ResNet18	4 UpProj	True	SGD	Transfer Learning Retrained	2	15.44	34.36	62.58	77.57	17.80

Figure 4.10 represents the results obtained using the architecture A, B and C from table 4.2. From visual inspection, it is possible to conclude that the trained model was not able to correctly predict a dense depth map from the input data. Architecture A achieved a RMSE value of 15.26m, which is a high error for this problem. It is also possible to observe that the ground truth presents a small amount of points, due to the number of channels of the LiDAR used and the lack of objects in the environment. The lack of reference points and features in this environment also lead to a small amount of points calculated using the feature matching algorithms. The combination of lack of information on the ground truth and the sparse depth input, leads to a training process that does not converge.

Figure 4.11 represents a comparison between architectures D and E from table 4.2. From visual inspection and analysis of the metrics, it is possible to conclude that directly using the previously trained model using the KITTI dataset [7] in this environment results in considerably worse results (RMSE value of 24.49m in the maritime environment, as opposed to a RMSE of 3.21m in the land environment). This is a direct result of the completely different environment, along with the different scale of distances and the sparsity of the input depth points obtained by feature matching. The right column in figure 4.11 represents the results obtained using transfer learning, showing the evolution from direct inference (left column) to the same model trained in

the maritime environment (right column). From visual inspection, it is possible to observe that the model diverged, resulting in a prediction that is visually similar to the other models presented in this section.

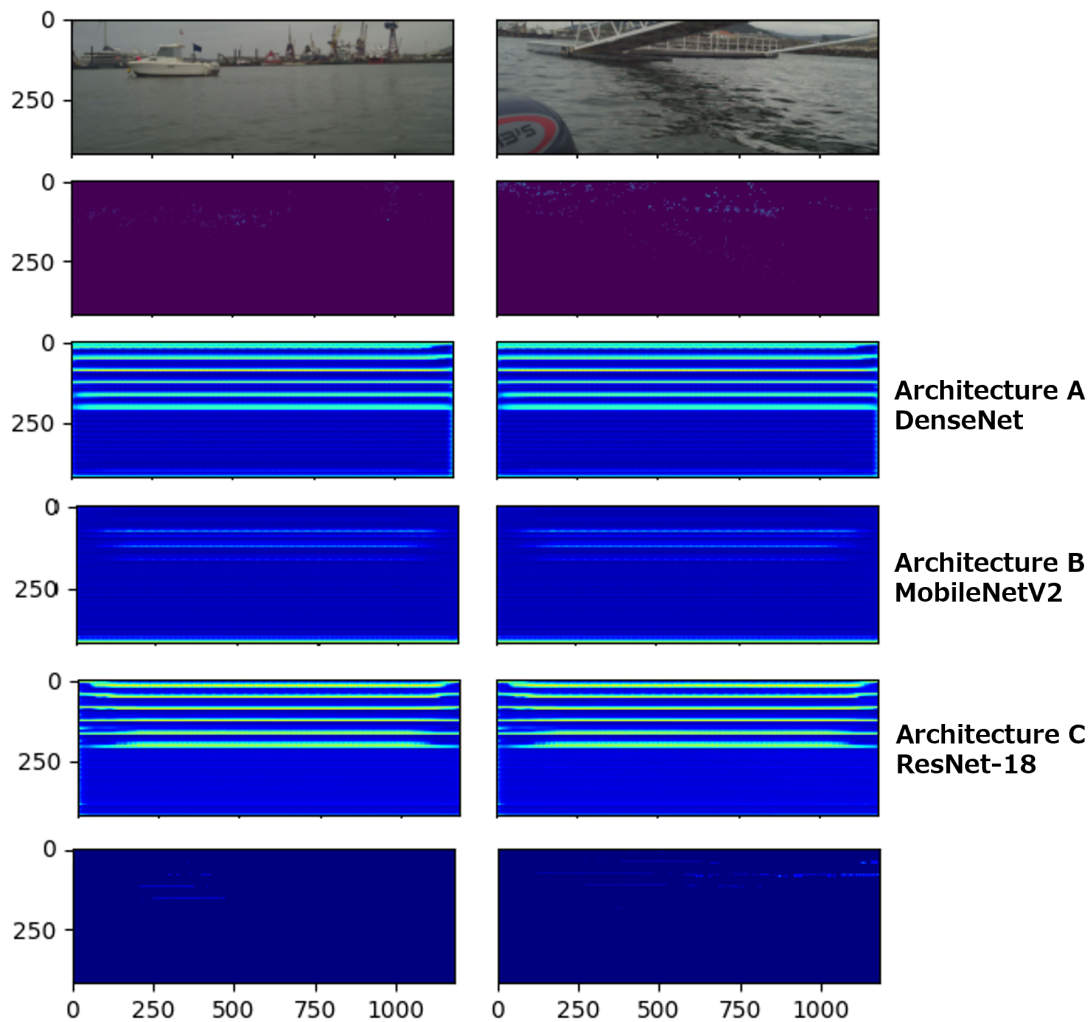


Figure 4.10: Dense depth map obtained using architecture A - DenseNet; B - MobileNetV2 and C - ResNet18, where the 6 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the 3 predicted dense depth map and the ground truth, displayed in this order starting from the top.

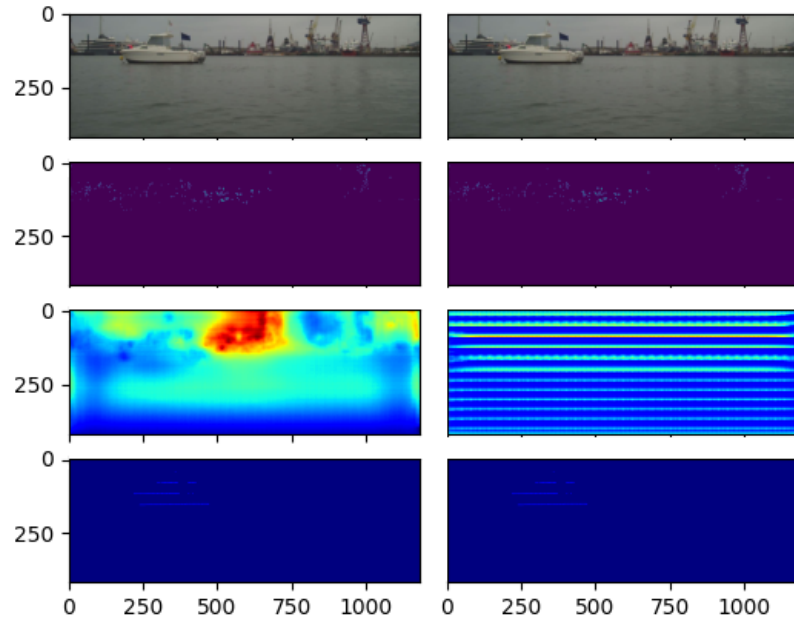


Figure 4.11: Dense depth map obtained using direct inference (left column) and dense depth map after using transfer learning (right column), using architecture number 9 from table 4.1, where the 4 different images are: the original RGB image obtained by the left camera; the sparse depth points obtained by feature matching; the predicted dense depth map and the ground truth, displayed in this order starting from the top.

Conclusions

When testing the developed architectures and methods in the maritime environment, it is important that the ground truth has enough information to train the neural network, as well as a depth input that is able to provide more depth cues to help predict a dense depth map. The lack of points in the ground truth leads to a loss function that is calculated using a reduced amount of points in a small area of the image, which results in a training process that optimizes the parameters of the model in the direction of producing a depth map that follows the ground truth. As in most cases the ground truth presents a small number of points forming a few horizontal lines, the training of the network results in the prediction of a depth map that also contains horizontal lines with a specific depth value, as observable in every depth map presented in this section. Ultimately, in this situation, the model is not able to converge. The inherent lack of reference points, features and texture information in the captured images in the maritime environment also lead to a reduced amount of points obtained using the feature-base stereo algorithm, which heavily contributes to the difficulties faced in this kind of environment. In section 5.1, crucial future work points are suggested to mitigate the problems presented in this section.

Chapter 5

Conclusions and Future Work

In this dissertation, a machine learning architecture was developed in order to fuse information provided by different sensors with the objective of improving the situational awareness of ASVs. This contributes to a better understanding of the vessel's surroundings, improving the safety and costs of different maritime missions, such as freight transportation and the inspection of off-shore structures. A good perception of the environment leads to reduced risks of collisions, preventing casualties, damage to the vehicle and damage to the environment.

In order to develop and understand what works well in the context of this dissertation, different models and methods were presented, discussed and tested. The results obtained using the developed architectures and each of their variations were then exposed and discussed.

The proposed architecture consists of using an encoder-decoder model, where the encoder is responsible for extracting important features from the input while the decoder interprets and up-samples the calculated features, obtaining a dense depth map as the output of the neural network. The sparse depth map, used as an input of the network was obtained using feature matching from a stereo camera, taking advantage of the ORB and SIFT keypoint detectors and descriptor extractors. This input was then fused with the RGB image provided by the left camera. The ground truth used to train the neural network was based on the point cloud acquired by the 3D LiDAR, where a projection to the 2D image plane was applied.

The proposed architectures were first developed and tested using a real dataset that was previously acquired in a road environment to prove its functionality. Real maritime data was then acquired and used to test the previously developed architectures. Results show that in the context of this dissertation, using the ResNet18 as a feature encoder and 4 Up-Projection blocks as the decoder, with skip connection from the encoder to the decoder was the best option from the tested architectures, obtaining a SILog error of 5,72% in the road environment. During the training process, the BerHu loss function and the SGD optimizer were used. When using the developed architectures on the maritime dataset, results show that the points provided by the LiDAR sensor with 16 channels in the maritime environment are not enough to serve as a ground truth for the training process. These results also demonstrate that using feature-based stereo to obtain sparse depth information does not perform well in this environment, due to the lack of features, points

of reference and texture information. The steps presented as future work have the objective of solving this problem faced during the development of this dissertation.

5.1 Future Work

With the objective of continuing the work developed during this dissertation, test and adjust the developed architectures using real data obtained in a maritime environment, the following points are suggested as future work:

- Use a LiDAR with more channels in order to obtain a ground truth with enough points to be used for the training process.
- Develop an alternative to the feature-base stereo method, to overcome the problems faced when using this technique in the maritime environment.
- Use and test a second LiDAR in order to obtain sparse depth points to be used as the input of the neural network, replacing the points obtained using feature-based matching.
- Test and adjust the developed architectures using the maritime data obtained using the sensors suggested in point one and two.
- Integrate the neural network within the SENSE ASV [8] to test the algorithm in real time.

References

- [1] Boris Crnokić, Miroslav Grubisic, and Tomislav Volaric. Different applications of mobile robots in education. *International Journal on Integrating Technology in Education*, 6:15–28, 09 2017.
- [2] Anita Rothblum, David Wheal, Stuart Withington, Scott Shappell, Douglas Wiegmann, William Boehm, and Marc Chaderjian. Human factors in incident investigation and analysis. *Proceedings of the 2nd International Workshop on Human Factors in Offshore Operations (HFW2002)*, 04 2002.
- [3] Pedro Nuno Leite, Renato Jorge Silva, Daniel Filipe Campos, and Andry Maykol Pinto. Dense disparity maps from rgb and sparse depth information using deep regression models. In Aurélio Campilho, Fakhri Karray, and Zhou Wang, editors, *Image Analysis and Recognition*, pages 379–392, Cham, 2020. Springer International Publishing.
- [4] Gangqiang Zhao, Xuhong Xiao, Junsong Yuan, and Gee Wah Ng. Fusion of 3d-lidar and camera data for scene parsing. *Journal of Visual Communication and Image Representation*, 25(1):165 – 183, 2014.
- [5] Ramon Brena, Antonio Aguilera, Luis Trejo, Erik Molino Minero Re, and Oscar Mayora. *Sensors*, 20:2350, 04 2020.
- [6] Jonathan Schmidt, Mário R. G. Marques, Silvana Botti, and Miguel A. L. Marques. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials*, 5(1):83, Aug 2019.
- [7] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- [8] Daniel Filipe Campos, Aníbal Matos, and Andry Maykol Pinto. Multi-domain inspection of offshore wind farms using an autonomous surface vehicle. *SN Applied Sciences*, 3(4):455, Mar 2021.
- [9] Hadi Ghaderi. Chapter eleven - wider implications of autonomous vessels for the maritime industry: Mapping the unprecedented challenges. In Dimitris Milakis, Nikolas Thomopoulos, and Bert van Wee, editors, *Policy Implications of Autonomous Vehicles*, volume 5 of *Advances in Transport Policy and Planning*, pages 263 – 289. Academic Press, 2020.
- [10] D. Kortenkamp. Perception for mobile robot navigation: A survey of the state of the art. *NASA Technical Reports Server (NTRS)*, 1994.
- [11] M. B. Alatisse and G. P. Hancke. A review on challenges of autonomous mobile robot and sensor fusion methods. *IEEE Access*, 8:39830–39846, 2020.

- [12] Sebastian Thrun, Mike Montemerlo, and Hendrik Dahlkamp et al. *Stanley: The Robot That Won the DARPA Grand Challenge*, pages 1–43. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
- [13] Dza-Shiang Hong, Hung-Hao Chen, Pei-Yung Hsiao, Li-Chen Fu, and Siang-Min Siao. Crossfusion net: Deep 3d object detection based on rgb images and point clouds in autonomous driving. *Image and Vision Computing*, 100:103955, 2020.
- [14] F. A. Kruse. Combined swir and lwir mineral mapping using master/aster. In *IEEE International Geoscience and Remote Sensing Symposium*, volume 4, pages 2267–2269 vol.4, 2002.
- [15] F. Kruse. Integrated visible and near infrared, shortwave infrared, and longwave infrared (vnir-swir-lwir), full-range hyperspectral data analysis for geologic mapping. *Journal of Applied Remote Sensing*, 9:096005, 09 2015.
- [16] Ove Steinvall, Daniel Svedbrand, and Thomas Svensson. Optical sensing during low visibility conditions. In Gary W. Kamerman and Ove Steinvall, editors, *Electro-Optical Remote Sensing XIII*, volume 11160, pages 134 – 153. International Society for Optics and Photonics, SPIE, 2019.
- [17] Nicolas Pinchon, Mohamed Ibn Khedher, Olivier Cassignol, Adrien Nicolas, Frédéric Bernardin, Patrick Leduc, Jean-Philippe Tarel, Roland Brémond, Emmanuel Bercier, and G. Julien. All-weather vision for automotive safety: which spectral band? *International Conference Night Drive Tests and Exhibition*, 2016.
- [18] R. Heinzler, P. Schindler, J. Seekircher, W. Ritter, and W. Stork. Weather influence and classification with automotive lidar sensors. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1527–1534, 2019.
- [19] Francisca Rosique, Pedro J. Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3), 2019.
- [20] Y. Li and J. Ibanez-Guzman. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE Signal Processing Magazine*, 37(4):50–61, 2020.
- [21] Xin Wang, HuaZhi Pan, Kai Guo, Xinli Yang, and Sheng Luo. The evolution of lidar and its application in high precision measurement. *IOP Conference Series: Earth and Environmental Science*, 502:012008, 06 2020.
- [22] M. Elhousni and X. Huang. A survey on 3d lidar localization for autonomous vehicles. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1879–1884, 2020.
- [23] O. Ervan and H. Temeltaş. Downsampling of a 3d lidar point cloud by a tensor voting based method. In *2019 11th International Conference on Electrical and Electronics Engineering (ELECO)*, pages 880–884, 2019.
- [24] Gérard Medioni, Chi-Keung Tang, and Mi-Suen Lee. Tensor voting: Theory and applications. *Proceedings of RFIA*, 01 2000.

- [25] E. Maltezos, A. Doulamis, N. Doulamis, and C. Ioannidis. Building extraction from lidar data applying deep convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters*, 16(1):155–159, 2019.
- [26] T. Peynot, J. Underwood, and S. Scheduling. Towards reliable perception for unmanned ground vehicles in challenging conditions. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1170–1176, 2009.
- [27] A. Filgueira, H. González-Jorge, S. Lagüela, L. Díaz-Vilariño, and P. Arias. Quantifying the influence of rain in lidar performance. *Measurement*, 95:143 – 148, 2017.
- [28] S. Hasirlioglu, I. Doric, A. Kamann, and A. Riener. Reproducible fog simulation for testing automotive surround sensors. In *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pages 1–7, 2017.
- [29] H. Essen, S. Stanko, R. Sommer, A. Wahlen, R. Brauns, J. Wilcke, W. Johannes, A. Tessmann, and M. Schlechtweg. A high performance 220-ghz broadband experimental radar. In *2008 33rd International Conference on Infrared, Millimeter and Terahertz Waves*, pages 1–1, 2008.
- [30] Rolls Royce. Aawa position paper: Remote and autonomous ships - the next steps. 2016.
- [31] D. L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.
- [32] Federico Castanedo. A review of data fusion techniques. *TheScientificWorldJournal*, 2013:704504, 10 2013.
- [33] Bahador Khaleghi, Alaa Khamis, Fakhreddine O. Karray, and Saiedeh N. Razavi. Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44, 2013.
- [34] X. Wang, L. Xu, H. Sun, J. Xin, and N. Zheng. On-road vehicle detection and tracking using mmw radar and monovision fusion. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):2075–2084, 2016.
- [35] L. Stanislas and M. Dunbabin. Multimodal sensor fusion for robust obstacle detection and classification in the maritime robotx challenge. *IEEE Journal of Oceanic Engineering*, 44(2):343–351, 2019.
- [36] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.
- [37] H. Courtois and N. Aouf. Fusion of stereo and lidar data for dense depth map computation. In *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 186–191, 2017.
- [38] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. pages 25–38, 12 2010.
- [39] M. Rayyan Akhtar, Huabiao Qin, and Guancheng Chen. Velodyne LiDAR and monocular camera data fusion for depth map and 3D reconstruction. In *Eleventh International Conference on Digital Image Processing (ICDIP 2019)*, volume 11179, pages 87 – 97. SPIE, 2019.

- [40] Zhen-shan BAO, Bo-wen LI, and Wen-bo ZHANG. Robustness of tof and stereo fusion for high accuracy depth maps. *DEStech Transactions on Computer Science and Engineering*, 01 2019.
- [41] K. Zhang, J. Xie, N. Snavely, and Q. Chen. Depth sensing beyond lidar range. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1689–1697, 2020.
- [42] X. Zhao, W. Chen, Z. Liu, X. Ma, L. Kong, X. Wu, H. Yue, and X. Yan. Lidar-tof-binocular depth fusion using gradient priors. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 2024–2029, 2020.
- [43] Ankit Dhall, Kunal Chelani, Vishnu Radhakrishnan, and K. M. Krishna. Lidar-camera calibration using 3d-3d point correspondences. *ArXiv eprints*, 2017.
- [44] H. H. Cholakkal, S. Mentasti, M. Bersani, S. Arrigoni, M. Matteucci, and F. Cheli. Lidar - stereo camera fusion for accurate depth estimation. In *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, pages 1–6, Nov 2020.
- [45] C. Guindel, J. Beltrán, D. Martín, and F. García. Automatic extrinsic calibration for lidar-stereo vehicle sensor setups. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2017.
- [46] S. Daw and R. Basak. Machine learning applications using waikato environment for knowledge analysis. In *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, pages 346–351, 2020.
- [47] Xian-Da Zhang. *Machine Learning*, pages 223–440. Springer Singapore, Singapore, 2020.
- [48] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629, Aug 2018.
- [49] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.
- [50] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.
- [52] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.
- [53] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567v3*, 2015.

- [54] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [55] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.
- [56] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv: 1801.04381*, 2018.
- [57] K. Gadzicki, R. Khamsehashari, and C. Zetzsche. Early vs late fusion in multimodal convolutional neural networks. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pages 1–6, 2020.
- [58] Cees Snoek, Marcel Worring, and Arnold Smeulders. Early versus late fusion in semantic video analysis. *Proceedings of the 13th ACM International Conference on Multimedia, MM 2005*, pages 399–402, 01 2005.
- [59] C. Fu, C. Mertz, and J. M. Dolan. Lidar and monocular camera fusion: On-road depth completion for autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 273–278, 2019.
- [60] Matthew Zeiler, Graham Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. *Proceedings of the IEEE International Conference on Computer Vision*, 2011:2018–2025, 11 2011.
- [61] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgb-d images. In *In Proceedings of the 12th European Conference on Computer Vision*, 2012.
- [62] Wouter Van Gansbeke, D. Neven, Bert De Brabandere, and L. Gool. Sparse and noisy lidar completion with rgb guidance and uncertainty. *2019 16th International Conference on Machine Vision Applications (MVA)*, pages 1–6, 2019.
- [63] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger. Sparsity invariant cnns. In *2017 International Conference on 3D Vision (3DV)*, pages 11–20, 2017.
- [64] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2018.
- [65] T. Wang, H. Hu, C. H. Lin, Y. Tsai, W. Chiu, and M. Sun. 3d lidar and stereo fusion using stereo matching network with conditional cost volume normalization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5895–5902, 2019.
- [66] J. Tang, F. P. Tian, W. Feng, J. Li, and P. Tan. Learning guided convolutional network for depth completion. *IEEE Transactions on Image Processing*, 30:1116–1129, 2021.
- [67] Jinsun Park, Kyungdon Joo, Zhe Hu, Chi-Kuei Liu, and In So Kweon. Non-local spatial propagation network for depth completion. In *European Conference on Computer Vision (ECCV)*, 2020.

- [68] Xinjing Cheng, Peng Wang, Guan Chenye, and Ruigang Yang. Cspn++: Learning context and resource aware convolutional spatial propagation networks for depth completion. *Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*, 2020.
- [69] Shanshan Zhao, Mingming Gong, Huan Fu, and Dacheng Tao. Adaptive context-aware multi-modal network for depth completion. *arXiv preprint arXiv:2008.10833*, 2020.
- [70] Y. Chen, B. Yang, M. Liang, and R. Urtasun. Learning joint 2d-3d representations for depth completion. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10022–10031, 2019.
- [71] Pedro Nuno Leite and Andry Maykol Pinto. Exploiting motion perception in depth estimation through a lightweight convolutional neural network. *IEEE Access*, 9:76056–76068, 2021.
- [72] Renato Silva, Pedro Leite, Daniel Campos, and Andry M. Pinto. Hybrid approach to estimate a collision-free velocity for autonomous surface vehicles. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–6, 2019.
- [73] Daniel Filipe Campos, Aníbal Matos, and Andry Maykol Pinto. An adaptive velocity obstacle avoidance algorithm for autonomous surface vehicles. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8089–8096, 2019.
- [74] Pedro Leite, Renato Silva, Aníbal Matos, and Andry Maykol Pinto. An hierarchical architecture for docking autonomous surface vehicles. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–6, 2019.
- [75] Maria Inês Pereira, Rafael Marques Claro, Pedro Nuno Leite, and Andry Maykol Pinto. Advancing autonomous surface vehicles: A 3d perception system for the recognition and assessment of docking-based structures. *IEEE Access*, 9:53030–53045, 2021.
- [76] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 239–248. IEEE, 2016.
- [77] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.
- [78] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.
- [79] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.
- [80] Chuan Luo, Wei Yang, Panling Huang, and Jun Zhou. Overview of image matching based on ORB algorithm. *Journal of Physics: Conference Series*, 1237(3):032020, jun 2019.
- [81] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2004.
- [82] Laurent Zwald and Sophie Lambert-Lacroix. The berhu penalty and the grouped effect. *arXiv preprint arXiv:1207.6868*, 2012.

- [83] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2017.