UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Geometric computer vision meets deep learning for autonomous driving applications

Javier García López

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Doctoral Programme

AUTOMATIC CONTROL, ROBOTICS AND VISION

Ph.D. Thesis

GEOMETRIC COMPUTER VISION MEETS DEEP LEARNING FOR AUTONOMOUS DRIVING APPLICATIONS

Javier García López

Advisors:
Antonio Agudo and Francesc Moreno-Noguer

Barcelona, December 2020

Geometric computer vision meets deep learning for autonomous driving applications

A thesis submitted to the Universitat Politècnica de Catalunya
to obtain the degree of Doctor of Engineering

Doctoral programme:
Automatic Control, Robotics and Vision

This thesis was completed at:
Institut de Robòtica i Informàtica Industrial, CSIC-UPC

Thesis advisors:
Antonio Agudo and Francesc Moreno-Noguer

To my family.

Abstract

This dissertation intends to provide theoretical and practical contributions on the development of deep-learning algorithms for autonomous driving applications. The research is motivated by the need of deep neural networks (DNNs) to get a full understanding of the surrounding area and to be executed on real driving scenarios with real vehicles equipped with specific hardware, such as memory constrained (DSP or GPU platforms) or multiple optical sensors, which constraints the algorithm's development forcing the designed deep networks to be accurate, with minimum number of operations and low-memory consumption. The main objective of this thesis is, on one hand, the research in the actual limitations of Deep Learning based algorithms that prevent them of being integrated in nowadays' ADAS (Advanced Driver-Assistance Systems) functionalities, and on the other hand, the design and implementation of Deep Learning algorithms able to overcome such constraints to be applied on real autonomous driving scenarios, enabling their integration in low-memory hardware platforms and avoiding sensor redundancy. Deep Learning applications have been widely exploited over the last years but have some weak points that need to be faced and overcame in order to fully integrate Deep Learning into the development process of big manufacturers or automotive companies, like the time needed to design, train and validate and optimal network for a specific application or the vast knowledge from the required experts to tune hyperparameters of predefined networks in order to make them executable in the target platform and to obtain the biggest advantage of the hardware resources. During this thesis, we have addressed these topics and focused on the implementations of breakthroughs that would help in the industrial integration of Deep Learning based applications in the automobile industry. To this end, in the first part of this Thesis we focus on new approaches for vehicle pose estimation using only RGB planar images, avoiding the usage of other sensors that would lead to more costly and hard-to-manage systems. Furthermore, we investigate on the actual limitation of other hardware platforms different that the widely used GPUs (Graphical Processing Unit), such as FPGAs (Field-programmable Gate Array), to host a self-implemented algorithm that understands the surrounding scene of the ego-vehicle specifically designed to run fast and accurate, optimizing the FPGA resources. Finally, once we have done a deep research about the current constraints in the integration of Deep Learning based algorithms in the industry, we propose a new method to overcome them. We go deeper in the design of light DNN (Deep Neural Network) architectures and propose a framework for designing automatically DNNs to run on hardware constrained platforms. We test our development on commercial System on Chips (SoCs) such as Texas Instruments TDA2x SoC family, which are very typical in the automobile industry, obtaining remarkable results in image classification tasks in terms of accuracy, inference time and search time.

Keywords: 3D Pose estimation, deep learning, neural architecture search, depth estimation, adaptive network design, embedded platform.

Resumen

Esta tesis tiene como principal objetivo proporcionar contribuciones teóricas y prácticas sobre el desarrollo de algoritmos de aprendizaje profundo para aplicaciones de conducción autónoma. La investigación está motivada por la necesidad de redes neuronales profundas (DNN) para obtener una comprensión completa del entorno por parte del vehículo autónomo y por la necesidad de dichos algoritmos de ser ejecutados en escenarios de conducción reales con vehículos reales equipados con hardware específico, con memoria limitada (plataformas DSP o GPU) o múltiples sensores ópticos, lo que limita el desarrollo del algoritmo obligando a las redes profundas diseñadas a ser igualmente precisas, con un número mínimo de operaciones y bajo consumo de memoria. El objetivo principal de esta tesis es, por un lado, investigar las limitaciones reales de los algoritmos basados en aprendizaje profundo que impiden su integración en las funcionalidades ADAS (Autonomous Driving System) actuales, y por otro, el diseño e implementación de algoritmos de aprendizaje profundo capaces de superar tales limitaciones para ser aplicados en escenarios reales de conducción autónoma, integrables en plataformas de baja memoria y potencia y evitar la redundancia de sensores. Las aplicaciones de aprendizaje profundo se han expandido ampliamente en los últimos años, pero tienen algunos puntos débiles que deben enfrentarse y superarse para integrar completamente el aprendizaje profundo en el proceso de desarrollo de los grandes fabricantes o empresas de automoción, como el tiempo necesario para diseñar, entrenar y validar una red óptima para una aplicación específica o el alto conocimiento de los expertos requeridos para tunear hiperparámetros de redes predefinidas con el fin de hacerlas ejecutables en la plataforma de destino, obteniendo así el máximo beneficio de los recursos de hardware. Con este fin, en la primera parte de esta tesis nos enfocamos en nuevos diseños de algoritmos para la estimación de la pose 3D de vehículos utilizando solo imágenes planas RGB, evitando el uso de otros sensores que conducirían a sistemas más costosos y difíciles de gestionar. Más adelante nos centramos no sólo en la limitación real que supone el uso de múltiples sensores sino que investigamos la limitación real del uso de otras plataformas hardware diferentes a las GPU ampliamente utilizadas, como las FPGA, para albergar un algoritmo autoimplementado capaz de analizar y entender el entorno del vehículo diseñado específicamente para funcionar rápido y preciso, optimizando los recursos de la FPGA. Finalmente, tras haber investigado sobre las limitaciones actuales que previenen el uso de aplicaciones basadas en aprendizaje profundo por parte de la industria, proponemos una metodología para superar dichas limitaciones. Exploramos más en profundidad el diseño de arquitecturas DNN ligeras y proponemos un *framework* para diseñar DNN automáticamente basado en los útimos avances DNAS (Differentiable Neural Architecture Search) para que se ejecuten en plataformas con restricciones de hardware. Probamos nuestro desarrollo en System on Chips (SoCs) comerciales como la familia Texas Instruments TDA2x SoC, muy típicos en la industria del automóvil, obteniendo resultados notables en tareas de clasificación de imágenes en cuanto a precisión, tiempo de inferencia y tiempo de búsqueda.

Keywords: Estimación de pose 3D, aprendizaje profundo, búsqueda de arquitectura de red, estimación de la profundidad, diseño de red adaptativo, sistemas embebidos.

Acknowledgements

I would like to thank many people that have contributed in one way or another in this work. Firstly to my parents and brother, without their unconditional support in every choice I make and work I do, it'd have been much harder for me to accomplish this goal of finishing my PhD. Even in the distance, they have always been supportive and loving, giving good advices in the right moments, when my strengths were failing and I was thinking of quitting. For all this and more, a big part of this work is thanks to them.

Secondly to my partner Sara, who has stood by me in every step of the way and has lived in first person the struggles, the stress and the insomniac I have experienced the years this research work has lasted. For this and the new challenges to come and for standing by me and supporting me every day I thank you.

Thirdly to my friends. The ones in Madrid, that I have known for decades and have always been there for me, their support and long conversations about my problems in the thesis were healing and truly important. To the ones in Germany, people I have lived the best experiences with and in short time have become really close and important people in my life. And for sure, to the ones in Barcelona, these I have met during this thesis and have seen me daily struggling and striving. Thanks for your support.

Lastly to my supervisors, Antonio and Francesc. Thanks for the big help and for accepting me as a candidate in the first place. I might not have been the best student in the world, the industrial thesis has not always been easy to deal with for me, but I assure you I did my best, even if sometimes it was not visible.

This work was supported by the Catalan Government inside the program "Doctorats Industrials" and by the company FICOSA ADAS S.L.U. J. García López is supported by the industrial doctorate of the AGAUR.

Contents

Figures

Tables

Nomenclature

AD	Autonomous Driving
AP	Average Precision
BN	Batch Normalization
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNAS	Differentiable Neural Architecture Search
DNN	Deep Neural Network
DSP	Digital Signal Processor
FCN	Fully Convolutional Network
FLN	Feature Learning Network
FLOP	Floating Point Operation
FOV	Field of View
FPGA	Field-programmable Gate Array
GPU	Graphical Processing Unit
HLS	High Level Synthesis
LLVM-IR	Low Level Virtual Machine - Intermediate Representation
MAC	Multiple Accumulate operations
NAS	Neural Architecture Search
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SoC	System on Chip
VFE	Voxel Feature Encoding

1 Introduction

1.1 Introduction

Over the last two decades, the dream of autonomous vehicles has passed from an idea to almost a reality thanks mainly to the advances achieved on new hardware platforms that allow heavy computations in a short time. Although technologies such as machine learning or even Deep Learning have been deeply studied, it was not until these powerful hardware platforms appeared in the market, that big companies started considering the possibility of integrating artificial intelligence based functionalities in their vehicles.

A significant step forward for Deep Learning took place in 1999-2000, when computers started becoming faster at processing data and GPUs (graphics processing units) firstly appeared. One of the first companies that launched such devices to render 3D objects instead of only 2D was NVIDIA with their card GeForce 256. They defined the term GPU as "a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second" [26].

Ten years after GPUs were launched, the big automobile company AUDI started incorporating NVIDIA GPUs in their infotainment to improve their navigation and entertainment systems. Big manufacturers discovered then the benefits of these powerful platforms in improving the visualization and the user experience in their vehicles, making them more attractive for the customers, but still refused to incorporate Deep Learning based functionalities. However, simultaneously, the Advanced Driver-Assistance Systems (ADAS) functionalities were significantly boosted over these years. Traditional computer vision techniques and the beginning of sensor fusion led to new assistance functions such as blind-spot detection or parking assist to help the parking maneuver through a rear camera and ultrasounds sensor mounted on several points of the vehicle. These new ADAS functions not only help the driver but also reduce the number of accidents as shown in Figure 1.1, what have motivated the creation of new regulations by the Governments to standardize these new ADAS functionalities or even make them mandatory for all new models, like the Automatic Emergency Brake (AEB) system, which has been proposed to be mandatory in all vehicle from 2022 on [27].

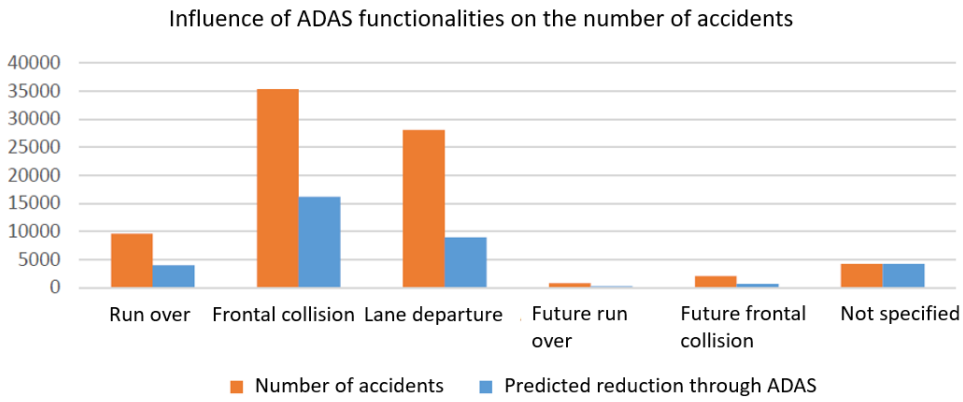


Figure 1.1: Impact of the introduction of new ADAS functionalities in the reduction of car accidents [1].

In 2014 the company Tesla introduced the Autopilot function on their Model S cars, what meant a step-forward for Deep Learning technology, since for the "first" time, it was not only being widely used and studied in a research environment but also in a big manufacturer. In less than 15 years, the automobile industry was completely re-designed and new modern functionalities were introduced step-by-step in not only luxury models but also more affordable vehicles, impacting directly the society and their driving routines. Nowadays it is not hard to find a vehicle with an automatic-braking system, a drowsiness alert for the driver or an auto-parking assistant.

Moreover, not only the automobile companies have been deeply impacted by these technological breakthroughs, but also the camera suppliers and sensor vendors have been forced to develop faster, cheaper and more accurate devices to be mounted on the vehicles and to provide the data to these new functionalities. Figure 1.2 shows the increment of the automotive sensor market from 2012 to what is expected in 2022. As specified in [2], advanced proximity and positioning sensors help to remotely determine the location of an object, and this segment is expected to grow at a CAGR (Compound annual growth rate) of over 11.0 % from 2015 to 2022. These group of products include radar, infrared, ultrasonic, and laser devices, and are finding application in the detection of objects around vehicles in addition to occupancy sensing.

1.2 Motivation

The usage of Deep Learning is becoming a daily reality for more and more companies for which it has meant a true transformation of their processes, reducing the expenses and increasing the benefits. That is due to the substitution of their inspection and supervision activities by a piece of code that can do it autonomously and accurately. Although there is a huge potential behind Deep Learning , its application on certain platforms or to specific problems is sometimes hard to handle. One of the main bottlenecks of the application of this technology is the need of several costly sensors to perform activities such as 3D pose predictions. Although there are several types of optical sensors, like LIDAR, that can provide a

straight measurement of the surrounding objects depth, these are still too big and expensive to be used in series products and mounted on real vehicles. To overcome this limitation, a large amount of published research works have focused on extracting 3D information from only images taken from single or multi-cameras. A precise and trustworthy surrounding scene understanding together with a calculation of the 3D shapes taking only planar images as input brings a big advantage to real applications, since sensor redundancy could be avoided or minimized leading therefore to a more profitable system.

Together with the above mentioned need of optimizing the types and number of required sensors for particular applications, now there is a big concern on how to get the best profit of this technology and how to make Deep Learning based functions integrable on hardware platforms with limited resources, such FPGAs or DSPs.

Typically classification and segmentation problems have been addressed using some of the already available neural architectures (like AlexNet [28], VGG [29] or GoogLenet [30]) and tuning their hyperparameters to fulfill the expectation of the particular application they are needed for. Although this technique can provide good results and is still being widely used, it is not very optimal since these architectures might not be the best choice for the target application and in the end, there are misused resources. Together with this, as mentioned before, industrial applications require hardware platforms that have usually constrained resources, in terms of available memory and computational power due to the limited space of their designs and low budget dedicated. If they have to manufacture in series, the companies shall target low-priced platforms in order to avoid impacting the global price of the product.

This limitation or constraint has typically prevented the integration of Deep Learning based functions on series products manufactured by such big companies, since all of these applications normally require specific hardware requirements to be executed. Several investigations have focused on overcoming this limitation to ease the usage of deep neural networks, like [31] or [32]. In order to do it, several new design strategies are being proposed to obtain light and optimal neural architecture that can be both accurate and light (low memory consuming) [22, 33]. Some of these strategies are based on the hyperparameter pruning, trying to reduce the size of weights or using less hidden layers, such as [34] or [35], while others propose to minimize the required amount of operations in the network, or MACs (mulitpply-accumulate opperations) [18,36,37], since this would make the network lighter and easier to be computed by resource-limited platforms.

Together with the need of making lighter networks, several recent approaches that overcome the difficulty of designing neural networks like [16], or [36] have been published, since this is a difficult task that normally require experienced experts to tune the hyperparameters of pre-defined networks properly or to design them from scratch. For this reason, many research works have been focused on the development of frameworks to automatically design deep neural networks, [38]. These Neural Architecture Search (NAS) methodologies define the search space of all the architecture candidates to perform one specific task as a combination of operations over the input data. Typical approaches, such as [33] or [18], propose a framework that samples network architectures or operation combinations from this search space and train them through a proxy dataset with a reduced number of images. Based on the training results, the next network candidate is selected until an optimal design is found that fulfills some required KPIs (key performance indicators).

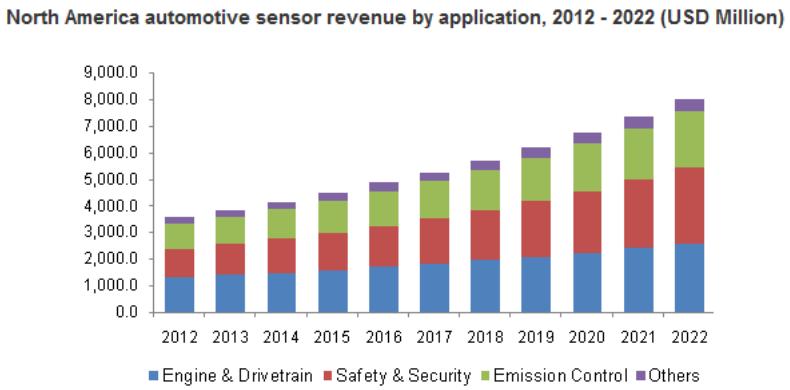


Figure 1.2: Evolution of the automotive sensor market volume in North America, [2]. The Global Automotive Sensor Market size is expected to reach USD 36.76 billion by 2022.

This reward-based methodology is normally combined also with a factor that measures real latency or FLOPs (floating point operations) on the real platform, so that the performance of the candidate's training over the proxy dataset takes into account also these indicators to assess how good the network candidate is. These methods can obtain good results, but have some weak points, like the required search time, which is normally very high due to the sampling process of all combinations of operations over the search space. To overcome this limitation, some research works have proposed a variant of the above mentioned NAS, which defines the search space in the same way, but then approximate the combination of possible operations with a *softmax* function, so that it is differentiable, like [15, 22] or [39]. These are the DNAS approaches (Differentiable Neural Architecture Search).

Through this, the search problem is converted into an optimization problem, in which the weights and architecture parameters that minimize the loss function are searched. Being able to derive the loss function, and therefore the search process, the search speed increases significantly, finding optimal architectures in much less time.

1.3 Thesis Objectives

The specific objectives of this thesis are summarized as follows:

- (i) Develop robust vehicle 3D pose estimation algorithm based on planar images only.

- (ii) Implement a reliable detector of vehicle semantic points of interest to characterize their orientation.

- (iii) Investigate and implement a object detector on LIDAR point clouds and apply it on a real automotive application with real sensor.

- (iv) Investigate on techniques to facilitate the porting of Deep Learning algorithm on resource constrained platforms such as FPGAs.

- (v) Create a framework for automatic neural architecture design.

1.4 Thesis Outline

The contents of this thesis are organized into 3 parts: the first part deals with the contributions on the objectives (i)-(ii). The second part refers to the contributions on the objective (iii). Finally, the third part focuses on neural architecture design for embedded platforms, objectives (iv)-(v). The road map of this thesis is shown in Figure 1.3, which gives the general scheme and illustrates the connections among chapters. Specifically, the contents of Chapters 3-5 are summarized as follows:

1.4.1 Chapter 3: Region based CNNs for 3D pose estimation

This chapter summarizes the research done in the field of 3D pose calculation applied on vehicles. When calculating the 3D pose in single-camera scenarios there is a big problem that has been deeply studied, which is how to calculate the depth of the surrounding objects having only planar images available. Over the years several solutions to this problem have been presented, such as using multi-sensors and combining their measurements or using stereoscopic cameras. These options are effective when calculating the depth from the surrounding environment but include complexity to the system and increase the price. For this reason, real applications try to minimize the number of sensors like LIDAR, which nowadays are expensive and with big dimensions, what make their use unrealistic.

In this chapter several approaches are investigated for characterising the orientation of the surrounding vehicles with reference to the ego-vehicle considering planar images as input data. The following publications summarize the results from the research done in this field:

- J. García-López, A. Agudo, F. Moreno-Noguer: *Vehicle Pose Estimation using G-Net: Multi-Class Localization and Depth Estimation*, CCIA 2018.

- J. García-López, A. Agudo, F. Moreno-Noguer: *Vehicle pose estimation via regression of semantic points of interest*, 11th IEEE International Symposium on Image and Signal Processing and Analysis, 2019, Dubrovnik, Croatia, pp. 6.

1.4.2 Chapter 4: Object detection on 3D point clouds running on FPGAs

In this chapter the results in the investigations done about object detection on 3D point clouds are presented. As mentioned before, single-camera system have some limitation such as predicting the depth of the surrounding objects. To overcome this limitation, some systems use other optical sensors like LiDARs. These sensors are formed by a light source sending light beams, a rotating body proyecting the light beams outside the body and an IMU (Inertial Measurement Unit). LiDARs measure the time between light is sent outside the body of the sensor towards the environment, it reflects against surrounding object and comes back to the LiDAR. Knowing the frequency of the light beam and the difference between the moment the light beam is sent and when it returns, the distance between the surrounding object and

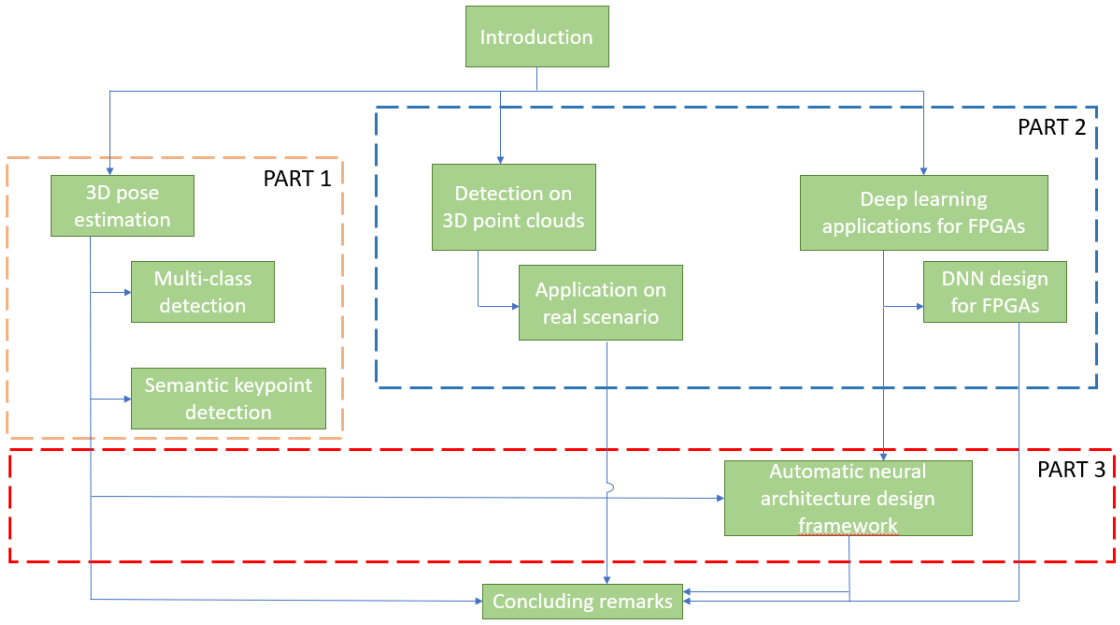


Figure 1.3: Schema of thesis organisation.

the ego-vehicle can be calculated. Sending multiple light beams at the same time that the sensor body rotates around its axis, several light beams returns can be collected at the same time, so that not only the distance but also the shape of the objects can be predicted.

DNNs able to detect objects directly from the point clouds leads to an optimal usage of these kind of sensors, getting as much information as possible with one single sensor. Moreover, being able to integrate this development on embedded hardware platforms such as FPGAs, which are low priced and widely used in the automobile industry would mean a step forward in the usability of Deep Learning based algorithms by big manufacturers, which is one of the biggest barriers that Deep Learning faces right now. Field programmable gate arrays (FPGAs) are low-power devices very suitable for embedded systems. Furthermore, an FPGA is able to perform parallel processing and data communications on-chip. Hereby, FPGA are very powerful platforms that can lead up to higher inference speed in DNNs and that, due to their low price, have been widely studied to be used in real automotive applications. Despite these advantages, FPGAs present some constraints that have prevented their use in real scenarios: low memory resources and the lack of trustworthy frameworks that help developers program CNNs and translate them into executable code on FPGAs.

As mentioned above, one of the biggest differences when programming an application that runs on a GPU or on a FPGA is the fact that available memory to handle the multiple meta-parameters and weigths during CNNs training is more constrained in a FPGA than in a GPU or CPU. In this chapter, it will be presented a network designed adapted to be executed on FPGAs, getting the best profit of the advantages such platform can offer to DNN applications. The design considerations, results obtained and special development done in order to adapt such DNNs to FPGA architecture will also be shown in this Chapter. Results on this research are presented in the following publication:

- J. García-López, A. Agudo, F. Moreno-Noguer: *3D vehicle detection on an FPGA from LiDAR point clouds*, 2nd International Conference on Watermarking and Image Processing, 2019, Marseille, France.

1.4.3 Chapter 5: E-DNAS: Differentiable Neural Architecture Search for Embedded Systems

Deep Learning technology has reached a maturity level enough to give solutions to many problems from different fields, and is leading to a real revolution in terms of processes optimization and urban mobility, among others. Despite the great breakthrough Deep Learning means, it has some limitations such as: the long time needed to develop customer-oriented DNNs and the required expert knowledge for DNN design and hyperparameter tuning. These constraints are limiting the incursion of Deep Learning in the daily life of many users. For this reason, several research work have focused lately on developing frameworks for automatic DNN design, so make such technology accessible to everyone and overcome the before mentioned limitations. In this chapter, a new methodology for fast neural architecture search is presented, demonstrating good results in embedded platforms such as DSPs. The results are shown in the following publication:

- J. García-López, A. Agudo, F. Moreno-Noguer: *E-DNAS: Differentiable Neural Architecture Search for Embedded Systems*, International Conference on Pattern Recognition, 2020, Milan, Italy.

Based on the work presented in this Chapter we propose an extension of that research applied on object detection challenges. We introduce the Stacked-NAS method, which is a two-step pipeline that aims to find the best DNN architecture to detect objects on images. The first step is a region proposal network, which looks for object candidates on the image that will be fed to the following step two for their classification. The main contribution of this work is the above mentioned region extraction DNN explained as step one, since the classification step bases on the E-DNAS research, explained at the beginning of this Chapter. The proposed object candidates are searched using a NAS approach, in which the target architecture has a encoder-decoder structure and looks for high feature-density regions to mark them as object candidates. The results are shown in the following publication:

- J. García-López, A. Agudo, F. Moreno-Noguer: *Stacked-NAS for object detection*, Journal on Computer Vision and Image Understanding, 2020 (under review).

2

Background and state of the art

2.1 Typical sensors and platforms in autonomous cars

In this Chapter the main technical concepts that this dissertation bases on will be explained. Firstly, the main typical sensors in autonomous driving will be defined, with special detail on RGB cameras and LiDAR sensor, since these are mostly used in this thesis. Furthermore, a short overview of the Convolutional Neural Networks (CNN) will be exposed, in particular the CNNs that aim to extract 3D pose information from the input data and the considerations that shall be taken into account when implementing a CNN on a memory-constrained platform such as an FPGA. Finally, it is given and introduction on Neural Architecture Search methods followed by technical concepts and operations that will be used and mentioned throughout the thesis.

2.1.1 Types of sensors

As commented in Section 1.1, the automotive sensor market has been directly impacted by the introduction of new ADAS functionalities in the vehicles. The market demands new assistance systems that help the driver make turnover maneuvers, park or that avoid run overs. In order to develop reliable and accurate functionalities that can provide a solution to these needs, several type of sensors mounted on the vehicle are required in order to extract as much information as possible from the surrounding area. In Figure 2.1 is presented a schema with the typical sensors used in autonomous vehicles and their disposition.

In this section, the typical sensors utilized in automobile industry will be defined, explaining their benefits and limitations and how are they nowadays used for the development of new ADAS functionalities, such as blind-spot detection (BSD), automatic parking, or back-over protection (BOP):

- Camera. An optical sensor converts light rays into an electronic signal. The purpose of an optical sensor is to measure a physical quantity of light and, depending on the type of sensor, then translates it into a form that is readable by an integrated measuring device. Optical Sensors are used for contact-less detection, counting or positioning of parts. RGB cameras use a light sensor

Figure 2.1: Sensor map in an autonomous vehicle [3].

to translate the amount of light received through the lens into a matrix of RGB values or image. Classic computer vision techniques can then be applied on that image to perform classification, detection or segmentation tasks and obtain therefore information of the environment. Typical cameras used in automobile industry have an image-quality module that processes the output of the sensor and aims to improve the quality of the obtained image, removing dead pixels, improving the contrast or minimizing the blurriness. Cameras are one of the most used sensors in autonomous driving due to the big amount of information that they can provide, their relatively small size and low price. However, they have some limitations such as the degradation suffered on the obtained images in scenarios adverse weather conditions, the dependence on the light conditions and, as before commented, the need of an image-enhancement step (via hardware or software) after the sensor, so that the image that comes into the computer vision module has a high quality. Being able to enhance the image as required using low hardware resources and it short time is not an easy task and requires experts and time to find the best implementation.

- LiDAR. Light Detection and Ranging (LiDAR) sensors emit pulsed light beams (lasers) at high rate and measure the reflection time obtaining the distance to the collision, known as range. These are very typical also in automobile industry, however they are usually not used in commercial vehicles due to mainly their size and price. Although these light sensors can provide an accurate measurement of the depth of the surrounding objects by measuring the time between light is emited and the reflected light beam is received back, they are normally big sensors (compared to cameras or ultrasounds) that include a high level of complexity in the system. LIDARs are formed by a global positioning system (GPS) that: records the x,y,z location of the scanner, an Inertial Measurement Unit (IMU) that measures the angular orientation of the scanner relative to the ground (pitch, roll, yaw) and a clock that records the time the laser pulse leaves and returns to the scanner. In a LIDAR sensor the x/y/z coordinate of each return is calculated using the location and orientation of the scanner (from the GPS and IMU), the angle of the scan mirror, and the range distance to the object. The collection of returns is then known as a **point cloud**. The Figure 2.2 shows an example of the measurement provided by a LIDAR sensor.

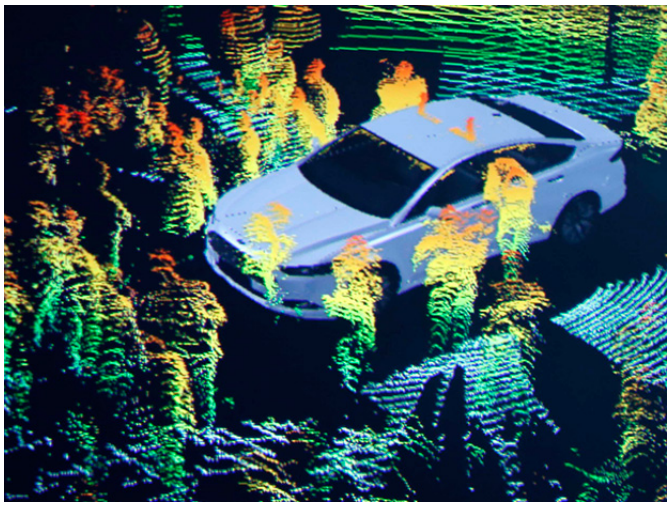


Figure 2.2: Illustration of the measurement of a LIDAR sensor [4].

- Ultrasound. An ultrasonic sensing system operates by transmitting short bursts of sound waves and measuring the time taken for the sound to travel to a target object, be reflected, and return to the receiver. The distance to the object is a function of the travel time and the speed of sound in air, approximately 346 m/s [40]. It works like the before explained LIDAR sensor but emits sound waves instead of light beams. The ultrasonic sensors (USS) are the most used sensors in the automotive industry due to their low price, small size and accurate measurements. Nowadays it is hard to find a vehicle without an USS sensor to help the driver parking emitting a sound when he is too close from other parked vehicle.

2.1.2 Types of platforms

In this section of the thesis it will be defined the different platforms that can be used nowadays to run Deep Learning based application on the automobile industry. The usage of these platforms and their impact on the implementation of algorithms is one of the objectives of this dissertation.

- FPGA: Field-Programmable Gate Array are integrated circuits which can be “field” programmed to work as per the intended design. It means it can work as a microprocessor, or as an encryption unit, or graphics card, or even all these three at once. The designs running on FPGAs are generally created using hardware description languages such as VHDeep Learning and Verilog. FPGA are made up of thousands of Configurable Logic Blocks (CLBs) embedded in an several programmable interconnections. These blocks are made of Look-Up Tables (LUTs) and multiplexers primarily which implement complex logic functions. Some FPGAs also contain dedicated hard-silicon blocks for various functions such as RAM memory or DSP (Digital Signal Processing) Blocks or even processor cores inside the FPGA itself so that it can take care of non-critical tasks whereas FPGA can take care of high-speed acceleration and heavy computational operations. FPGAs are then more suited for parallel computation and can support more flexible architecture. Also these platform

have lower power consumption compared to GPUs or CPUs. However, FPGAs have some downsides as well, such as the available memory, which is usually much more restricted than in other platforms, and the usability, which is still hard for the developing of Deep Learning applications.

- GPU: Graphic Processing Units are programmable processors though to render images. They are designed for computing in parallel multiple threads. While CPUs (central processing units) can have some cores, have low latency and are designed for serial processing, GPUs are good for parallel processing and can do thousands of operations at once. A CPU also has a higher clock speed, meaning it can perform an individual calculation faster than a GPU so it is often better equipped to handle basic computing tasks. A central processing unit (CPU) is designed to handle complex tasks, such as time slicing, virtual machine emulation, complex control flows and branching, security, etc. In contrast, graphical processing unites (GPUs) only do one thing well. They handle billions of repetitive low level tasks. Originally designed for the rendering of triangles in 3D graphics, they have thousands of arithmetic logic units (ALUs) compared with traditional CPUs that commonly have only 4 or 8. Many types of scientific algorithms spend most of their time doing just what GPUs are good for: performing billions of repetitive arithmetic operations. Figure 2.3 illustrates the different hardware architecture between CPUs and GPUs. In Deep Learning applications, chips are typically formed by a CPU that pre-processes the data and a GPU that actually performs the computation.

- SoC: System on Chips are integrated circuits which are composed by a processor or chip and a computer or electronic system built onto it. They usually have a central processing unit, inputs and output ports and internal memory, among others. They act like small and portable computers and have opened the door a limitless of products such as mobile phones or tablets. SoCs usually come with an operating system installed in order to ease the usage of its resources. Similar than GPUs are CPUs, SoCs are based on processing units which are instruction-based hardware. Instruction-based hardware is configured via software, whereas FPGAs, for instance, are instead configured by specifying a hardware circuit.

2.2 Object detection

2.2.1 Convolutional Neural Networks

Since the ending of the 20-th Century that the first promising neural network research appeared like LeNet [41] reveling that images have distributed features which can be learned through convolutions, there has been multiple research works that make use of this concept and extend it for the purpose of feature extraction, class localization or even object reconstruction.

LeNet provided a good first approach of what a **convolutional neural network (CNN)** can do and what they are based on. It was not until beginning of 21-th Century with AlexNet [28] that the LeNet concept was extended into a wider idea showing that a neural network could learn more

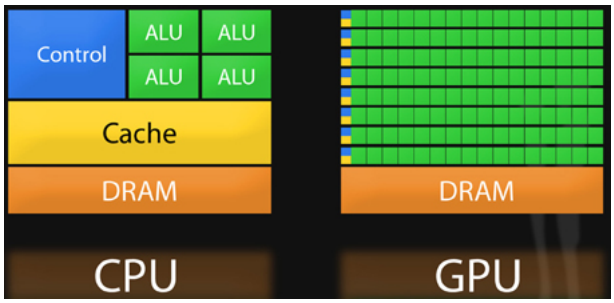


Figure 2.3: Difference between CPU and GPU hardware architecture [5].

complex objects introducing concepts like the ReLu and dropout layers commonly used nowadays in any network implementation. Later on, the appearance of better computers and GPUs triggered the usage of CNNs as they could run faster and therefore be applied to multiple other problems. Over the years, CNNs have been evolving and new architectures have appeared to optimize the needed resources for training and validation improving the obtained results. Over the 21-st century networks like VGG [29], GoogLenet [30] or ResNet [42] have meant a breakthrough in the history of Deep Learning allowing better performance in problems like object detection, better detection rate and optimized computational resources.

Together with the above mentioned networks, there are some CNNs approaches that have become very popular for detecting objects in images due to their good performance and accuracy. These are the region-based CNNs [43] which propose a two-step pipeline for extracting the objects found on planar images. First versions of these approaches proposed a first processing step for grouping parts of the images with similar characteristics such as color intensity or texture [44, 45] followed by a second step with a CNN to predict the position of the bounding boxes around the detected objects and their classes.

These techniques proved to be very accurate but slow. Due to this, a faster version of the RCNN appeared later on that applied the CNN directly on the input image to generate a feature map from which the object proposals are extracted through selective search, [25]. The extracted objects were then fed into a set of fully connected layers to predict bounding box position and object class. This new version achieved good results in shorter time that their predecessor but the selective search step was still too slow and there was no learning taking place there. Next improved version of the region-based CNNs were the FASTER-RCNN, [46] which no longer use selective search but a region proposal network (RPN) applied on the generated feature map after first ConvNet. Both FAST and FASTER-RCNN use a a ROI (Region Of Interest) pooling layer after the extraction of the object proposals and before step two in the pipeline in which, the extracted ROIs with the object proposals to reshape them into a fixed size so that it can be fed into a fully connected layer.

RPN is then formed by a ConvNet, such as VGG-16 [29] or ZF [47], which form a feature map from which ROI proposals around object candidates are calculated.The ROI proposals are then passed to a ROI pooling step that resizes all ROIs to same size so that they can be fed to the final fully connected layers. These layers output, from one side if there is an object or not in the anchor, and from the other side which is the position of the bounding box around the detected object in that anchor. RPM have then a classifier

and a regressor.

These ROIs are extracted by sliding a window or "anchor" around the feature map. The paper suggests to slide 9 anchors (area that network will look at to see if there is an object inside it or not) centered at each pixel in the feature map. The generation of these candidate boxes or anchors take into consideration two parameters: scales and aspect ratios. The boxes need to be at image dimensions, whereas the feature map is reduced depending on the backbone network architecture (VGG or ZF typically). This means, on each pixel we extract 3 anchors with 3 different aspect ratios (1:1, 1:2 and 2:1) multiplied by anchors on 3 scales (1, 1/2 and 1/4).

As mentioned above, once all anchors are extracted, Non-maximum suppression [14] is applied to ignore redundant boxes. Then, ROI pooling is applied on the ROIs to convert them all to the same size so that they can be fed to the FCs to calculate whether there is an object or not (classifier) and the position of the bounding box around the object (regressor). RPM has a loss function that contemplates both mentioned regressor and classifier and during training the error between the predicted bounding box and the ground-truth bounding box shall be minimized.

2.2.2 Image based 3D object detection

Being able to calculate the 3D pose of the surrounding objects using only planar camera images has been an important field of study over the last years, since it avoids the need of multiples sensors to predict the depth of these objects. This leads to easier-to-manage and less costly systems. Having the information from the shapes of the surrounding areas in an autonomous driving application helps to the calculation of driving trajectories and obstacle avoidance.

There has been important works showing good results in human 3D pose estimation like [48–51] or [52].

Research works like [53] have showed very good results when calculating the 3D human pose from joint localization. This is achieved by passing the input image through several hourglass phases, to generate heatmaps to capture features at various scales. The motivation of doing so is the need of spatial information for calculating the pose. An understanding of the whole body is crucial to predict the body pose.

The architecture of hourglass architecture is formed by a Convolutional and max pooling layers used to process features down to a very low resolution. After reaching the lowest resolution, the network begins the upsampling to the original resolution and combination of features across scales. Hourglass networks are symmetric, so for every layer present on the way down there is a corresponding layer going up. After reaching the output resolution of the network, two consecutive rounds of 1×1 convolutions are applied to produce the final network predictions. The output of the network will be the mentioned heatmaps where the human joint will be for each one predicted with pixel accuracy [53].

The hourglass is a simple, minimal design that has the capacity to capture multiple features and bring them together to output pixel-wise predictions [53]. The aim of using this network architecture is to obtain full context information important for pose extraction, meaning that, not only the exact prediction of the position of the keypoints is important, but also the pose estimation requires a full understanding of the vehicle.

These approaches tend to use the texture information provided by the input images to predict the 3D bounding boxes from 2D images. However, the accuracy of image-based 3D detection approaches are bounded by the accuracy of the depth estimation. One of the reasons that motivate the usage of LIDAR point clouds when solving the issue of the 3D bounding box calculation is this one, since these optical sensor already provide depth measurement and no error is included in the detection pipeline when predicting the depth.

2.2.3 CNNs on FPGAs

The emergence of field-programmable gate arrays (FPGAs) in image processing and Deep Learning is increasing nowadays due to the benefits of such hardware for conducting faster mathematical computations and processing operations, but mostly because of the appearance of new frameworks that allowed developers to port their work to an FPGA in a more straightforward way. Several studies like [54] proof the advantages of considering FPGAs as an option for image processing and Deep Learning applications.

In order to overcome the challenge of extracting 3D information from 2D data, several researchers have presented works and methodologies that have proven remarkable results in human pose estimation or face expression recognition, like [51] or [55].

In the field of vehicle pose estimation, other image-based approaches suggested a system with two cameras separated a known distance for feature matching, 2D detection followed by a 2D-3D matching phase for calculating the 3D position of the vehicles, such as in [21], or even a system in which the ground plane equation is known in advance together with a 2D vehicle detection network (e.g., [56]) to predict as last step the 3D bounding box around the detected vehicles in the image.

To avoid these mentioned constrains, the usage of point clouds from optical sensor such us LIDARs is a good option because these provide already the required 3D information. However the accuracy of these sensors and the difficulty to manage 3D point clouds compared to 2D images have lead to these sensors not being widely used in 3D pose estimation problems.

LIDAR sensors in autonomous driving work normally by reflecting light beams from a light source in an internal mirror that outputs the beam outside the sensor toward the object to localize. These sensors rotate around themselves so that they provide depth information of a 360º surrounding area. The rotation frequency together with the number of light beams emitted each cycle are key to obtain accurate environment information to be used for training of a neural network.In addition, LIDAR is not subjected to environmental illumination. Normally these sensor were not used in commercial applications for autonomous driving due to their high price and difficulty for data synchronization. However, lately precise 32 or 64 channel LIDAR sensors have come out in the market with reasonable price and size that make it more reasonable to be integrated in a commercial vehicle.

As stated in works such as [13], although graphics processing units (GPUs) are more suitable for parallel processing, they do need a high power consumption, which could make them a bottle-neck for the integration of Deep Learning algorithms into vehicles, as they have limited power supply. In this scenario, FPGA are a low-power consumption option more suitable for embedded applications as they can be programmed as a customized integrated circuit that is able to perform massive parallel processing and data communications on-chip. We believe this is a enough motivation for pursuing a breakthrough in

the field of Deep Learning applications on FPGA, since the usage of this platform in commercial vehicles is widely extended already (e.g., for image data-stream conversion) and the appearance of frameworks for porting networks to run on FPGA platforms is boosting up their usage in autonomous vehicles against GPUs.

2.3 Neural Architecture Search (NAS)

Deep Learning is a power-full concept that can mean a revolution in several areas, but it has some important constraints or limitations that need to be overcome in order to obtain its full potential. One of the typical constraints is the hardware resources (e.g. computational power, memory) required to execute some Deep Learning applications, what some times limit its use. Together with this, another known constraint of this technology is the usually manual design of the neural networks, what is typically performed by experienced profiles that tune hyperparameters of already known architectures in order to apply or adapt them to a specific functionality.

In this direction, some works have focused on reducing the network size and optimizing its hyperparameter by pruning weights from DNNs, like [34] or [35]. Although these approaches could be effective for some applications, manually selecting the redundant weights and using unstructured sparse filters does not necessarily mean a real advantage in real platforms. Based on a similar idea, some recently published papers propose a method to design networks that can evolve during the design process based on some feedback in order to obtain the optimal number and type of layers for a specific application. These so called NAS (neural architecture search) approaches have recently achieved better performance than hand-crafted models by automating the architecture design. Based on this NAS idea, works like [57] propose a framework for automatic network design called LEAF, which demonstrates that architecture optimization provides a significant boost over traditional manual hyperparameter optimization technique, and that networks can be minimized and optimized at the same time with little drop in performance. This framework has three main levels or layers: algorithm, system and problem-domain. The first (algorithm layer) is where the evolving of the network architecture and hyperparameters takes place, that means, network candidates are generated. On the second level (system layer) the framework provides a parallel training using a different platform to speed this process up (such as Amazon AWS, Microsoft Azure , or Google Cloud) and evaluates the performance of the candidates. The last level (problem-domain layer) of the framework LEAF receives feedback from the other two levels: the network designer or algorithm layer and the network candidate "checker", or system layer. Problem-domain layer solves problems like hyperparameter tuning, architecture search, and complexity minimization.

Further on, [37] proposed a framework that uses direct metrics when optimizing a pretrained network to meet a pre-defined resource budget. In this research, empirical measurements are taken directly from the real platform to evaluate the network candidates generated, so that no previous knowledge from the target platform is needed. This research shows good results and is able to find an optimized network that fulfills memory budget by incorporating direct metrics such us *latency* to the optimization loop.

Some NAS approaches like [33, 58] or [18] apply the concept of reinforcement learning for finding the best neural architecture. These approaches propose a framework with a recurrent neural network

(RNN) as a controller from which child architectures will be extracted and trained to get their accuracy. Based on this accuracy, the reward signal for the controller will be calculated and fed back to it, so that on next iteration the controller will give higher probabilities to architectures that receive higher accuracies (controller learns to improve its search over time), [18, 33]. Although this reward-based approaches showed really good results in providing efficient network architectures to be executed on mobile platforms, it still had one big disadvantage, which is the extremely long training time needed (e.g., [18] requires 2000 GPU days in the ImageNet or CIFAR-10 dataset or approach proposed in [59] takes 3150 GPU days). This is due to the multiple network candidate to be taken from the big search-space. Since each candidate shall be extracted, trained through a batch of training and validation data, executed on the platform to obtain the metrics (such as energy or latency) and based on this, feed the controller back with the proportional reward signal, this makes these methodologies to be very time consuming.

Recently, a newer and faster version of the NAS has appeared, which achieves the target neural network design quicker by using gradient-based optimizations. The differentiable neural architecture search (DNAS) method were first published in works like DARTS [38], in which it is proposed to convert the design process into an optimization function that can be solved through gradient descent. This approach reduces significantly the search time compared to traditional NAS.

Although methods like DARTS have given good results in terms of accuracy and searching time compared to NAS, they still face some weak points, such as the still relatively long time needed for the architecture finding. Together with this, DNAS approaches such as DARTS [38] have proven not to be practical to be used in large datasets. As stated in [60], the instability of methods like DARTS can be summarized in two causes:

- Multicollinearity of similar operations leads to unpredictable change of the network parameters during search.

- Suboptimal architectures are some time obtained because of the difference between the proxy search stage (search step in which the architecture candidate is executed through a batch of the training/validation data to obtain metrics) and the final training.

For this reason, several researchers have focused lately to overcome these limitations of the DARTS. Recent works like [60] proposes a two-step idea: firstly the best operation group for the given data is found to find the deep of the candidate and secondly the best operation in the activated group is learned. Once the best group of operations and the best operation of that group is selected, the redundant operations are pruned from the original data so that an optimized network architecture is found for the given dataset.

In recent research works it has been proved that a good way to reduce the number of operations on a DNN and therefore make it lighter, is to calculate the weighted sum of the convolution kernels rather than the output features. That is the concept of "additivity" of the convolutions proposed by [61]. This allows to conduct a convolution operation on the input data only once to get a single feature map.

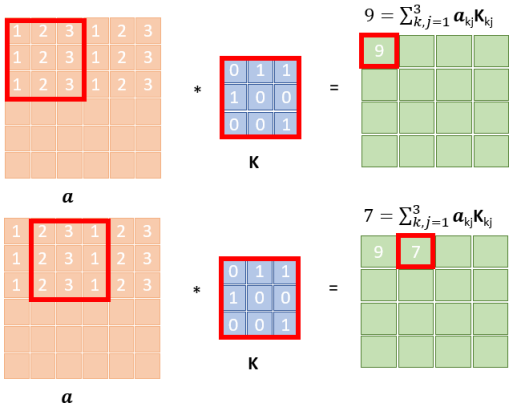


Figure 2.4: Illustration of convolutional operation of a kernel or filter 3×3 (**K**) over an input image 6×6 (**a**).

This has a direct impact on the search strategy, since the search space is encoded on convolutional kernel and "shrink" multiple convolutional kernel candidates into one, what reduces the needed memory for storing intermediate features and the resource budget for conducting convolution operations. Based on this idea, this method proposes a modification in the typical gradient-based NAS, reformulating the typical relaxation of the categorical space by a variant of the softmax that takes into consideration the multiple candidate meta kernels. Through this, a differentiable objective function is obtain and gradient descent can be applied to obtain the weights and the kernels of the target architectures.

2.4 Background

In this section, some necessary definition, mathematical tools and properties are introduced, which will be used in this thesis.

2.4.1 Convolution operation

In a Deep Neural Network, convolution is a type of layer that aims to extract features from an input image.

The convolution operation done on this layer preserves the relationship between pixels by learning image features using small squares of input data. This mathematical operation takes two inputs such as image matrix and a filter or kernel and outputs a feature map.

Some important parameters of the convolution operation are the following:

- Stride: Number of pixels shifts over the input matrix. When the stride is 1 the filter moves one pixel at a time. Figure 2.4 has a stride of 1 and Figure 2.5 has a stride of 3. The stride impacts on the resolution of the output feature map.

- Padding: When the filter does not perfectly fit in the image, the input picture shall be padded with zeros (zero-padding) so that it fits.

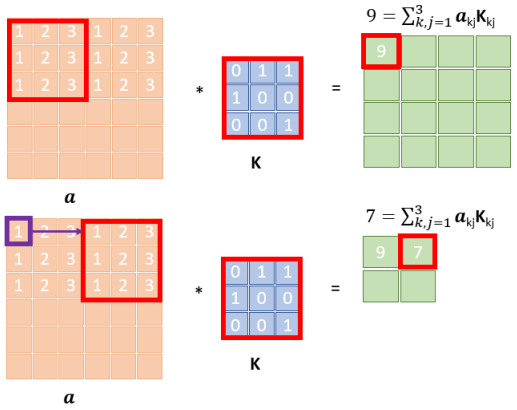


Figure 2.5: Illustration of convolutional operation of a kernel or filter (**K**) over an input image 6×6 (**a**) with a stride of 3 and padding of zero.

Separable convolutional layer

The spatial separable convolution consists in splitting the convolution operation with kernel K into two convolutions with smaller kernels K_1 and K_2. By doing this, the number of operations (multiplications) done in total is reduced, what leads to a faster and lighter network.

Depthwise separable convolutional layer

In the case the convolution involves multiples channel input data (e.g., 3 channels), the original convolution can be divided into two steps, as shown in Figure 2.6. Assuming input image has a resolution of I_W width, I_H height and I_D depth and the kernel has a K_W width, K_H height and K_D depth:

- Firstly a 2D convolution is applied to each input channel and their outputs are concatenated $(I_W, I_H, i)(K_W, K_H, 1) \| i = 1, .., I_D$.

- Secondly a convolution with a kernel of a size $1 \times 1 \times K_D$ is applied to the output of the first step.

This kind of convolutional operations aims to reduce the number of multiplications, obtaining lighter and faster designs.

Dilation convolutional operation

A normal convolution can be generalized by the following formula:

$$(\mathbf{I}_l \mathbf{K})(p) = \sum_{i+lt=p} F(\mathbf{i}) K((\mathbf{t})), \tag{2.1}$$

where l is the dilation factor. $l = 1$ would lead to a regular convolution. Increasing the value of l we would get a bigger receptive field of the convolutional kernel.

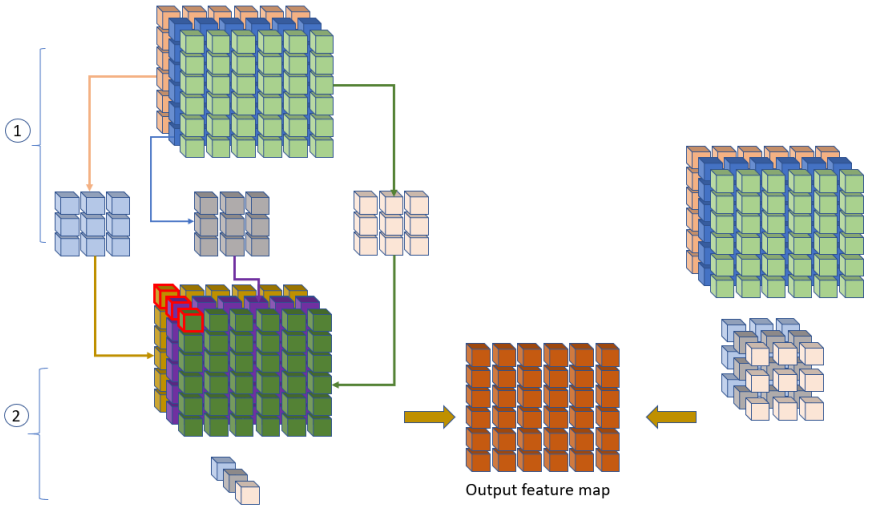


Figure 2.6: Schema comparing a normal convolution operation (right side) and a depthwise-separable convolution (left side). Both end up calculating the same output feature map but with the depthwise convolution the number of operations is less.

This type of convolution was firstly proposed in [62] aiming to aggregate multi-scale contextual information without losing resolution. How these operations work can be seen in Figure 2.7.

2.4.2 Activation functions

The activation or transfer function can be defined as the function that decides whether a neuron shall be fired, that means, decides which information of the present layer shall be passed to the next layer. Considering how the transmission of information works between layer, each step there is an output Y calculated as the sum of the weights multiplied by each input plus a bias term: $Y = \sum(W_i * I_i) + bias$. The activation function is applied to Y.

There are several types of activation functions, such as:

- Rectified Linear Unit: This function consists of the following formulation.

$$ReLu(Y) = \begin{cases} 0 & Y < 0 \\ Y & Y >= 0 \end{cases} . \tag{2.2}$$

  This kind of activation function is widely used although it has one downside. For negative output values, the ReLU gives zero, therefore the gradient will be zero and the gradients will not be updated in the gradient descent. This is normally compensated by using a *leaky ReLU* in which the horizontal line is substituted by a non-horizontal line with small slope.

- Softmax: This function attends to the following formulation.

$$f(Yi) = \frac{e^{Y_i}}{\sum e^{Y_i}}. \tag{2.3}$$

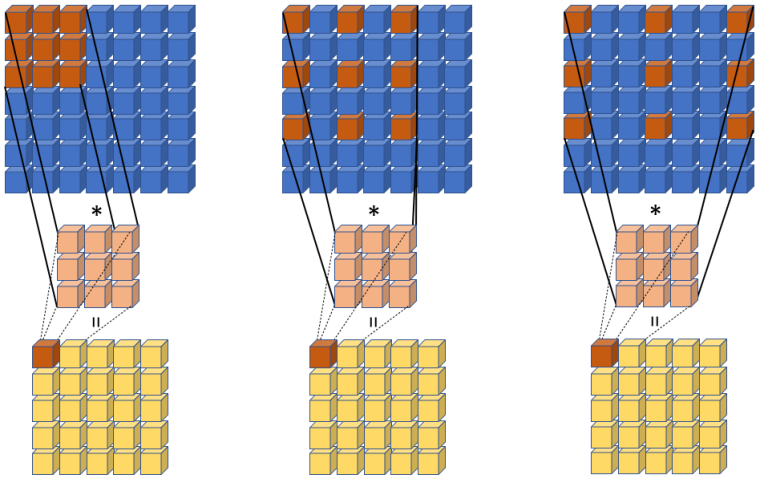


Figure 2.7: Example of dilated convolutions with three different dilation factors: $l = 1$ on the left, typical convolution, $l = 2$ in the middle and $l = 3$ on the right side.

This activation function is also widely used since it transforms the output of each layer into probabilities between zero and one. It is typically used and last layer in classification DNNs to predict a probability for each class.

2.4.3 Learning rate

The learning rate defines how quickly a network updates its parameters. Low learning rate slows down the learning process but converges smoothly.

Larger learning rate speeds up the learning but may not converge.

$$w_i^+ = w_i + l_r * \frac{\partial Loss}{\partial w_i}. \tag{2.4}$$

Equation (2.4) shows how the learning rate l_r affects the weight w_i update during training. Typical learning rate selecting follows the formula $l_r = l_{r_0} * \mathrm{e}^{-Kt}$, named *exponentially decaying learning rate*.

2.4.4 Pooling layer

Another type of layer that can be present in a DNN is the pooling layer. These aim to reduce the dimensionality of each map (downsampling) but retaining important information. There are three type of pooling layers depending on the criteria for the downsampling:

- Max. Pooling (Figure 2.8).

- Average Pooling.

- Sum. Pooling.

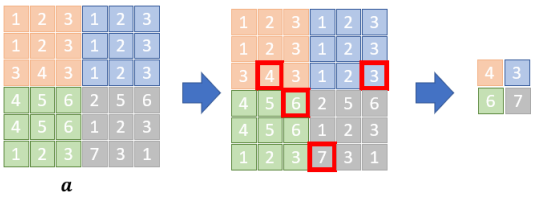


Figure 2.8: Illustration of max. pooling operation with 3×3 filters and stride of 3.

3

Region-based CNNs for 3D pose estimation

This chapter proposes some efficient vehicle 3D pose estimation algorithms using only 2D planar images. Firstly, the results on multi-class detection with parallel depth detection will be presented. As mentioned above, the 3D estimation problem has been widely studied for humans and several researchers have extended their work for vehicles. The approach when applying such algorithms for pedestrians and vehicles changes, due mainly to the different nature itself of both classes. While people are articulated and when constraining the human pose problem, the relative position of each joint shall be taken into account, vehicles are rigid bodies which move as whole with no deformations what means, the relative distance between vehicle parts is constant throughout the sequence of frames. This shall be taken into account when predicting vehicle orientation.

3.1 Introduction

As commented in Section 1.1, the automobile industry is introducing more and more ADAS functionalities that aim to improve the user experience and increase the safety while driving. Several functions that nowadays seem standard in a vehicle and come "by-default" on any model were unthinkable 10-15 years ago. Although these new ADAS applications are making vehicles more and more autonomous, allowing cars to change driving lane or park without driver's intervention, we are still far away from intelligent cars that can take us from point A to point B completely autonomously, this is known as level 5 ADAS level of autonomy (see Figure 3.1). One of the main reasons why this is still unlikely, is the uncertainty of the surrounding objects movement in a typical usecase inside a city. Predicting how pedestrian, bikers or other vehicles will be moving at instant-level so that collisions can be avoided is very difficult. However, through the usage of sensors mounted on the vehicle and on the road it has been achieved to control these agents' movement in a controlled environment, allowing vehicles to drive "by themselves" within this environment.

An example of this is the automatic cruise control functionality, which maintains a certain speed on the ego-vehicle in a controlled environment such as a high-way, in which there are no pedestrians expected and vehicles can only be moving in one direction.

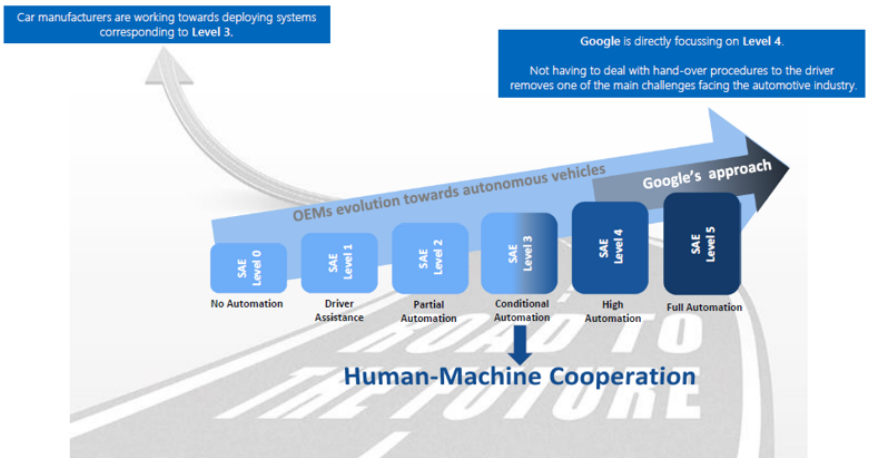


Figure 3.1: Typical levels SAE (Society of Automotive Engineers) levels to standardize the autonomy level of a vehicle. Zero level is considered as no autonomy at all and level 5 is fully autonomous, without requiring any assistance or attention from the driver. An example of level 0 is the cruise control to maintain the vehicle's speed, driver sets the speed, car only maintains it. Level 1 functionality is the adaptive cruise control (ACC), in which the driver sets the speeds and vehicle acts on gas an pedal to maintain it and avoids crashes. Example of Level 2 is the Tesla autopilot, in which vehicle can act on speed and steering wheel on certain conditions. Level 3 functionality is the traffic jam control from AUDI, in which vehicle moves completely alone during a traffic jam, monitoring the whole environment. Examples for level 4 and 5 do not exist yet. Both mean fully autonomy but the difference is that in level 5 the vehicle does not even have steering wheel or pedals, there is no chance for the drive to intervene.

All these conditions constraint the scenario and allow this functionality to actively act on the gas and break pedal.

This and the rest of ADAS functions are based on the data collected by several sensors mounted on the vehicle to obtain the required information about the surrounding objects. In the case of the automatic cruise control function, it uses several ultrasound sensors disposed on the front of the vehicle together with different electromagnetic sensors mounted on the wheels to get their speed.

As explained in Section 1.1, one of the most used sensors in autonomous driving are the RGB cameras, due to mainly their reduced price and small size. Together with this, RGB cameras are able to extract a lot of information from the environment. With the proper software implementation, a single frame can be classified and objects within can be detected and segmented, [28, 32] or [63]. Moreover, the handling of RGB frames is easier and less memory consuming compared to LIDAR point-clouds. This is the main reason why it is more profitable to develop software that processes 2D data from the cameras to extract 3D information (through tracking and feature matching [64], e.g., than using LIDAR sensors that would provide the depth of the surrounding objects directly.

In this section we present two self-designed methods for calculating the 3D pose from surrounding vehicles using planar images from RGB cameras: first one bases on the simultaneous detection of vehicles parts to infer its pose, and second ones detects vehicle semantic points with pixel-wise precision to reconstruct vehicle's shape and orientation.

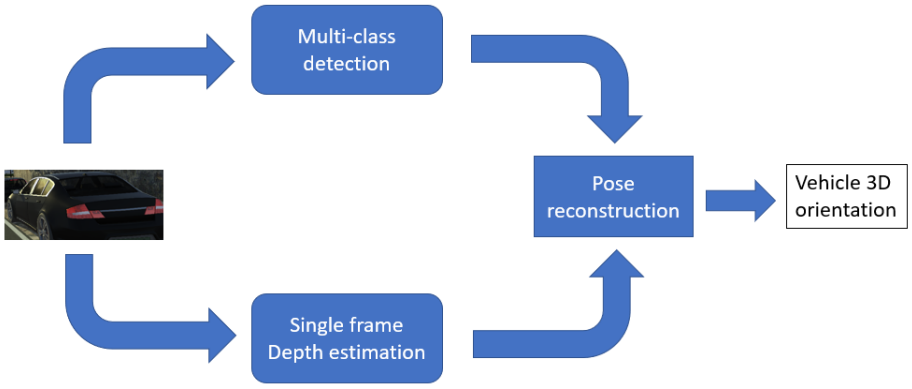


Figure 3.2: High-level Schema of algorithm design for vehicle pose prediction proposed in Chapter 3 of this Thesis.

As above commented, in sub-section 3.2 we present a novel technique for vehicle pose estimation through multi-class detection at frame-level. We propose a single-frame depth estimator using CNNs and combine it with an object detector based on region proposal network [46] to estimate the 3D pose of surrounding vehicles. It has been tested using KITTI Detection benchmark and vKITTI dataset, [20] and [7] respectively, showing accurate results when predicting the vehicle pose of the surrounding cars on an unspecific usecase, overcoming state-of-the-art methods.

Moreover, in sub-section 3.3 it we propose a generalized method based on vehicle keypoint extraction with a similar purpose, vehicle 3D pose extraction, tested on the same benchmark but in a general scenario. The main contributions of this chapter are:

- In sub-section 3.2 we implement a single-frame depth estimator based on a CNN with multiple steps that is able to accurately draw a depth map from the input RGB image in camera coordinate system (CCS) based on a first general scene estimation network followed by a refinement CNN using local information. Together with this, we implement a multi-class region-based proposal network from a FASTER-RCNN [46] that aims to accurately detect vehicles with different level of occlusion on the input RGB image together with their wheels and other objects that would characterize their shape. With the same network we achieve to detect several parts of the vehicle, optimizing the training and inference steps. The combination of this detection plus the before-mentioned extracted depth map will be used in the final step of the proposed method to predict the 3D pose of the vehicles in the image.

- In sub-section 3.3 we propose a new method for vehicle 3D pose prediction based on the accurate detection of vehicle's key-points that would characterize their shape, such us front left light or rear license plate. Following this, up to 20 semantic points are extracted for each detected vehicle. The proposed method implements a self-developed DNN based on Stacked-Hourglass architecture [53], very typical in human pose extraction problem. The method presented in this section consists in a two-step pipeline with a first detection of the keypoint together with a prediction of the occlusion level of each keypoint followed by a final step of 2D to 3D conversion of the detected keypoints.

The second step is a second network architecture based on [8] that predicts the 3D location of the detected vehicle semantic points, which characterize the final 3D pose of the vehicle.

- We extensively test our approaches in the KITTI Detection benchmark [20] and vKITTI [7] dataset, comparing it with different methods that aim to solve the same problem. Both methods defined in sub-sections 3.2 and 3.3 are compared between each other, outperforming in both cases other state-of-the-art studies.

3.2 Vehicle Pose Estimation using G-Net: Multi-Class Localization and Depth Estimation

As stated above, in this chapter we present the DNN architecture named G-Net, which is a dual-pipeline algorithm based on multi-class object detection with parallel single-image depth estimation.

The first idea proposed in this chapter is illustrated in Figure 3.2. It is based on a parallel multi-class detection of vehicle parts that characterize its shape (such as vehicle) with a parallel step of single image depth prediction. Both algorithm parts converge on a final calculation of the vehicle orientation, in which the 3D position of the detected vehicle parts lead to final vehicle orientation prediction assuming the vehicle is a rigid body and that all wheel shall be on same plane.

3.2.1 Multi-class detection

As shown in Figure 3.2, the algorithm has a first step with object detection that uses an adaption of the FASTER-RCNN [46] for finding vehicles and later vehicle wheels on the input images. Region-based networks are formed by 3 neural networks: Feature Network, Region Proposal Network (RPN), Detection Network. FASTER-RCNN are an evolution of the region-based CNN (RCNN) [43] which have a similar pipeline but instead of a first RPN (Region Proposal Network) proposes a pre-processing step named selective search, [44]. Such step aims to group image pixels, so that several zones are detected which have a similar colour or similar texture, among others. These groups are then warped in squares and fed into a following CNN that acts as a feature extractor followed by FCNs to predict the bounding box position around the detected object plus a classification of the type of object detected.

This approach proved good results but it had some problems like the high time required to train. To overcome such limitation, an evolution of RCNN were published, the FAST-RCNN, [25]. This new approach applied the selective search on a different step in the pipeline. In this case, the input images are fed directly to the CNN. Region proposals are then extracted through selective search from the feature map created after the CNN, warped in squares and fed to a sequence of FCNs that act as a regressor of the bounding boxes and classifier of the detected objects.

This chapter proposes a pipeline based on a FASTER-RCNN for object detection, which is an improvement of the before mentioned FAST-RCNN that substitutes the selective search preprocessing by a RPN (Region Proposal Network) for the generation of ROIs around the candidate objects. Such new region-based CNNs are formed by three steps:

- Feature extraction network.

- Region Proposal Network (RPN).

- Detection Network.

The first Feature Network is usually a pretrained VGG [29] or ZF [47] passing through the input image generating a feature map with same size and dimension as input data. The purpose of the following RPN is to generate a number of bounding boxes called Region of Interests (ROIs) that has high probability of containing any object. In order to do that, every pixel of the input feature map is considered an "anchor". Each of it is a set of 9 anchor combination with 3 different sizes and 3 different combinations of aspect ratios (1:1, 1:2 and 2:1).

Therefore 9 anchors are generated per anchor. After anchor creation, Non-Maximum Suppression [14] algorithm is applied to ignore overlapping rectangles. A binary class label is then assigned (of being an object or not) to each anchor. A positive label is assigned to anchors with the highest IOU (Intersection-over-Union) overlap with a ground truth box, or to anchors with IOU overlap higher than 0.7, [46]. These labels are used to train the RPN. Finally, the set of extracted feature ROIs are passed as a batch to the Detection Network. This network is formed by initial multiple layers for downsampling the input cropped images that are passed to a similar network architecture as RPN. The final dense layers output for each cropped feature ROI, the score and bounding box for each class.

Following the above mentioned concepts, the presented G-Net implements a FASTER-RCNN to firstly detect vehicles and secondly to detect wheels on the extracted vehicles, since such features characterize the orientation of the detected vehicles. Being able to accurately predict the position of vehicle wheels have some important implications when calculating vehicle pose, such as the characterization of the ground plane (vehicles are expected to be placed on the ground plane).

3.2.2 Single-image depth estimation

Estimating depth is an important component of understanding geometric relations within a scene. Being able to perform this task accurately leads to a better understanding of the environment obtaining better results in recognition or collision avoidance activities. As commented above, image depth maps have been normally calculated using stereo-cameras through feature matching or using other optical sensors to complement the cameras, like LiDARs, that are able to provide a direct measurement of the depth of the surrounding objects. However, such strategies lead to systems that are more difficult to handle and more expensive. For this reason, the implementation of algorithms that aim to calculate depth maps through single camera images, obtaining simpler systems and getting the best profit from cameras have become more and more popular. Finding the depth map from a single image is not a straightforward task. It requires the integration of both global and local information from various cues.

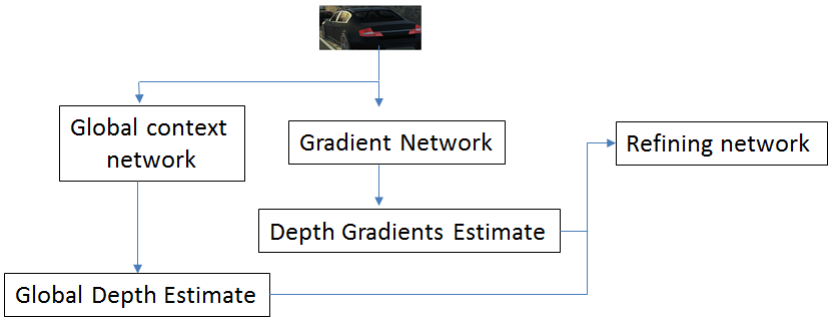


Figure 3.3: Schema of algorithm design for vehicle pose prediction.

In this chapter, we address this task with a multi-step pipeline, similar as [65], as illustrated in Figure 3.3:

- Global prediction based on the entire image.
- Local refinement of this prediction.
- Estimation of gradient depth maps.

3.2.3 Global prediction based on the entire image

The proposed approach for predicting the depth of the input images is based on a first "global context network" that aims to calculate a rough depth map of the whole scene from only RGB images. One of the biggest ambiguities that the calculation of depth maps has is the dependency on the image scale caused by the projection from 3D space to the 2D image. Two images that look identically can in fact depict different real world scenes depending on the distance the images were captured. Since the information about the scene's absolute scale cannot be extracted through vision only, it is reasonable to consider both cases to be identical. We need therefore to use a loss function that is scale invariant, that is, that produces the same loss in same RGB images with different scales. In this case, we consider the following loss function:

$$L(y, y^*) = \frac{1}{N} \sum_i \left(\frac{y_i - y_m}{\sqrt{y_v}} - \frac{y_i^* - y_m^*}{\sqrt{y_v^*}}\right)^2, \tag{3.1}$$

where y is the estimated depth map, y^* is the ground truth, y_m and y_m^* denote mean values of respective depth maps and y_v and y_v^* denote respective variances of these depth maps.

The proposed architecture is AlexNet based and is formed by 5 convolutional layers followed by ReLu as activation function and finishing with two folowing fully-connected layers with dropout in the middle to avoid overfitting, that finally output a low resolution global depth map of the input image. The network design is illustrated in Figure 3.4.

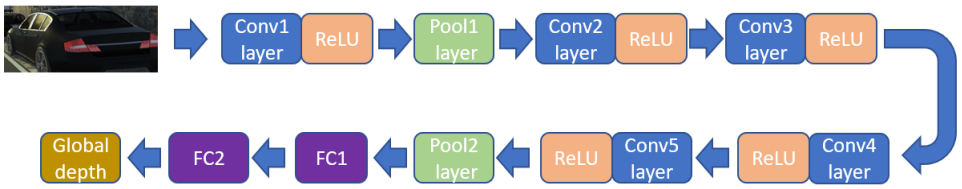


Figure 3.4: Global context network architecture.

3.2.4 Gradient network

The proposed method in this chapter bases on a parallel network to calculate the image gradients that will be trained together with the above mentioned global network. The objective of this network is to calculate both horizontal and vertical gradients of the depth map globally, for the whole RGB image. Input for the gradient network is an RGB image. The network architecture is similar to the global context one. Due to this similarities in the design, both have been jointly implement so that the training is done together for both networks, what proved to improve the results, since estimating depth map and corresponding gradients of the depth map are related tasks. The joint network architecture can be seen in Figure 3.5.

The main difference of both Global Context and Gradient extraction networks is the initialization strategy and weights used, due to the different size in both cases, [66]. The forth convolutional layer in the case of the global context network is initialized with Xavier approach and not with AlexNet because of the different number of input channels.

3.2.5 Refinement network

Last part of the presented pipeline is the refinement network that aims to refine the rough depth map produced by the global context network. This third neural network takes as input the RGB image, the gradients calculated on the second step and the global depth map. The gradients are mean variance normalized before being fed to the refinement network. Refining network is a fully convolutional network and similarly to the other parts of the model, it is also based on AlexNet and thus contains five convolutional layers. Loss function in this case is defined in Eq. (3.2):

$$L(y, y^*) = L_a(y, y^*) + \frac{1}{N}\sum_i (y_i - y_i^*)^2, \tag{3.2}$$

where $L_a(y, y^*)$ is the loss function and the second term is a summation over all depth maps of each estimated depth map minus each ground truth. The refinement network can be found in Figure 3.6. The before mentioned networks were trained with NYU v2 dataset [67] with indoor scenarios and virtual KITTI [7] which covers already the usecases desired for the target application. Images had to be properly scaled and resized to be fed in the implemented networks.

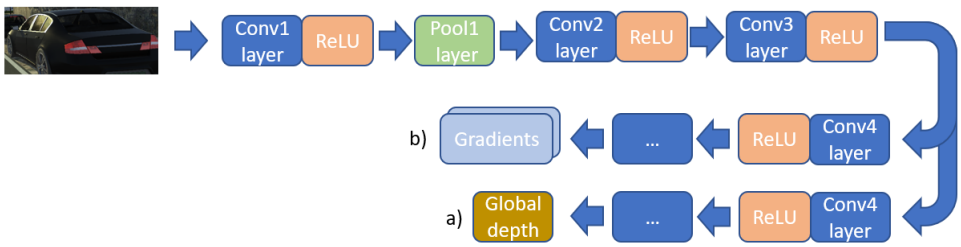


Figure 3.5: Gradient extraction network schema for joint training with global context step. The branch "a" corresponds to the global network architecture and "b" to the gradient extraction path.

It has been used data augmentation strategy for the training and testing datasets when using virtual KITTI following transformations such as: scale changing of input images, rotations of input data a certain number of degrees, translation of some parts of images, horizontal flip, shifting of hue, saturation and value of input or changing of contrast.

3.2.6 Geometric pose extraction

In this chapter it is presented a new method for 3D object pose estimation based on planar images (see Figure 3) with region-based CNNs and weakly supervised learning. Following it will be presented how the pose from the vehicles can be extracted from the already predicted bounding boxes and depths explained in Section 3. Once the net is working, for the extraction of the pose the intrinsic and extrinsic parameters of the camera will be needed, to obtain 3D points from the detected wheels based on mathematical operations done with the camera parameters:

$$X_{2D} = [K][Rt]X_{3D}, \tag{3.3}$$

where x_{2D} are the coordinates of the point in camera coordinate system (image coordinates) and X3D are the coordinates in world coordinate system. The matrix K are the intrinsic parameters and [R t] are the extrinsic of the camera.

As the training dataset is vKITTI [13], the intrinsic parameters of the camera are known (see Eq. (3.3)), and the extrinsic parameters are given per frame.

$$K = \begin{pmatrix} 725 & 0 & 620.5 \\ 0 & 725 & 187.0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Making use of the inferred depth map, it will be estimated if the located wheels belong to the same side of the car or to the front/rear part of the vehicle. The order of the projection of the vector between wheels over the vehicle planes changes if the detected wheels belong to one left/right side or front/rear part.

In the case that wheels from same side are detected, the vector defining the orientation of the vehicle will be calculated as indicated in Eq. (3.4):

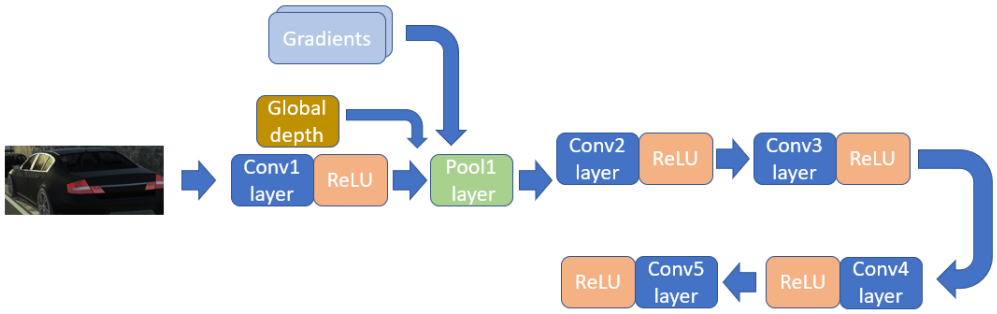


Figure 3.6: Illustration of the refinement network.

$$V(x, y, z) = P_1(x, y, z) - P_2(x, y, z), \tag{3.4}$$

being P_1 and P_2 the center of two bounding boxes in camera coordinate system (CCS). Using trigonometry, the yaw, pitch and roll angle to determine the 3D orientation of the vehicle can be calculated based on the projection of the predicted angle to the vehicle planes:

$$\begin{aligned} yaw &= \arccos \frac{V_{xz} V_z}{|V_{xz}||V_z|}, \\ pitch &= \arccos \frac{V_{yz} V_y}{|V_{yz}||V_y|}, \\ roll &= \arccos \frac{V V_{xz}}{|V||V_{xz}|}, \end{aligned} \tag{3.5}$$

being Vxz the projection of V (see Eq. (3.4)) over XZ-plane and Vyz and Vy the projection of V over YZ and then over Y, respectively.

3.2.7 Experiments and results

In this section we will present some results of the described method and its comparison with other methods. As specified before in this section, the used dataset for training and validation has been a set of 6300 planar images from KITTI [20] and vKITTI [7] datasets. We compare the average predicted pose of vehicles in KITTI dataset using different methods for 3D pose extraction. These results are presented as the percentage of well detected poses and shown in Table 3.1. Our approach has been applied to a subset of 3420 images in which a minimum of two wheels are visible. In Table 3.2 and Figure 3.7 there are some qualitative results which illustrate the fusion of the above-mentioned steps that form the G-Net. We prove an average precision (AP) or true positive ration of 88% when detecting wheels (assuming the test-dataset contains images of vehicles where wheels are visible) and a precision in the calculation of the orientation above 90% as shown in Table 3.1.

Moreover, in Table 3.2 we present the depth maps calculated for three different scenarios, the wheels detected on these images and the vector between them that characterize the 3D shape of the detected vehicle.

Method	Time (s)	Easy	Moderate	Hard
Deep Manta [21]	2	97.44	90.66	82.35
3DVP [68]	40	65.73	54.60	45.62
SubCNN [69]	2	83.41	74.42	58.53
3DOP [48]	3	91.45	81.63	72.97
DPM [70]	-	47.27	55.77	43.59
OC-DPM [71]	-	73.50	64.42	52.40
AOG [72]	-	43.81	38.21	31.53
Mono3D [51]	4.2	91.01	86.62	76.84
G-Net (Ours)	2.3	93.21	86.33	80.90

Table 3.1: Results for orientation extraction (AOS) on validation set. AOS is defined as "average orientation similarity" ($AOS = \frac{1}{N}\sum s(r)$) where s(r) is measuring what fraction of detected car orientations are similar to ground truth car orientations in the image.

Together with this, the quantitative results for these images are also shown as accuracy in the prediction of the shape, calculated as the difference between the predicted 3D orientation and the ground truth. The results from Table 3.2 compare the calculated pose for each detected vehicle through the dataset with its correspondent labelled pose. Important to note is the definition of easy, moderate and hard in terms of percentage of the object occluded [20]:

- Easy: Min. bounding box height: 40 Px, Max. occlusion level: Fully visible.

- Moderate: Min. bounding box height: 25 Px. Partially visible.

- Hard: Min. bounding box height: 25 Px, Max. occlusion level: Difficult to see.

Our G-Net approach clearly outperforms other monocular approaches for the 3D pose estimation task like [51] which searches for vehicle candidates placed on a known ground plane to secondly infer their 3D orientation through class segmentation, instance level segmentation, shape and contextual features. Our experiments show good and trustworthy results although methods like DeepManta [21] achieve better accuracy in the calculation of the vehicle orientation. This is due to the depth map calculation proposed in the second step of the G-Net.

This increases the false positive rate and leads to a bit more inaccurate orientation estimation compared to this work. However, the G-Net obtains good enough results avoiding the usage of big and hard-to-find datasets of 3D models that make the detection pipeline slower and harder to manage.

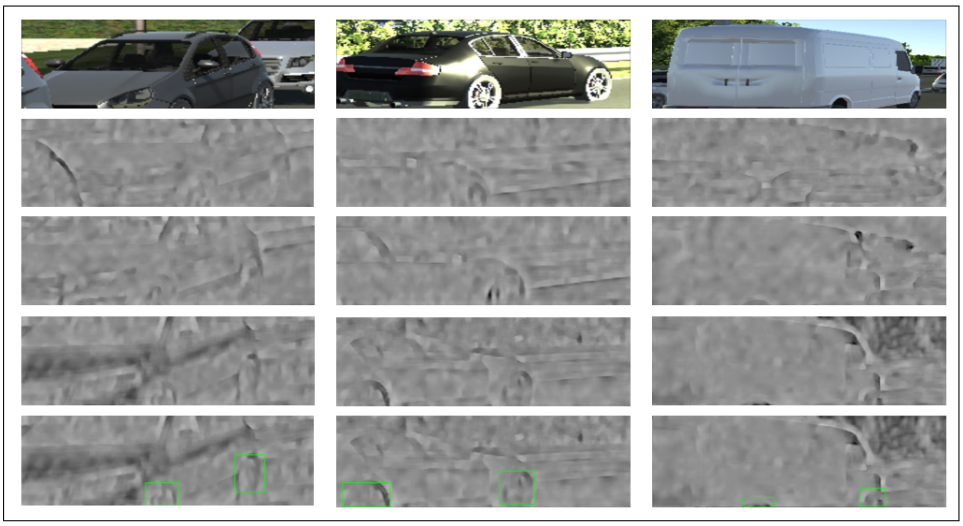


Figure 3.7: Qualitative results on 3 images part of the test-set from vKITTI dataset. First row illustrates the input RGB image of the detected vehicle, the second corresponds to the output of the gradient network, the third row contains the output of the global context network while the forth shows the result of applying the final refining network. Lastly, the fifth row combines the detected wheels located on the input RGB image and the calculated depth map.

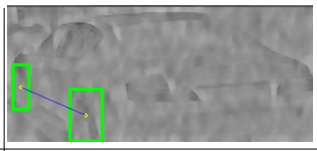
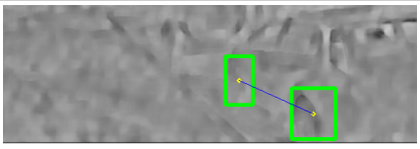
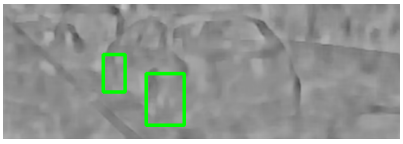
Image			
YAW(°)	0.0045	0.004	0.005
PITCH(°)	0.007	0.006	0.007
ROLL(°)	-1.566	-1.566	-1.566
Acc(rad)	0.002 0.002 0.002	0.06 0.06 0.065	0.06 0.0197 0.0198

Table 3.2: Accuracy of our algorithm on vKITTI images [7] having roll, pitch and yaw calculated based on (6), (7) and (8). For simplicity reasons only three images have been taken to show the performance of the algorithm and the obtained accuracy. We show to get good results as the accuracy, calculated as the difference between the predicted angle and the labelled angle, remains close to zero (predicted and ground truth values shall be as similar as possible).

3.3 Vehicle Pose estimation via Regression of Semantic Points of Interest

Together with the above presented method for vehicle 3D pose calculation, in this chapter we show a straightforward algorithm able to predict the 3D orientation of the surrounding vehicles using only 2D

planar images as input. This approach bases on the precise localization of vehicle semantic points as if they were joints from humans. We propose a distribution of 20 points that characterize the shape and pose of vehicles and propose a DNN able to locate them in the 2D space, so that in a second step the 3D position of these points can be calculated through a second deep neural network to finally infer the vehicle pose.

In this chapter we address the problem of extracting vehicle 3D pose from 2D RGB images. An accurate methodology is presented that is capable of locating 3D coordinates of 20 pre-defined semantic vehicle points of interest or *keypoints* from 2D information. The presented two-step pipeline provides a straightforward way of extracting three-dimensional information from planar images and avoiding also the usage of other sensor that would lead to a more expensive and hard to manage system. The main contribution shown in this chapter is the presented dedicated network architectures that are able to locate simultaneously occluded and visible semantic points of interest to convert these 2D points into 3D space in a simple but efficient way. The presented method uses a robust network based on Stack-Hourglass architecture for precise prediction of semantic 2D *keypoints* from vehicles even if they are occluded. Furthermore, in the second step another dedicated network converts the 2D points into 3D world coordinates and therefore, the 3D pose of the vehicle can be automatically extracted, outperforming state-of-the-art techniques in terms of accuracy.

The presented method overcomes some constraints of the previously mentioned approach, such as the need of the key vehicle elements to be detected to be present in the image (wheels in the previous method). This is sometimes hard to obtain since camera images not always cover the whole shape of surrounding vehicles, therefore this limitation made previous approach not applicable for all scenarios. However, the algorithm presented in this part of the chapter aims to provide a solution to this constraint by implementing a network architecture able to locate and classify vehicle "*keypoints*" present in the image, independently to the percentage of vehicle surface inside in the 2D input planar image.

3.3.1 Related work

Stacked Hourglass Architecture

Over the years there has been many approaches for solving the issue of human pose prediction. Studies like [73] started working to deep networks for addressing this problem. Others like [74] introduced the generation of heatmaps around each body joint that combines an efficient sliding window-based architecture with multi-resolution and overlapping receptive fields.One of the key points of this study is the joint use of a ConvNet and a graphical model that learns typical spatial relationships between joints.

Other works that are based on these networks like [75] organize the detections into typical orientations so that when their classifier makes predictions additional information is available indicating the likely location of a neighboring joint. The hourglass network is based on encoder-decoder architectures [76] and makes use of the convolutional/de-convolutional architecture.

The **Stacked-Hourglass** method proposed by Newell *et al.* in [53] for human pose estimation implied a big step forward into robust and accurate 3D human pose extraction. This network is based on the successive steps of pooling and up-sampling that are done to produce a final set of predictions, as

presented in Figure 3.8.

In the case of human pose estimation, extracting human joints and learning typical spacial relationships between them has proven to provide good results to the problem of human pose extraction. Research works like [75] cluster detections and predict the probable location of a neighboring joint [53]

In this work we use nearest neighbor up-sampling and skip connections for top-down processing. We also perform repeated bottom-up, top-down inference by stacking multiple hourglasses.

This network uses Deep Learning to detect the *keypoints* and predict the heatmap of each of them by drawing a gaussian around these points.

The hourglass is a simple, minimal design that has the capacity to capture multiple features and bring them together to output pixel-wise predictions [53]. The aim of using this network architecture is to obtain full context information important for pose extraction, meaning that, not only the exact prediction of the position of the *keypoints* is important, but also the pose estimation requires a full understanding of the vehicle.

Pose estimation

The appearance of new methodologies for predicting human pose like DeepPose [73] introduced a new approach to the typical ones for solving the problem of human pose extraction. This work presented a network to calculate the (x,y) coordinates of human joints. Further research like [77] proposed a similar idea as the one presented in this work based on heatmap calculation around detected human joints.

Another direction to address the pose estimation problem is comparing several projections from several 3D models with image contours, as done in [21] or [78]. This work uses the 3D projection compared with image contours to refine the pose estimated by discriminative part based model detector using the Pascal3D+ dataset [79]. Similar to this approach and also based on available 3D models, other works have sampled the object position, size and viewpoint to compare the projection of 3D models on 2D space with the detected 2D object using histogram of oriented gradients (HOG) features and check possible correspondence between 2D-3D object, like [80]. Latter on, one improvement based on these approaches appeared: 3D Voxel Pattern (3DVP), that jointly encodes the key properties of objects including appearance, 3D shape, viewpoint, occlusion and truncation [68]. This work has a multi-step processing pipeline starting with an alignment of 2D images and the 3D rendered models, followed by the generation of the voxel exemplars used for training the network together with the voxel patterns (ground truth). A detector for each 3D voxel pattern is obtained that detects specific visible characteristics on each object. Through the calculation of these visible characteristics a 2D segmentation of the image is obtained that leads to the calculation of the 3D pose.

Based on this principle and on [53] the presented design on this section has a two-step pipeline with first accurate *keypoint* detection and heatmap calculation and final step of pose extraction based on that prediction.

More recently and more focused on pose extraction for vehicles, new studies came out showing accurate results like [21] or [63]. The presented pipeline in Deep Manta is based on a precise vehicle detection phase following a region-based CNN followed by a refinement step using Non-Maximum Suppression [14] algorithm and ending with a 2D-3D matching phase for comparing the extracted 2D

vehicles and their 2D information (visible parts, part coordinates, etc.) with multiple 3D models the extract the best 3D model that would fit into the information extracted from the first step of the pipeline. This network includes a semantic segmentation feature that runs in parallel with the detection, so that this method is able to provide pixel-wise classification of the image, as well as accurate class localization. This method has shown good results in pose estimation by adopting this Mask R-CNN for predicting K-masks (being K the number of keypoints) followed by a 3D point conversion of the obtained 2D *keypoints* for the pose calculation.

The pipeline presented in this work tries to address and solve some of the limitations presented in works like Deep Manta, that have a very good performance but require a lot of resources in terms of training image-set for the vehicle detection phase and big 3D model set for the 2D-3D matching step. As shown in Table 3.4 this work presents good results in extracting the pose from single vehicles and has achieved an efficient and accurate pipeline for semantic *keypoint* prediction. One of the big achievements in this work is also the prediction of occluded points (by other vehicles or by the same vehicle) that leads to a better calculation of the pose. Many other approaches have been published and presented also good results in vehicle 3D pose extraction such as [48, 51]. They have also shown accurate results using similar approaches as the Deep Manta, the first object proposal extraction through CNNs followed by pose extraction through an energy minimization approach that places object candidates in 3D using the fact that objects should be on the ground-plane.

Following a similar approach than [81] in this work we represent the 3D pose from a vehicle with N=20 keypoints and parameterized by a 3N vector P = $[p_1, \ldots, p_N]$, where p_i is the 3D location of the i-th keypoint. Similarly, 2D poses are represented by 2N vectors U = $[u_1, \ldots, u_N]$, where u_i are pixel coordinates. Our goal is then estimate the 3D pose vector y. This will be achieved by using a simple but effective network architecture, adding residual connections and using batch normalization, trained on vKITTI dataset [7] with labelled vehicle *keypoint* and taking into consideration the camera frame as global coordinate frame following the idea of [8] since this makes the 2d to 3d problem similar across different cameras.

Commonly the problem of extracting 3D information from planar images has been solved by using stereoscopic cameras. Identifying the same pixels viewed from both cameras and knowing in advance the relative position of both cameras solve the third degree-of-freedom of the depth that is unknown when using single cameras. Stereo cameras usually consist of two parallel cameras with overlap in their field of views. Many research papers used these cameras for extracting human-pose form only planar images, like [82] or [83].

Recent studies like [19] have used this concept from the human-pose problem and have applied to vehicle pose extraction showing promising results.

In this work, we show a new approach for vehicle 3D pose estimation using a powerful state-of-the-art network architecture that has good results compared to other existing methods.

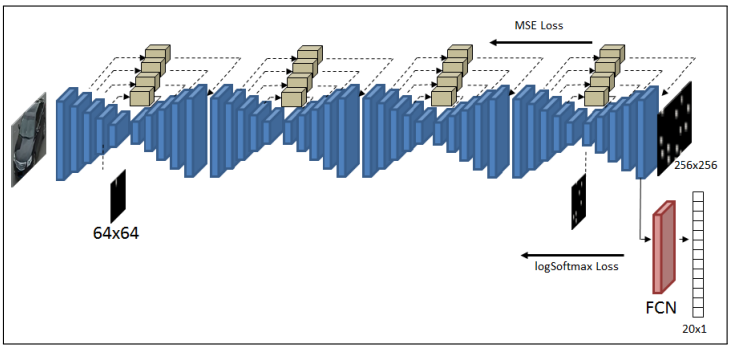


Figure 3.8: Presented network architecture with 8 loops of down-sampling/up-sampling and parallel occlusion prediction. Heatmaps extracted from encoder/decoder architecture are followed MSE (Mean Squared Error) loss calculation and occlusion prediction is followed by a fully connected layer and posterior softmax loss calculation. A parallel residual block runs with the series of convolutional / deconvolutional for a more precise up-sampling of the image from 64×64 to input resolution (256×256).

3.3.2 Proposed method

We use input images with a resolution of 256×256 pixels with their semantic keypoint labeled as shown in Figure 3.9 that correspond to images in which only one vehicle is present, as explained in [84]. The vehicle in the image can be in multiple positions and training and validations image-sets correspond to both real and virtual images (virtual images extracted using vKITTI simulator [7]). This method also predicts the probability of a *keypoint* of being occluded based on the notation of the training dataset . This probability is displayed together with the calculated heatmaps with gaussians around predicted keypoints.

This network needs full input resolution of 256×256 and the highest resolution of the hourglass is 64×64. The full network starts with a 7×7 convolutional layer with stride 2, followed by a residual module and a round of max pooling to bring the resolution down from 256 to 64 [53].

Given an input image, the network joint optimization minimizes the global function:

$$L = L_1 + L_2, \tag{3.6}$$

being L the global network loss function, L_1 the loss function for the heatmaps prediction following the least squares method and $L_2$the loss function for the occlusion prediction.

$$L_1 = \frac{1}{N}\sum_i (p_i - p_i')^2, \tag{3.7}$$

$$L_2 = log(\frac{e^{y_i}}{\sum_i e^{y_i}}). \tag{3.8}$$

In the loss calculation is p_i the heatmap probability for *keypoint* "i", p_i' the *keypoint* location as ground truth for *keypoint* "i" and y_i is the i-th position of the output vector of the final FCN layer for the occlusion

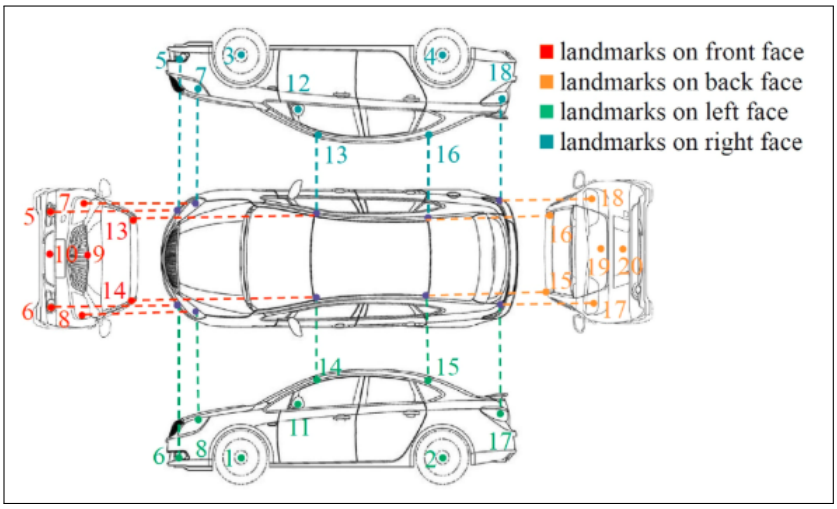


Figure 3.9: Labeled semantic *keypoints* on the vehicles present in the training dataset [6].

prediction in the forward pass.

Dataset generation

For the purpose of this research we have used a set of 43600 images of different vehicles in different positions in which 20 semantic points have been labeled, following the procedure defined by [6]. We have increased the real images provided in the dataset with new images of vehicles created with vKITTI dataset [7] and notated in a similar way as in Figure 3.9.

Also, the original VeRi Dataset was extended with the notation of the occluded *keypoints* in the images plus one binary term indicating if the *keypoint* is occluded or not. This labeling is necessary for the occlusion prediction explained in Section 3.3.3. As explained in Section 3.3.2 together with the *keypoint* detection, this method is capable of predicting the probability of a *keypoint* to be occluded in the image. This is obtained by the labeling of the training and validation dataset, by defining the position of the *keypoints* followed by a binary term (0 or -1) indicating not occluded or occluded respectively. For the final step of the presented processing pipeline a dataset based on known virtual Kitti images [7] has been created and self labeled in a similar way than the real dataset used for the *keypoint* detector. As explained in 3.3.2, the training data for the 3D calculation network have been pre-processed (rotated and translated) to have all training data in a single global camera frame [8].

After this, the second network of the presented pipeline was fed with the coordinates of the labeled semantic vehicle points (instead of the images), which provide less information to the network but reduce drastically the computational need of the network, making them easy to work with. The presented network is able then to learn the 2D-3D correspondence between training data points and provides therefore a set of 20 x,y,z coordinates corresponding to the detected *keypoint* in 3D space.

Finally, a second transformation of the provided output data will be applied to present the vehicles poses in the same vehicle coordinate system.

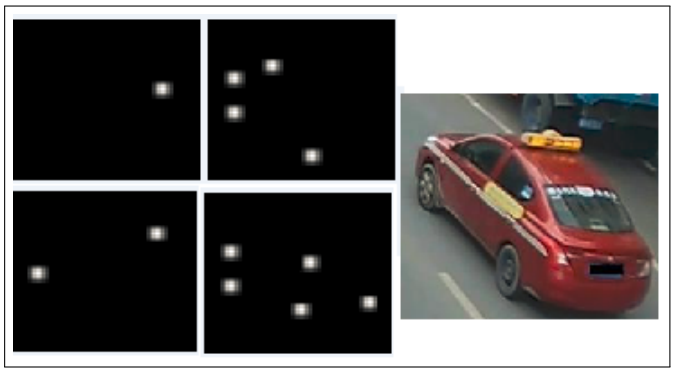


Figure 3.10: Predicted heatmaps with gaussian drawn around detected *keypoints* extracted during training and validation phase.

3.3.3 *Keypoint* prediction

Once the dataset has been generated, we follow the network architecture presented in Section 3.3.2 to train the model with 35,000 images for predicting the semantic *keypoints* and the occlusions of these keypoints.

For this purpose, six steps of encoding-decoding as part of the applied Stacked-Hourglass architecture have been implemented. The training images will come into each processing step and the output of the processing will be input of the next loop of encoding-decoding. This mechanism allows the network to learn not only local but also global context of the extracted features, which is one of the main advantages of this network architecture.

These steps allow the network to obtain several features from the training data We will then obtain a set of vehicle part candidates Δ_J for multiple keypoints. We define Δ_J as a set of vehicle keypoints defined as 2D points in pixel coordinates.

As detailed in the architecture, after the down-sampling/up-sampling phases of the Stacked-Hourglass we calculate the MSE (mean squared error) as loss function and feed it as input for the back-propagation phase.

In parallel to the *keypoint* extraction, we predict the occlusion of the detected kepoint by labeling it properly in the ground truth with the position of the vehicle *keypoint* plus one third number indicating if the point is occluded or not. As shown in Figure 3.8 we have added one output to the hourglass network for the occlusion prediction including a final FCN (fully connected layer) plus a logarithmic softmax error calculation that will provide the probability for the point to be occluded. Therefore, this network is not only able to detect the 2D position of visible *keypoints* of the vehicle, but also their position when they are occluded (partially or totally) and their possibility of being occluded, as shown in Figure 3.10. The error during training of the occlusion of the keypoint.

3.3.4 3D vehicle pose calculation

Typically in human pose extraction there are several approaches that have proven good results like [85–89], or [90].

Figure 3.11: Results of *keypoint* detection vs. ground truth labeling considering also self-occluded points in the vehicle. The yellow circles are the predictions and the red squares mark the ground-truth with non-occluded points. Blue circles are ground truth and red circles are predictions of *keypoint* position with self-occluded points (partially or totally occluded).

One of the main limitations of these proposed methods was the need of large training and validation datasets.

For that reason, it seemed reasonable the appearance of new methodologies splitting the pose estimation in a two-step pipeline [91, 92] and due to the good results provided by these approaches for human-pose extraction problem, we applied a similar idea to the vehicle pose calculation issue.

In research studies like [81] the second step of pose extraction is based on the representation of 2D and 3D poses using $N \times N$ matrices of Euclidean distances between every pair of joints. In the case of the presented research however a similar approach has been applied but using a common global representation frame and common coordinate system to correlate the 2D and 3D extracted information and to ease the problem. Another constraint of the system taken into consideration for the second step of the presented pipeline is that vehicles are rigid bodies in which the relative movement between *keypoints* from frame to frame very limited is. In the problem of human pose, the relative movement of the human joints is also limited and can be constrained when working with pair of joints. A similar idea will be then applied in this work to reduce the number of false positives and improve the results.

In this proposed research, once the 2D *keypoints* have been precisely detected, we would need to convert them from 2D to 3D to obtain the 3D pose of the vehicle in the image.

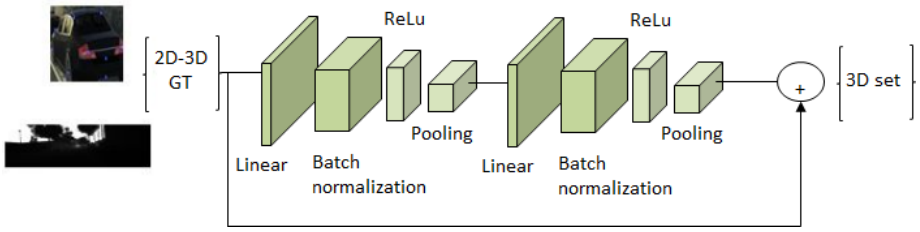


Figure 3.12: Implemented network for converting 2D coordinates to 3D coordinates using vKITTI dataset [7] and using a network architecture based on [8].

For that, this work proposes a network architecture formed by consecutive linear layer, batch normalization, RELU and dropout (Figure 3.12). This network is based on [8] which is meant for human pose estimation but adapted to the purpose of this research of obtaining the vehicle *keypoint* from an image with a single vehicle.

Before passing the extracting 2D *keypoints* to the second network, an additional processing on the data will be needed. The predicted points will be calculated for camera frame, since that makes a 2D to 3D problem similar across different cameras, [8]. As explained in that work, that is done by rotating and translating the 3D ground-truth according to the inverse transform of the camera.

Following this idea, the 2D *keypoints* will be translated and rotated to the camera frame according to the inverse transform of the camera. For training this network we make use of images from the vKITTI dataset [7] due to its full labeling in 2D and 3D, tuned for this purpose. This dataset has depth annotated and the intrinsic and extrinsic parameters of the cameras are known. For the generation of the training dataset the virtual images had to be first cropped and labeled following the semantic *keypoints* explained in Section 3.3.2 following the proper coordinate system. For the training, more the 5000 images were used as for the validation phase a subset of vKITTI and a subset of KITTI Benchmark [20] properly labeled for this purpose was utilized. We apply standard normalization to the 2d inputs and 3d outputs by subtracting the mean and dividing by the standard deviation.

By training the explained network in Figure 3.12 we are able to convert the detected 2D vehicle *keypoints* to 3D coordinates, so that we can extract 3D information from planar images, which was the main goal of the research.

Following the approach proposed by [8] we have avoided the use of raw images for training the proposed network architecture and use 2D and 3D points labeled in the determined dataset. Although these contain less information as the image, using points we achieve bigger training speed. In the presented work we have trained this second network for 5000 epochs, obtaining a mean error of 43mm between labeled 3D position and predicted one.

Once the vehicle *keypoints* have been converted to 3D coordinates, the pose will be extracted by calculating the direction-vector of the vehicle in VCS (Vehicle Coordinate System) as shown in Figure 3.15 and following the methodology for pose calculation proposed in [93], which origin is on the rear axle on the floor and in the middle point between the rear wheels. The direction-vector will be extracted starting on the origin of the VCS and pointing to the *keypoint* 9 (front side). Comparing the calculated pose with the labeled vehicle orientation from the vKITTI dataset the results of Table 3.4 were extracted.

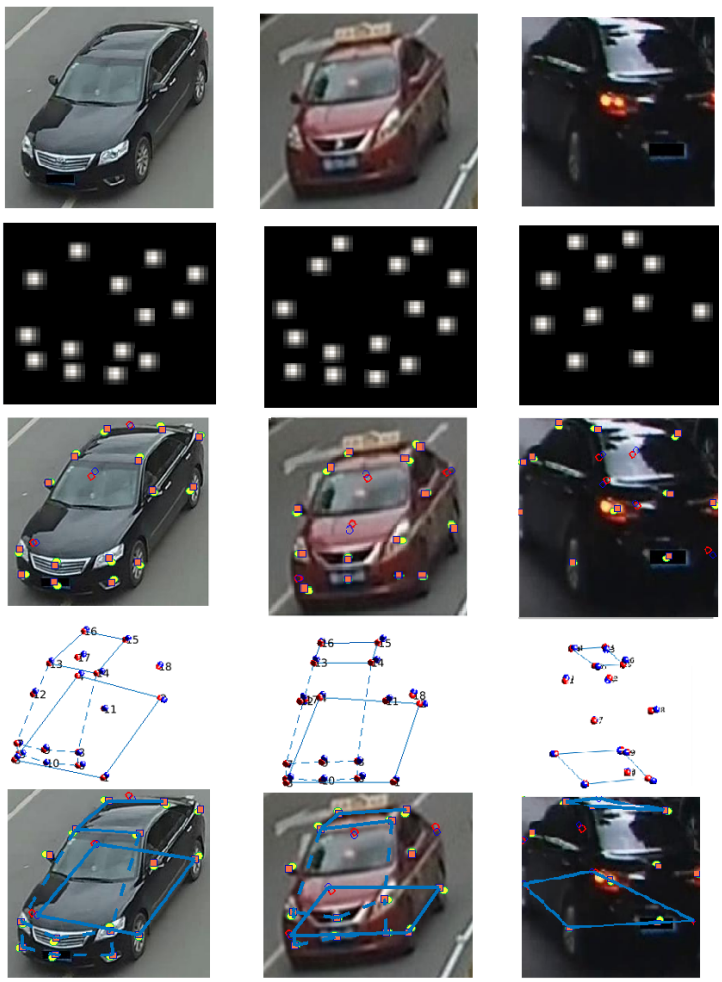


Figure 3.13: Predicted 3D semantic *keypoints* from calculated 2D *keypoints* . For each column, the original vehicle image is shown first, followed by *keypoint* heatmaps, detected vehicle semantic *keypoints* on the original image (being the filled red circles de 2D ground truth, filled yellow circle the predicted *keypoint* (non-occluded), the red circumferences the ground truth for occluded *keypoint* and lastly the blue circumferences is the predicted occluded *keypoint*. Underneath this image is the predicted 3D model of the semantic *keypoint* and lastly is this model adjusted to the original image to compare the results.

3.3.5 Evaluation and experimental Results

In this section we will evaluate the presented method and some results of the described method and its comparison with other methods. Some qualitative results are shown in Figure 3.13 or Figure 3.11. As specified before in this document, the method proposed in this research is based on two steps or two networks that work consecutively:

- Stacked Hourglass for vehicle *keypoint* detection.

- Linear networks for converting 2D coordinates to 3D and therefore extract the 3D pose.

Evaluation on virtual images

This section verifies the performances of the defined algorithm on virtual images extracted from the virtual KITTI dataset [7]. For this evaluation 1500 virtual images were extracted with different car models in different poses to be processed by our pipeline. The labeling of these vehicles was done following the specified steps on Section 3.3.2 and the pose was calculated on the determined coordinate system in 3.3.3. Results are presented in Table 3.3 and Figure 3.14. This method shows good performance not only on virtual environment but also on the real world, as the results from Table 3.4 show.

Evaluation on real environment

The evaluation of the first part of the network is presented in Table 3.5 and is based on two metrics. The first is simply its prediction accuracy in percentage of correct *keypoints* (PCK) with a threshold of 15 pixels (similar than [94]). The PCK is evaluated as the percentage of trials where the euclidean pixel distance between the actual and predicted joint location. The second metric will be the distance from predicted and actual *keypoint* positions.

These metrics have been obtained over the test image set consisting on 2300 images of 2D vehicles and are presented as the percentage of correct *keypoint* per point, following the same semantic *keypoint* description as in the labeling of the training data (see Section 3.3.2). Together with the 2D coordinates of the predicted vehicle characteristic points come the occlusion probability, calculated in the same step as the point locations as a parallel process from the encoding-decoding architecture. In Table 3.5 the results on occluded and not-occluded points is distinguished. The inferred 2D position of the occluded points is very relevant for the global vehicle pose extraction, but due to the occlusion the detection precision is lower on occluded points than on visible. To compensate this lower accuracy, in the pose extraction step of the pipeline together with the 3D coordinates calculation a refinement phase will be applied. This consists on applying several known constraints of the vehicle as non-rigid body like relative position of vehicle *keypoints* or body symmetry.

Once these points have been extracted, the vehicle pose is characterize. We will only need a good inference from 2D coordinates to 3D to create the vehicle 3D model.

As explained in section 3.3.4 we make use of a state-of-the-art network architecture [8] as baseline, we have adapted it for the purpose of this work of 20 *keypoint* transformation and trained it using the vKITTI dataset [7]. With this information, the 3D pose has been predicted. Some results of this prediction can be shown in Table 3.4.

The experimental results, as explained in Section 3.3.4 obtained with this method for the *keypoint* detection have been compared with other state-of-the-art methods for vehicle *keypoint* extraction and human joint detection. These comparison can be seen in Table 3.4.

Figure 3.14: Predicted 2D *keypoints* on synthetic vehicle extracted from virtual KITTI dataset.

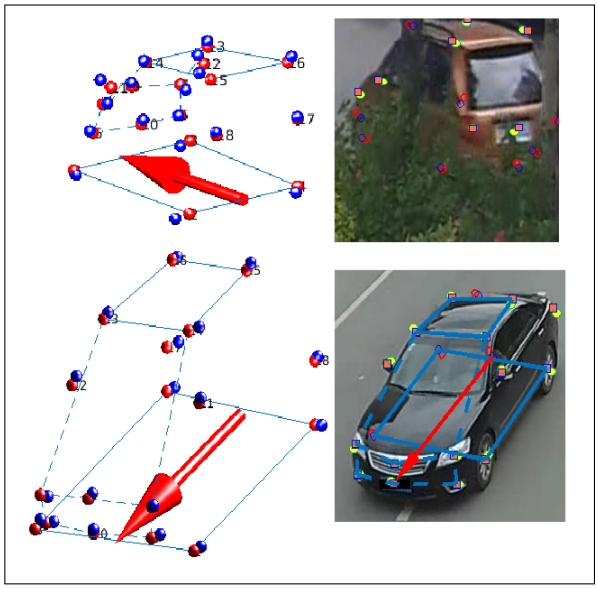


Figure 3.15: Pose extraction on Vehicle Coordinate System. Red arrows represents the direction-vector from the orientation vehicle with origin in the center of the rear vehicle axle.

Figure 3.16: Example of *keypoint* prediction with "hard" and "moderate" occlusion.

Method	Rotation (°)	Translation (cm)
Viewpoints [95]	9.10	N/A
3DVP [68]	11.18	N/A
ObjProp3D [96]	17.37	N/A
Reconstruct [97]	12.57	N/A
Monocular [19]	2.87 / 4.4134	4.73 / 6.21
Ours	3.40	6.10

Table 3.3: Results for pose extraction on test set of KITTI dataset evaluated as orientation and translation errors. These errors of the estimated pose with respect to the ground truth are expressed as geodesic distance (Eq. 3.9) for the rotation and distance between the centroids of two point sets (Eq. 3.10) for the translation error [19] respectively.

We use average orientation similarity (AOS) to evaluate vehicle orientation as proposed by the KITTI Benchmark [98]. The results shown in Table 3.4 demonstrate a good performance of the proposed approach in terms of orientation extraction for different occlusion possibilities.

Anyway the numbers of Deep Manta [21] show a light better performance due to the 2D-3D classification phase with many 3D models to identify the kind of detected vehicle presented in that work. One of the breakthroughs of the research shown in this section is the achievement of very good results avoiding the collection of big 2D and 3D datasets and their corresponding labeling, which could be a difficulty when facing vehicle pose estimation problem. We obtain 2D detections using the state-of-the-art stacked hourglass network of [53] trained on the our dataset based on our extended version of the VeRi [99] and [6]. The average error between these detections and the ground truth 2D points is 5.3 pixels, which is in a similar order of magnitude as the one for human pose estimation calculated by [8] and [81].

For the purpose of this research the Stacked Hourglass model has been extended to include a parallel learning of the probability of occlusion of the predicted keypoint, as described in section 3.3.3 in which a obtained training and validation error of less than 5 pixels is shown. The accuracy of the pose reconstruction has been evaluated over testset of KITTI dataset by obtaining the error in orientation and pose calculation. These results are shown in Table 3.3. In this work we used hardware was 2 GPU NVIDIA GTX 1080, the training was done over more than 40000 images of only vehicles in different poses. For the second network used for predicting 3D coordinates from 2D points, the used hardware was the same and the training and validation took place over more than 15.000 virtual images from virtual KITTI dataset [7] and KITTI dataset, as explained in 3.3.4.

3.3.6 Conclusions

Here it is presented a method for extracting 3D information from planar images that makes use of state-of-the-art network architecture, like the Stacked Hourglass networks, to predict 2D *keypoints* from images of vehicles. An accurate *keypoint* extractor would lead to an accurate vehicle detector in which only its shape can be detected.

Method	Hard	Moderate	Easy
Deep Manta [21]	80.55	89.91	96.32
3DVP [68]	65.38	75.77	87.46
SubCNN [69]	76.68	88.62	90.67
3DOP [48]	76.62	86.10	91.44
DPM [70]	46.54	61.84	72.28
OC-DPM [71]	52.40	64.42	73.50
AOG [72]	24.75	30.77	33.79
Mono3D [51]	76.84	86.62	91.01
Voxel [68]	78.29	65.73	54.67
Ours	79.25	86.11	92.47

Table 3.4: Results for orientation extraction (AOS) on validation set. In order to make comparative results with other state-of-the-art methods, the test set of the KITTI Benchmark [20] was used. The labeling of this dataset has the occlusion distinguished between visible point (easy), partially occluded (moderate) and very occluded (hard). The obtained results are only lightly worse that the Deep Manta [21] due the 2D-3D matching phase proposed on that method with multiple 3D model of all possible cars in the image. The breakthrough of this research is to obtain good results in a similar application as Deep Manta without needing any 3D model dataset and making the method agnostic to the possible models to find in the image.

In this work, the extracted *keypoints* were fed into a second network for calculating the 3D coordinates from the 2D points. Getting these 3D points would allow us to calculate the 3D shape of the vehicle and therefore the pose, which was the main goal of this work.

This shown methodology performs as good as many state-of-the-art methods (see Tables 3.4 and 3.3) in terms of accuracy and performance. The main contribution of this work is the implementation of a new pipeline based on Stacked Hourglass CNNs for *keypoint* detection and simple network architectures for predicting 3D coordinates.

$$L_{Rg} = \frac{\left\| log(R_{pred}^{T} R_{GT}) \right\|_{F}}{\sqrt[2]{2}}, \tag{3.9}$$

$$L_{Rt} = \left\| \frac{\sum_k p_{predk}}{N} - \frac{\sum_k p_{GTk}}{N} \right\|. \tag{3.10}$$

This work proofs good results in comparison to other methods with similar purpose for point detection (occluded and non-occluded points) and it presents an efficient way of extracting 3D information without needing big datasets of 3D models or any other measurements from sensors that could be hard-to-manage and difficult to work with.

Keypoint	PCK1 (%)	PCK2(%)
Front left wheel	92.26	65.12
Rear left wheel	90.11	62.45
Front right wheel	93.14	68.39
Rear right wheel	88.74	59.25
Front right anti-fog	86.89	66.27
Front left anti-fog	84.41	61.88
Front right light	72.74	61.49
Front left light	69.31	52.97
Front brand symbol	94.11	68.67
Front license plate	93.23	64.98
Left mirror	88.14	72.03
Right mirror	87.69	72.34
Front left roof corner	76.12	63.91
Front right roof corner	68.64	58.91
Rear left roof corner	70.45	63.27
Rear right roof corner	61.98	53.41
Rear left light	88.54	71.13
Rear right light	89.67	69.47
Rear brand symbol	89.14	69.91
Rear license plate	88.23	68.24

Table 3.5: Table representing the accuracy of the semantic *keypoint* detection measured in PCK with a threshold of 15 pixels (second column, PCK1). This has been evaluated over 12077 images of vehicles in different positions. Third column, PCK2, shows the performance of the *keypoint* detection only for points labeled as occluded (partially or totally).

4

Object detection on 3D point clouds running on FPGAs

4.1 Introduction

Typically a LiDAR device places a number of laser scanners vertically and rotates them azimuthally to scan the surrounding obstacles. These sensors are optical devices formed by a rotating body, a light source, an IMU and a GPS. They work in such a way that the light source sends light beams (number of light beams depends on type of LIDAR) that reflects on a mirror placed inside the sensor body so that they are oriented outside the LIDAR itself. Since the body is rotating around its axis there are light beams being sent 360º around the sensor aiming to reflect on the surface of the surrounding objects. Such devices measure the time between light beams are sent and received back, after they have reflected on the surface of an object. These sensors are therefore able to provide direct measurements of the depth of surrounding bodies, which is crucial for several applications such as trajectory calculation in autonomous cars or collision avoidance in robots.

LIDAR sensors provide sparse point-clouds and have a variable point density. They provide reliable depth measurements that can be used to accurately predict object's shapes and position. These sensors are therefore powerful devices which measurements can lead to robust and reliable algorithms. However, despite the advantages these sensors can provide, they have some drawbacks. The data they provide is harder to handle since the treatment of several 3D points (around 100k points in one recording campaign) is heavier than handling of 2D images and this impacts on the type of platform selected to treat the data, because it shall have some minimum hardware resources available to process such big data. Such constraint together with the elevated price these sensors can have in the market, limits nowadays the usage of LIDARs in mass producer companies and they normally choose low-price cameras over these sensors.

However, over the last years some cheaper LIDARs have emerged on the market what have motivated companies to choose them for their autonomous systems and avoid the usage of redundant cameras.

They aim to get as much information of the surrounding environment as possible with a single LIDAR,

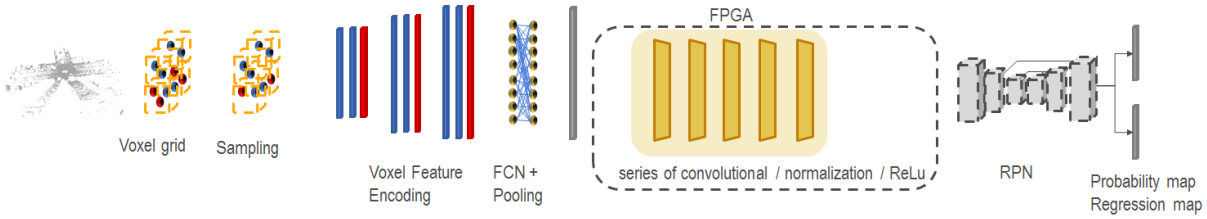


Figure 4.1: Implemented pipeline proposed in chapter 4.

avoiding the need of other sensors, obtaining therefore a simpler and cheaper system. This motivated several research works to publish papers related to object detection on 3D point clouds, such as [100] or [101].

These research works proposed a method based on a first projection of 3D point-clouds on a 2D plane so that secondly, typical feature extraction techniques over images can be applied on the projections. Other works, like [102, 103] or [104], propose a direct processing of the point-clouds to convert them into a 3D voxel grid and encode each voxel with handcrafted features, like point density inside the grid, intensity or reflectance. These have proven to provide accurate and fast results on object detection tasks applied on LIDAR point-clouds and the method proposed in this chapter bases on a similar idea.

Despite the good results provided by these works, the need of manual handcrafted features prevent them to be fully exploiting 3D information provided by the point-clouds. To move from the manual hand-crafted features to automatically learned features, some interesting research papers were published recently: PointNet [105], PointNet++ [106] and VoxelNet [9]. These methods propose frameworks able to learn feature directly from the 3D point-clouds. The first two show good results in feature extraction and object detection directly on 3D points but require much computational power and memory available to treat these points. However, VoxelNet work presented a framework able to reduce the hardware requirement needed without compromising the detection accuracy on 3D point-clouds achieved. A similar architecture as proposed in that work was implemented in the method showed in this chapter.

The presented algorithm is presented in Fig. 4.1 and is able to simultaneously learn discriminative feature representations form point-clouds and to predict accurately 3D bounding boxes. It bases on 3 main steps, similar than [9]:

- Feature learning network (FLN).

- Convolutional network (CNN).

- Region proposal network (RPN).

The main contribution of this work is the implementation of a network able to precisely detect 3D objects from 32-channel LIDAR sensors extracted from Nuscenes dataset, [10] (with lower point density as proposed in [9]), achieving good results in terms of detection accuracy compared to other state-of-the-art methods and, further on, the adaption of the proposed network to be executed on a hardware-constrained platform, such as an FPGA (see Chapter 4.4).

4.2 Related work

4.2.1 Image based 3D object detection

Over the last years, many image-based approaches for 3D object detection have been presented showing several ways of predicting 3D information from 2D images such us [48–51] or [52].

Studies like [53] have showed very good results when calculating the 3D human pose from joint localization. This is achieved by passing the input image through several hourglass phases, to generate heatmaps to capture features at various scales. The motivation of doing so is the need of spatial information for calculating the pose. An understanding of the whole body is crucial to infer the body pose.

The architecture of hourglass architecture is formed by a Convolutional and max pooling layers used to process features down to a very low resolution. After reaching the lowest resolution, the network begins the upsampling to the original resolution and combination of features across scales. Hourglass networks are symmetric, so for every layer present on the way down there is a corresponding layer going up. After reaching the output resolution of the network, two consecutive rounds of 1×1 convolutions are applied to produce the final network predictions. The output of the network will be the mentioned heatmaps where the human joint will be for each one predicted with pixel accuracy [53].

These approaches tend to use the texture information provided by the input images to predict the 3D bounding boxes from 2D images. However, the accuracy of image-based 3D detection approaches are bounded by the accuracy of the depth estimation. One of the reasons that motivate the usage of LIDAR point clouds when solving the issue of the 3D bounding box calculation is this one, since these optical sensor already provide depth measurement and no error is included in the detection pipeline when predicting the depth.

4.2.2 LIDAR sensors

There are several LIDAR point clouds disposition or pre-processing approaches for machine learning and deep learning applications used over the years. Studies like [107, 108] or [109] the LIDAR points were projected onto the image for later feature extraction.

Other approaches like [110] created dense depth map from the LIDAR point cloud to afterwards use this as input for machine learning techniques and infere 3D shapes.

On the other hand, more recent works such as [111] proposes a point cloud processing for transforming them to view and top-view image and combine these with the input image.

This research uses the sparse point clouds directly from the LIDAR sensor following a similar approach as the one proposed in [9] without treating this input data and avoiding the usage of planar images as training data. The VoxelNet approach then compensates the high disparity and variance of the input data by following these steps:

- Voxel creation: 3D gridding is calculated through the input scene to divide it in differents voxels of a variable size depending on the object to be located. The points belonging to each voxel will then have been grouped after this first step.

- Random sampling: To avoid the different number of point that could be contained in the different voxels, a random sampling of points inside each voxels with a number of points bigger than a predefined threshold is conducted.

- Stacked Voxel Feature Encoding: One key of the work of [9] is precisely this encoding step, in which the points inside a voxel are converted into concatenated feature with surface information and geometrical information.

- Sparse Tensor Representation: Once the voxel features together with the voxel spatial information is obtained, a tensor with this information is created. This representation reduces the memory usage and computation cost during backpropagation.

- Convolutional middle layers: The convolutional middle layers add more context to the shape description by passing the tensors through convolution, batch normalization and ReLu to add more context to the shape description inside the tensor.

- Region Proposal Network: A probability score map and a regression map are finally calculated by passing the feature maps from the previous CNN to three FC Layers for downsampling-upsampling for obtaining the high resolution feature map.

4.3 Our method: Feature learning network (FLN)

This first step starts with the creation of a 3D grid around the 3D space containing the point-cloud provided by the LIDAR. Gridding is equally spaced and through this, we obtain several 3D cubes that encapsulate 3D points but with a variable number of them inside of each cube or voxel. To overcome this variability and save computational power on voxels with high point density, a **random sampling** process is applied on voxels containing more than T points, so that we can limit the maximum number of 3D point inside voxels and we therefore balance them.

Random sampling is followed by **Voxel Feature Encoding (VFE)** which is the step in which the network learns features from the 3D points automatically. Figure 4.2 illustrates this step, similar as proposed in [9]. As showed in Fig. 4.2 3D points are firstly converted into pointwise input denoted as $\boldsymbol{V_{in}} = \{\boldsymbol{p_i} = [x_i; y_i; z_i; r_i; x_i - c_x; y_i - c_y; z_i - c_z]^T \in \mathbb{R}\}_{i=1...t}$. These are fed into a FC network in which information from the point features is added to encode the shape of the surface encapsulated inside the voxel. These point-wise features are sent then to a point-wise max-pooling layer to add local features from input feature space. Finally, concatenating these two outcomes, we obtain the features (f_i) from the input voxel encoded in a feature set, $V_{out} = \{f_i\}_{i...t}$.

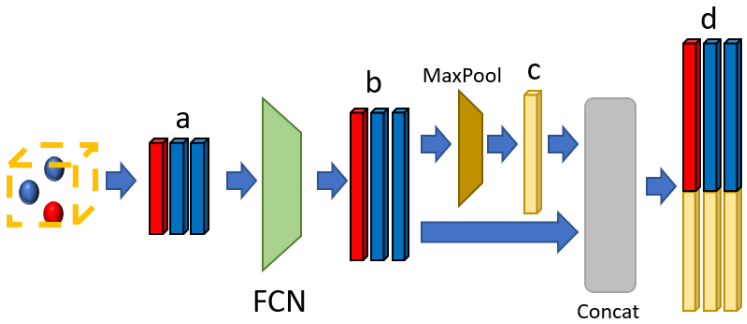


Figure 4.2: Illustration of Voxel Feature Encoding (VFE) step [9]. This sequence of layers aims to obtain tensors with global and local feature information from the input 3D points. In the image, reference "a" represents the pointwise input of the FCN network, "b" represents the correspondent pointwise feature after FCN, "c" makes reference to the locally aggregated feature obtained after the element-wise max-pooling steps and finally, "d" represents the concatenated pointwise features plus the local features.

The output feature combines both point-wise and local aggregated features, so that VFE encodes point interactions within a voxel and provides a feature representation to learn descriptive shape information.

VFE step is followed by a **sparse tensor representation** to illustrate the voxel-wise features, with dimension C, into a 4D tensor of size $C \times D' \times H' \times W'$. We follow a similar nomenclature as proposed in [9] in which point range D, H, W are the depth, height and width respectively of a cube encapsulating whole point-cloud along the Z, Y, X axes respectively. Each voxel has a size of v_D, v_H, and v_W and, therefore, each 3D voxel grid has a size of $D' = D/v_D$; $H' = H/v_H$; $W' = W/v_W$. In this step only non-empty voxels are processed, what reduces drastically the computational power needed in this pipeline, since typically the most part of voxels in a grid are empty.

The configuration of the before mentioned grid generated encapsulating available 3D points is as follows for the both tasks required: detection of vehicles and pedestrians.

- Vehicle detection: The point cloud range considered is $[-4, 2] \times [-40, 40] \times [0, 80]$ meters along Z, Y and X axis respectively. Therefore, the voxel size will be v_D = 0.2, v_H = 0.2 and v_W = 0.2 meters which leads to D' = 30, H' = 400 and W' = 400. For this selection, we took into consideration the point cloud density and distribution of the selected dataset and we followed the steps proposed by [9]. As maximum number of points inside a voxel T, we chose 50. A total of 3 middle convolutional layers was selected.

- Pedestrian detection: The point cloud range considered in this case is $[-4, 2] \times [-20, 20] \times [0, 50]$ meters along Z, Y and X axis respectively. The voxel size will be also v_D = 0.2, v_H = 0.2 and v_W = 0.2 meters and therefore D' = 30, H' = 200 and W' = 1000. Since the detection of these classes will require a bigger number of lidar points in each voxel to have a better perception of the shape, the maximum number of LIDAR points on each voxel in this case was set to 50.

4.3.1 Convolutional neural network

The above mentioned 4D tensor is then fed to a sequence of 3 convolutional layers, batch normalization (BN) and ReLu that aim to provide context information to the shape detection already obtained from each Voxel. One of the breakthroughs of this research is the implementation of this set of CNNs, BN and ReLu layers running on the FPGA.

4.3.2 Region proposal Network (RPN)

Region proposal network (RPN) [46] is an optimized approach for efficient object detection, but it needs data to be dense and organized in a tensor structure like an image which is not the case for LiDAR point clouds. In this work, the typical RPN architecture proposed in [46] had to be adapted together with the above mentioned feature encoding strategy to overcome such limitation. The input of RPN is a feature map generated by the previous CNN and the output is a probability score map and a regression map.

RPN in our case is then formed by three blocks of convolutional layers for downsampling. Each block aims to downsample the input feature map to its half, so that the output of each block has a size of 1/2, 1/4 and 1/8 the size of the input map, respectively. Each output is then upsampled to a fix resolution and concatenated to construct the high resolution feature map. From this then will be extracted the desired probability score map and regression map.

The loss function used for training the presented approach is similar as in [9] and is as follows:

$$Loss = \alpha \frac{1}{N_{positive}} \sum L_{cls}(p_i^{positive}, 1) + \beta \frac{1}{N_{negative}} \sum L_{cls}(p_i^{negative}, 1), \tag{4.1}$$

where $(p_i^{positive}, 1)$ and $(p_i^{negative}, 1)$ are the output for positive anchor and negative anchor respectively. L_{cls} is then the binary cross entropy loss and α and β are positive weights balancing the influence of the positive and negative anchors in the global loss calculation.

4.3.3 Chosen dataset

As mentioned before, the datasets chosen for this work are the known Kitti and Nuscenes [10], which was released in its last version in March 2019 and contains more than 7000 samples of images and point clouds fully annotated. The reason of choosing this last state-of-the-art dataset is mainly because the high quality of its labelling and big availability of synchronized sensors. An overview of the sensor disposition from the Nuscenes dataset is shown in Fig. 4.3. This dataset offers full autonomous vehicle sensor suite composed by 6 cameras, 5 radars and 1 LIDAR. 23 classes and 8 attributes are labelled in each of the 1000 scenes of 20s long each.

However, Nuscenes is based on a 32-channel Lidar when Kitti uses a 64-channel one. This makes that point clouds in the case of the Kitti dataset are more dense and therefore the Voxelnet configuration varies in one case and the other. Another motivation for choosing this dataset for this work is the synchronization assurance between data from different sensors provided by Nuscenes. The data synchronization between sensors it crucial for any image processing methodology that takes samples from different sensors. Being able to match LIDAR point clouds with camera frames taken both at the

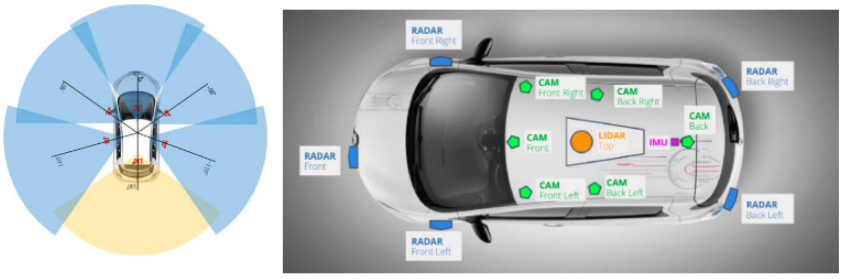


Figure 4.3: Sensor disposition of the used Nuscenes dataset [10].

exact same time, so that a direct matching between LIDAR measurement and object on the image is possible, is highly relevant. In the case of this work, since LIDAR data points were taken for the detection and correspondent images for the visualization this synchronization between data was also an important point. The chosen dataset assures the synchronization of the data of recording time by triggering exposure of a camera when the top lidar sweeps across the center of the camera's FOV, as explained in [10].

4.4 Adaption of our method to FPGAs

Field programmable gate arrays (FPGAs) are low-power devices very suitable for embedded systems. Moreover, an FPGA is able to perform parallel processing and data communications on-chip. Hereby, FPGA are very powerful platforms that can lead up to higher inference speed in DNNs and that, due to their low price, have been widely studied to be used in real automotive applications. Despite these advantages, FPGAs present some constraints that have prevented their use in real scenarios: low memory resources and the lack of trustworthy frameworks that help developers program CNNs and translate them into executable code on FPGAs.

As mentioned above, one of the biggest differences when programming an application that runs on a GPU or on a FPGA is the fact that available memory to handle the multiple meta-parameters and weigths during CNNs training is more constrained in a FPGA than in a GPU or CPU. The proposed algorithm in this chapter achieves good detection accuracy and low inference time when running on an FPGA, getting the best profit of the advantages such platform can offer to DNN applications. The design considerations, results obtained and special development done in order to adapt such DNNs to FPGA architecture will also be shown in this chapter.

As stated in works such as [13], although graphics processing units (GPUs) are more suitable for parallel processing they do need a high power consumption, which could make them a bottle-neck for the integration of deep learning algorithms into vehicles, as they have limited power supply.

In this scenario, FPGA are a low-power consumption option more suitable for embedded applications as they can be programmed as a customized integrated circuit that is able to perform massive parallel processing and data communications on-chip. We believe this is a enough motivation for pursuing a breakthrough in the field of deep learning applications on FPGA, since the usage of this platform

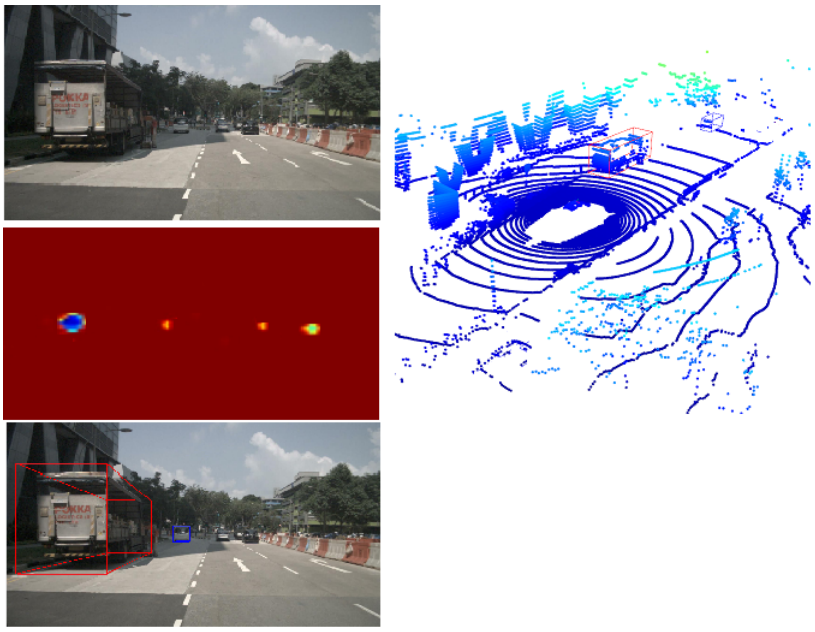


Figure 4.4: Example of the performance of the presented pipeline in this work for vehicle detection. Planar images are only used for visualization but not for testing.

in commercial vehicles is widely extended already (e.g., for image data-stream conversion) and the appearance of frameworks for porting networks to run on FPGA platforms is boosting up their usage in autonomous vehicles against GPUs.

As mentioned in this chapter and defined in [112], one of the breakthroughs of the research presented in this chapter is the implementation of the set of CNNs, BN and ReLu layers running on the FPGA. For that purpose, the legu-up 4.0 [12] framework has been used together with ModelSim HLS suite to convert the implemented layers into readable code by the FPGA.

Making use of the mentioned legup [12] framework (version 4.0) and the Modelsim HLS design Suite for Intel Arria 10 FPGA the porting from the tensorflow source code of the convolutional, BN and ReLu layers to translated code readable by the FPGA was performed. Nevertheless, for making the best use of the HW resources of the platform and due to memory limitations, the hyper-parameters of the network running on the FPGA were optimized with the q-factor explained in Section 4.4.2.

The CNNs implemented in this architecture were modified and adapted to the application presented since they are running on a FPGA Hardware.

These platforms have some promising improvements in machine learning and deep learning like being able to speed up heavy computations, however it has some implications that need to be considered during implementation phase. One of the biggest differences when programming an application that runs on a GPU or on a FPGA is the fact that available memory to handle the multiple meta-parameters and weigths during CNNs training is more constrained in a FPGA than in a GPU or CPU. Therefore, one of the breakthroughs of this work is the implementation of a simulated quantization step needed to run the training and validation on an FPGA based on a a similar approach as the one presented in [13]. As

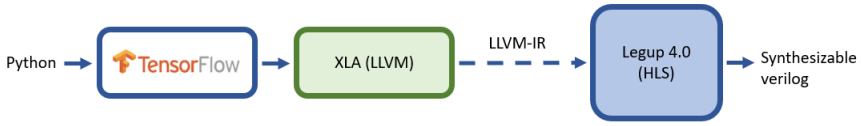


Figure 4.5: Illustration of the open-source framework used in this research to generate hardware code to be executed on a FPGA [11, 12].

explained there, when using CPU or GPU floating-point operation are used, which create gradients during training.

This approach for converting floating point data to fixed point shall be designed carefully since this conversion could lead to a considerable accuracy loss, due to the rounding of big variables with several decimals to less bit consuming integer variables. This step of the presented pipeline is defined in Section 4.4.2.

4.4.1 Convert python-code into FPGA-friendly commands

FPGAs are powerful platform for parallel computing and with low price in market, therefore these are suitable devices to be used in industries such as the automotive with mass production, in which component price is desired in order to maximize the benefits. FPGAs can provide real-time performance with much lower power consumption than Graphic Processing Units (GPU). However, these devices have some constraints, like the difficulty of finding experts who can implement optimized C code to map the designed neural network to a hardware implementation. This is normally done using a high-level synthesis tool, or writing Register-Transfer Level (RTL) code and compiling, [11]. This is a hard and time-consuming task. To avoid it, in this work the usage of an open-source framework is proposed, through which we can translate implementation done with Tensorflow into HW code to be executed on a FPGA plaform. LeFlow [11] enables the conversion of software developed in Tensorflow, Keras or PyTorch into hardware code. It uses Google's Accelerated Linear Algebra (XLA) [4] compiler compiler which outputs LLVM [113, 114] code directly, [115] as intermediate representation (IR), which is fed into the proposed High Level Synthesis Tool legup 4.0 [12] to perform allocation, scheduling, and binding to generate a hardware description in Verilog. In Fig. 4.5 is the workflow of this framework illustrated.

A typical bottleneck in any parallel implementations is the memory handling system. In the used HLS LegUp 4.0, each array in the C code is mapped into dual-port RAMs. This constraint can then be overcome through memory partitioning at the LLVM-IR level enabling this pass at the python-development phase, since FPGAs contain a vast number of independently accessible memories [11].

As proposed in [13], in order to properly process the information along the boundaries, zero padding shall be applied to each feature map produced by each convolution layer. Pixels of a feature map are written to the corresponding address locations in the feature map buffer. Figure 4.6 shows this memory allocation strategy. The feature map buffer where zero padding is applied feeds the Convolutional layer and is fed by the ReLU. The RAM functions as the feature map buffer.

4.4.2 Data quantization: preparing the data for FPGA

As commented in Section 4.4, FPGAs have some constraints that shall be addressed during the implementation of the network. Taking these design constraints into consideration, one of the most important processing steps that was included in this work was the quantization of the data to port it from floating point to fixed-point so that it can be processed by the FPGA. Fixed-point variables, weights and operations are normally used in some platforms because they have no native libraries for floating-point usage and because these normally have less memory resources such as FPGA.

Floating point operations require big amount of memory since every value requires normally between 32 and 64 bits each. That is some times not an option on platforms with no GPU and that is the main reason why these values are normally re-scaled to be stored in smaller data types. It is also for that reason, that scaling and the chosen precision are very important. Errors due to a bad porting from floating point to fixed point can be very critical and make a re-projection step to project a point out of an image or an algorithm not converge. GPU platforms generally use floating-point operations that can generate continuous gradients in the training.

To solve the commented problem on FPGA platform this works proposes an implementation for a porting to fixed point from weights and gradients with the following approach.

First the training and validation phase has to be set on the GPU platform. After this first step, a short software to go through all available variables and weights and analyze their data type and possible values during the execution was developed. Like this, we can predict the variables that will overflow if the fixed-point conversion is done and they have to be re-scaled. An adaptive calculation of some factors determining how many bits will be dedicated to integer part and how many for the fractional part has been developed for this purpose. For the candidate variables to suffer from overflow at some point of the training that we the outcome of the mentioned first analysis these factor will change at the moment that a bit overflow is predicted.

If a weight or variable is defined as a factor 10.21, e.g., meaning 10 bits for integer and 21 bits for fractional part, this bit arrangement can vary if during training one possible overflow is detected. This detection is done via some little memory reserved for internal diagnosis (some number of bits being utilized on execution time for integer and fractional part to be used by the localized sensible variables to suffer overflow). This for sure affected the available resources during training but enabled us the possibility of dynamically change these factors.

For the utilized hardware the maximum number of weights and variables to be observed and analyzed during training for this purpose was 4000.

To evaluate the proposed method for factorizing and optimizing the weights and variables, we performed the training of the network with the KITTI [20] dataset for several bit amounts, as shown in Table 4.3.

4.4.3 Convolutional block: preparing data for FPGA

In the method presented in this work a convolutional block similar than the one proposed by [13] has been implemented. One of the main issues of the deep neural network implementation is the so-called

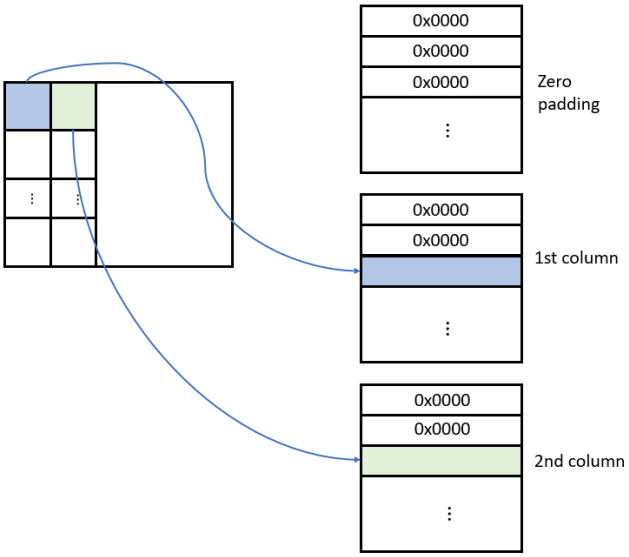


Figure 4.6: Representation of the zero-padding approach applied on feature maps generated by the CNN, similar as proposed in [13].

Name	F1 (%)	AP(%)
no quantization	94.05	88.29
quantization with 12 bits	88.25	79.24
quantization with 16 bits	90.59	84.24
quantization with 18 bits	90.81	86.01
quantization with 24 bits	94.07	92.03
quantization with 32 bits	94.66	88.5

Table 4.1: Table representing the F1 score (F1) and average precision (AP) for different configuration of q-factors for the data quantization step, Section 4.4.2.

vanishing gradient, which means that the gradient of the error used in the back-propagation during training to update the weights gets smaller and smaller on each layer. That leads to the fact that, the deeper the network is, the smaller the gradient would get on each step and therefore the longer the weight update will take. By re-circulating the input in the output, similar as proposed in the ResNet [42], the mentioned behavior could be avoided. In equation 4.2 the formula for weight update is presented, in which η is the learning rate.

$$w_i^+ = w_i + \eta * \frac{dError}{dw_i}. \tag{4.2}$$

To avoid such behavior, a convolutional block based on the good results shown by the [13] proposal has been implemented. As stated there in [13], our proposed convolutional block is based on three paths. One is a direct copy of the input, other with a 3×3 convolutional layer to encode local features and last

	Car			Pedestrian		
	Easy	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
VeloFCN [100]	15.20	13.66	15.98	N/A	N/A	N/A
MV (BV+FV) [116]	71.19	56.60	55.30	N/A	N/A	N/A
VoxelNet [9]	81.97	65.46	62.85	57.86	53.42	48.87
Ours (Kitti)	81.82	65.11	61.96	56.89	53.01	47.75
Ours (Nuscenes)	69.24	43.36	41.76	54.44	51.03	44.48

Table 4.2: Table comparing the performance in 3D detection of the proposed method for 3 levels of occlusion, hard (until 60 % of the object is visible), moderate (80% visible) and easy (fully visible).These results proof that with the proposed method, similar results are obtained when using the Kitti dataset running on a FPGA as in the VoxelNet execution on GPU.

Name	F1 (%)	AP(%)	Runtime(ms)
Chipnet [13]	94.05	88.29	17.59
Fused CRF [117]	88.25	79.24	2000
Mixed CRF [118]	90.59	84.24	6000
Hybrid CRF [119]	90.81	86.01	1500
LoDNN [120]	94.07	92.03	18
Ours (Kitti)	94.66	88.5	18
Ours (Nuscenes)	87.25	79.54	18

Table 4.3: Table representing the F1 score (F1), average precision (AP) and complete runtime execution of the complete pipeline when doing inference on the test set of 2000 samples.

one is a dilated 3×3 convolutional layer [62] to compute features in further positions, but takes less parameters. Adding these three paths a block equivalent to a 5×5 convolutional is obtained but with fewer parameters, which is helpful to avoid the mentioned vanishing gradient effect.

4.5 Evaluation and experimental Results

In this section, the experimental results on different datasets are presented. As it can be seen in Table 4.2 and 4.3 the presented methodology achieves comparable results with other state-of-the-art approaches in terms of accuracy for vehicle detection with a lower runtime execution (Table 4.3). Results with Nuscenes dataset seem to be a bit less accurate than other methodologies, but still acceptable when considering that Lidar sensor has 32-channels and therefore, point clouds have less point density for the classes to be detected.

The visualization of the steps of the presented approach with Nuscenes and KITTI dataset respectively can be seen in Fig. 4.4. In these visualizations the intermediate heatmaps obtained during training are presented together with a projection of the calculated 3D bounding boxes around the detected vehicles.

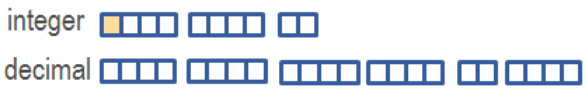


Figure 4.7: Example of implementation of a parameter with 22 bits for the decimal part, 9 bits for the integer a 1 bit for the sign (marked in orange).

These heatmaps calculated in the output of the FC layer before the 3 CNN-BN-ReLu phases show the 2D position of the 3D candidates to be a vehicle projected on a 2D space.

The loss function used for training the presented approach is similar as in [9] and is as follows:

$$Loss = \alpha \frac{1}{N_{positive}} \sum L_{cls}(p_i^{positive}, 1) + \beta \frac{1}{N_{negative}} \sum L_{cls}(p_i^{negative}, 1), \tag{4.3}$$

where $(p_i^{positive}, 1)$ and $(p_i^{negative}, 1)$ are the output for positive anchor and negative anchor respectively. L_{cls} is then the binary cross entropy loss and α and β are positive weights balancing the influence of the positive and negative anchors in the global loss calculation.

The training of this pipeline was done using a training set of 6000 LIDAR sweeps from the Nuscenes dataset and around 3700 samples of KITTI dataset. The validation set is formed by 2000 LIDAR sweeps in Nuscenes and around 3500 in KITTI dataset. The hardware used for the training was 2 NVIDIA GTX 1080 and the FPGA model used for the inference of the trained network is a Arria 10 Intel FPGA with the modelsim-altera software for FPGA development also from Intel.

5

E-DNAS: Differentiable Neural Architecture Search for Embedded Systems

5.1 Introduction

In this chapter it will be presented a method for automatic neural network design, train and validation to be executed on memory constrained platforms. One of the biggest constraints that Deep Learning has nowadays is the complexity of this technology. It is powerful and can mean a whole revolution when applied to the automation of processes in a company, for instance, but it is still not accessible to everyone, due mainly to the need of experts that can design or perform hyperparameter tuning in a way that DNN can be applied smartly to a concrete field. To overcome such limitation, following it is presented the implementation of a Differentiable Neural Architecture Search (DNAS) method able to design light and accurate DNN architecture to perform classification and detection tasks. It will be proofed in this chapter how the found architecture can obtain even better accuracy results that other known methods with less number of layers and hyperparamters.

Designing light Deep Neural Networks (DNNs) and doing it in an efficient manner are two of the main challenges faced in industries like the automotive, which typically need to deal with resource-constrained platforms. This has been addressed in recent works, like SqueezeNet [31] or MNet [36], focused on optimizing the design of neural networks to alleviate their computational cost without losing performance. Most these studies, however, are based on the optimization of "indirect metrics", such as the number of Multiply-ACcumulate operations (MACs) or the number of architecture parameters, which might not be good approximations to the "direct metrics" like energy consumption or latency. As discussed in [37,121], the relationship between these direct and indirect metrics can be highly non-linear and platform-dependent. Another drawback of [31] and [36] is that they require manual approaches and prior expertise, limiting thus their applicability and design efficiency.

The design method has been automatized by the so-called Neural Architecture Search (NAS) [18,33, 59] approaches. These techniques aim to automatically design light and accurate DNNs by optimizing over a search space defined by all possible operations of the target architecture. This optimization is

carried on using either reinforcement learning [18, 33] or evolutionary computing [59].

While NAS-based approaches provide state-of-the-art results in classification tasks for small datasets like CIFAR, they are very computationally and time demanding. There have been attempts to speed up the search process using weight prediction techniques or weight sharing across multiple architectures [122]. Unfortunately, the improvement is still far from providing solutions that can scale to large datasets like ImageNet due to the prohibitive time and resources required.

In this Section we introduce E-DNAS, a differentiable NAS approach that optimizes the direct metrics of an embedded platform, yielding accurate and low-latency DNNs that can be deployed in memory-constrained platforms. The presented research builds upon three main ideas. First, we apply a depth-aware convolution over the input image to compute high-resolution feature maps. Second, we propose a parallel architecture search pipeline that operates on these feature maps and learns the optimal size and parameters of the convolution kernels. This optimization process is ruled by a multi-objective differentiable loss function that combines classification accuracy and minimal latency, a direct metric. And third, we boost the architecture search velocity through a novel block that connects the learned meta-kernels during training. This block is shown in Figure 5.1 and aims to update the learned meta-kernel (from *feature map 1*) on each iteration with the result of the weighted sum of that kernel and a second one being learned in parallel (the one from *feature map 2*). We show that this training information exchange on each iteration speeds up the search for the optimal kernels.

We demonstrate remarkable results in terms of search-time and classification accuracy compared to other state-of-the-art NAS methods and comparable to other recent breakthroughs like [38] or [22], which are more oriented for mobile devices rather than to be integrated into embedded systems, that typically have less flexible architectures.

Furthermore, E-DNAS is completed with an improvement in its development that make it suitable not only to solve the image classification task but also the object detection. We name this next version of the E-DNAS "Stacked-NAS". It is also presented in this Chapter since it bases on the above-mentioned E-DNAS and therefore share some parts of its pipeline.

5.2 Related Work

Deep Learning is revolutionizing many technological areas, but it has some important constraints or limitations that need to be overcome in order to obtain its full potential. Some of these are the large amount of hardware resources (e.g., memory) needed to run some deep learning applications and also the manual network and parameter configuration traditionally done by experts to obtain an optimal DNN for a particular application.

To this end, several recent papers have published methods to reduce the network size by pruning weights from DNNs, like [34] or [35]. Despite the good results these works demonstrate, manually selecting the redundant weights is not always effective when designing architectures to be executed on embedded platforms. Based on a similar idea, the so called NAS (neural architecture search) approaches have recently been published, which can evolve over time based on some feedback to obtain the optimal numbers and types of layers. These achieved better performance than hand-crafted models by automating

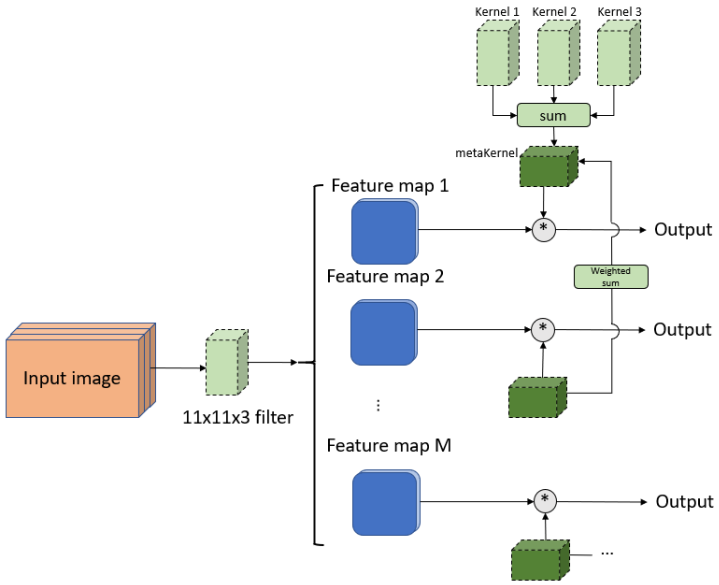


Figure 5.1: **General overview of E-DNAS.** Our approach has two main building blocks: a depth-aware convolution with a high resolution 11×11 kernel followed by pairwise learning of meta-kernels with loopy flow of information on each iteration between training paths.

the architecture design.

As mentioned above, some NAS approaches like [18, 33, 58] apply reinforcement learning for finding the best neural architecture. These approaches propose a reward based strategy to update the controller that selects the candidates on each iteration.

These approaches showed good and accurate results, however they still require a long time to find the proper architecture.

Lately, a faster version of the NAS has appeared, which can get to an optimal network design quicker by using gradient-based optimizations, like DARTS [38]. The differentiable neural architectural search (DNAS) propose to relax the search space to be continuous so that the architecture can be optimized with respect to its validation set through gradient descent. These techniques achieve a big efficiency improvement reducing drastically the cost of architecture finding in comparison to the non-differentiable approaches (NAS).

State-of-the-art works such as [22] or [23] have also exploited a similar approach as the one proposed in this work, making use of the additive property of the convolution to merge the searched operations and reduce the number of parameters in the DNN architecture. The main contribution of this work is the reduction of the search time through the self-designed *feedback block* defined in Section 5.3.1. Moreover, this work extends the DNAS method not only to general-purpose computing platforms like mobile devices but also to embedded platforms such as DSP, which, as above commented, are more restrictive and less flexible and are designed for single pre-defined functions.

5.3 Method

In this work we propose a methodology for automatic neural architecture design to be executed on embedded platforms. We demonstrate state-of-the-art results on image classification, as presented in Table 5.1. We present a DNAS approach that aims to find optimal neural architecture with low latency to be executed on memory-constrained system on chips (SoCs), such as the one used for the experiments in this work.

The presented pipeline has two main steps:

- High resolution feature extraction through depthwise convolution using big dimension convolutional kernels.

- Pairwise neural architecture cross-search for the calculated feature maps on previous step.

In this work we regard network MACs and FLOPs as the proxy of the computation consumption, [123].

5.3.1 Formulation

Convolutional filters

As it was demonstrated in AlexNet [28], each convolutional kernel is responsible to capture a local image pattern. The larger the convolutional kernel is, the higher resolution patterns it tends to detect at the cost of more parameters and computations.

In particular, there is an important idea proposed in MixConv work [23] that we exploit here and consists in having multiple kernels with different sizes in a single convolution operation to allow the network to capture different types of features from the input images. Based on this, we present a two-step pipeline in which: first a large convolutional kernel is applied over the input image to capture high resolution patterns, and second several learnable kernels with different sizes are applied on the calculated feature maps to learn different of patterns on the input data.

In order to reduce the number of operations and, hence, the resulting network size, there are two considerations that shall be taken into account:

- The first step proposed in this work suggests a separable *depthwise* convolution with a 11×11 kernel applied on the input image that leads to a reduced parameter size and computational cost, compared to the traditional convolution operation, [16, 23, 36, 124].

- The filters applied on resulting feature maps after a first 11×11 convolution are a sum of 3×3, 5×5 and 7×7 filters. This work exploits the additivity property of convolution: if several 2D kernels with compatible sizes operate on the same input with the same stride to produce outputs of the same resolution, and their outputs are summed up, these kernels are finally added on the corresponding position to obtain the equivalent filter which will produce the same output [61].

The main difference between the traditional convolution operation and the mentioned separable depthwise convolution over an input image (or tensor) is the number of steps in which this operation is

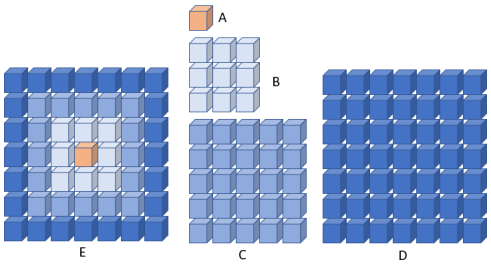


Figure 5.2: Example of a summed convolutional kernel (E), resulting of summing 1×1 kernel (A), 3×3 (B), 5×5 (C) and 7×7 kernel (D).

applied.

In this context, the additivity property is applicable because the sizes of the filter or kernels are compatible, which means, smaller ones can be "contained" in bigger ones (with same center). That is:

$$\boldsymbol{I} * \boldsymbol{K_1} + \boldsymbol{I} * \boldsymbol{K_2} = \boldsymbol{I} * (\boldsymbol{K_1} \oplus \boldsymbol{K_2}), \tag{5.1}$$

where $\boldsymbol{I}$ is the input feature map, $\boldsymbol{K_1}$ and $\boldsymbol{K_2}$ are two 2D kernels with compatible sizes, and $\oplus$ is the element-wise addition of the kernel or filter parameters on the corresponding positions, [22, 61].

The application of the additivity property is also valid for the following batch normalization (BN), as shown in Eq. (5.2) so that each single BN applied after each convolution from Eq. (5.1) produces the same output, as the summation of each single convolution and BN with added bias [61]:

$$\boldsymbol{O} = \boldsymbol{I} * (\frac{\gamma_1}{\sigma_1}\boldsymbol{K_{3\times 3}} \oplus \frac{\gamma_2}{\sigma_2}\boldsymbol{K_{3\times 1}} \oplus \frac{\gamma_3}{\sigma_3}\boldsymbol{K_{1\times 3}}) + b, \tag{5.2}$$

where $\boldsymbol{O}$ represents the output feature map, $\boldsymbol{I}$ is the input data or feature map generated by the previous layer, σ is batch standard deviation and γ and b are the BN parameters to be learned. The input I may need to be appropriately padded depending on the resolutions.

Feedback-block

One of the contributions of this work is the addition of one *feedback-block* into the training pipeline of each feature map to update the learned convolutional filters or kernels on each iteration, see Figure 5.2. The implementation of this *feedback-block* is just the weighted sum of the learned meta-kernels being trained in parallel:

$$\boldsymbol{K_1'} = \boldsymbol{K_2'} = \beta_1 * \boldsymbol{K_1} + \beta_2 * \boldsymbol{K_2}, \tag{5.3}$$

where $\boldsymbol{K_1'}$ and $\boldsymbol{K_2'}$ are the two meta-kernels candidates being learned, $\boldsymbol{K_1}$ and $\boldsymbol{K_2}$ are the kernels before the update and β_1 and β_2 are the weights for these kernels. These weights are calculated according to the loss calculated on each training "path".

We next compute the weights, in such a way that they will be close to one for small losses in the

forward pass:

$$\begin{aligned} \beta_1 &= \frac{\tanh \frac{1}{L_1}}{\tanh \frac{1}{L_1} + \tanh \frac{1}{L_2}}, \\ \beta_2 &= \frac{\tanh \frac{1}{L_2}}{\tanh \frac{1}{L_1} + \tanh \frac{1}{L_2}}, \\ & with \;\; \beta_1 + \beta_2 = 1. \end{aligned} \tag{5.4}$$

where L_1 and L_2 represent the value of the loss function on two parallel network candidates being searched (illustrated in Figure 5.1). Through this implementation, on each iteration the closer kernel to the "expected" one has more influence.

After training of each image, all learned kernels are encoded into one, following a similar approach as detailed in Eq. 5.4:

$$\begin{aligned} \boldsymbol{K} &= \sum_{j=1}^{j=N} \Gamma_j * \boldsymbol{K_j}, \\ & with \;\; \Gamma_j = \frac{\beta_j}{\sum_i \beta_i}. \end{aligned} \tag{5.5}$$

5.3.2 Search space

Following [38], we define the search space of each output $x^{(j)}$ in Eq. (5.6) (e.g., feature map in convolutional networks) as the combination of operations $o^{(i,j)}$ applied on inputs $x^{(i)}$, assuming the inputs as the outputs of the previous two layers:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}). \tag{5.6}$$

The gradient-based NAS methodologies [38, 60] relax the categorical choice to a softmax to make it continuous. Let $\boldsymbol{O}$ be a set of candidate operations (e.g., max pooling, convolutions) where each operation represents some function $o(-)$ to be applied on the input $x^{(i)}$, a particular operation can be represented as:

$$\overline{o}^{(i,j)}(x) = \sum_{o \in O} \frac{exp({\alpha_o}^{(i,j)})}{\sum_{o' \in O} exp({\alpha_{o'}}^{(i,j)})} o(x). \tag{5.7}$$

As demonstrated in [38], the task of architecture search reduces to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$.

The relaxation of the categorical choice presented in Eq. (5.7) can also be defined as follows, using the additivity property of convolutions. Based on this, each operation can be calculated as $\boldsymbol{I} * \boldsymbol{K}$, where $\boldsymbol{I}$ is the input of the operation and $\boldsymbol{K^{(i)}}$ is the kernel to be learned:

$$o(x) = \boldsymbol{I} * \boldsymbol{K^{(i)}}. \tag{5.8}$$

$$\overline{o}^{(\mathrm{i,j})}(x) = \sum_{o \in O} \frac{exp({\alpha_o}^{(\mathrm{i,j})})}{\sum_{o' \in O} exp({\alpha_{o'}}^{(\mathrm{i,j})})} (\boldsymbol{I} * \boldsymbol{K}^{(\boldsymbol{i})}). \tag{5.9}$$

After relaxation of the search space, the proposed search network algorithm aims to learn jointly the architecture α and the weights w.

5.3.3 Multi-objective loss function

Based on the Eqs. (5.8) and (5.9), the goal of the presented DNAS approach shall be calculating the weights w that minimize the validation loss:

$$min_{\alpha} = L_{val}(w(\alpha), \alpha). \tag{5.10}$$

These weights w are learned during forward- and backward-pass and represent typical connection weights but include also the β values defined in Eq. (5.3) and Eq. (5.4), which aim to update the kernel's value after each training iteration to speed up the search stage. In order to let this method generate models adaptively depending on the target embedded platform, we propose to include one more term to the global loss function to be minimized during training that attends to the latency of the network candidate.

The proposed loss function in Eq. (5.10) has a term to observe the latency of the proposed architecture. As demonstrated in different works such as [37] or [15], extracting indirect metrics like MACs or number of weights might not be good proxies for the resource consumption of a network since networks with fewer number of MACs can be slower when executed on embedded targets:

$$L_{LAT}(\alpha) = \sum_{l} LAT(b_l)^{(\alpha)}, \tag{5.11}$$

where $b_l^{(\alpha)}$ denotes the block at layer-l from the network architecture candidate α, [15].

In the proposed implementation, we use a latency look up table model to estimate the global latency (Eq. (5.11)) of the network candidate based on the runtime of each operator, similar as proposed in [15]. The latency lookup table has been created by checking the runtime of multiple operators on the target platform.

5.3.4 The search algorithm

The formulation of the network search problem that is solved through the proposed method can be expressed as follows:

$$\begin{gathered} \max_{a_i} Accuracy(a_i) \\ \text{constrained by } LAT(a_i) <= Budget, a = 1, ..., N. \end{gathered} \tag{5.12}$$

where a_i is the sampled network from the search space, $LAT(a_i)$ is the latency on the platform of the sampled network and *Budget* is the predefined latency budget.

The problem presented in Eq. 5.12 is solved iteratively by the presented method by minimizing on

each iteration the following loss function:

$$L(a, w_a) = CE(a, w_a) + \beta L_{LAT(a)}, \tag{5.13}$$

where $CE(a, w_a)$ is the cross-entropy loss of the network candidate a with weights w_a and $LAT(a)$ is the measured latency of network candidate a in microseconds, [15]. Typical NAS approaches like [16, 18] or [33] are based on the iteratively training of sampled architectures candidates from the search space on a small proxy dataset through some epochs to be then transferred to the target dataset after training. In the end, the objective of these NAS methodologies is finding the network weights w and optimizing the network candidate $a \in A$ (being A the search space), same as in the presented work but the needed resources and time to train thousand of network architectures before reaching the optimal solution make them some times infeasible.

Motivated by this problem, DNAS like [15, 38, 60] or [22] have become more popular lately. In this research we adopt a different paradigm of solving the same problem based on DNAS.

In the presented work we relax the categorical choice of a particular filter or kernel in the target architecture by formulating the sampling process in the search stage, similar as proposed in [125] and [15], and define the probability of sampling the i-th kernel candidate $\boldsymbol{K^i}$.

$$P(\overline{\boldsymbol{K}} == \boldsymbol{K^i}) = softmax(\alpha^{(i)}) = \frac{exp(\alpha^{(i)})}{\sum_{j=0}^{N} exp(\alpha^{(i)}))}, \tag{5.14}$$

where $\overline{\boldsymbol{K}}$ is the sampled kernel during the search stage. Following this we reformulate the equation 5.9 and focus on making the Eq. 5.14 differentiable so that the loss function 5.13 can be optimized through stochastic gradient descent (SGD) approach [15, 126, 127].

The objective function in Eq. (5.14) is already differentiable with respect to the weigths of the kernels but not to the architecture parameters α due to the sampling process. To solve this, we follow a similar approach as in NAS related works [16, 60, 125, 128]. We adopt the Gumbel Softmax function [129] to rewrite the equation 5.14:

$$P(\overline{\boldsymbol{K}} == \boldsymbol{K^i}) = \frac{exp((log(\theta_i) + g_i)/\tau)}{\sum_{j=0}^{N} exp^{(}(log(\theta_i) + g_i)/\tau)}, \tag{5.15}$$

where $g_i \sim$ Gumbel(0,1) represents a white noise function that follows the Gumbel distribution between zero and one, τ represents the temperature parameter of the Gumbel Softmax function, [129], which makes the discrete sampling probability function in Eq. (5.14) become continuous as τ approximates to one. Lastly θ_i represents the class probabilities calculated in Eq. (5.14), [22].

Once the loss function is differentiable Eq. (5.15), the SGD method to optimize function (5.13) is applied, so that on each iteration the network architecture weights w_i and probability parameters α are updated based on the partial derivative of the loss function with respect to w and α, respectively.

The search process is now equivalent to training the stochastic network after generating all kernel candidates from the 11×11 meta kernel, similar to [22]. During training, the value of the loss function $L(a, w_a)$ in Eq. (5.13) is calculated. Together with it, $\partial L/\partial w_a$ and $\partial L/\partial \alpha$ are computed to update

weights and architecture probability parameters on each iteration. Through this, we train each operator's weight and update the sampling probability for each operator, respectively, so that when training finishes, we can obtain the optimal network architectures with the best kernels from the learned α parameters.

As will be shown in the experiment section, the proposed approach works faster than RL and typical NAS methodologies and provides acceptable results for high resolution input images.

Algorithm 5.1: The search architecture methodology

Result: Find weights w_i and architecture probability parameters α to optimize the global loss function (5.13), given a defined search space with a combination of operations $\overline{o}^{(i,j)}$, defined in Eq. (5.9), a latency budget and an input dataset.

random initialization of α parameters

while *not converge* **do**

Similar to [22], we generate the kernel candidates. Calculate Loss through Eq. (5.13). Calculate $\partial L/\partial w_a$ and $\partial L/\partial \alpha$. Update weights and architecture probability parameters α. Update Kernels using Eq. (5.3) and Eq. (5.5).

end

Extract more optimal architecture from learned α parameters.

5.4 Stacked-NAS Method

In this section we introduce Stacked-NAS, which is an improvement of the defined algorithm in Section 5.3 that aims not only to classify images but also identify and detect objects within them. The Stacked-NAS is a two-step method to automatically design efficient neural network architectures which can accurately detect objects on the input images. It bases on a first architecture search of an encoding-decoding convolutional architecture that aims to extract the ROI (region of interest) around the object candidates on the input image. This initial step is followed by a second and final phase that classifies the extracted candidates between the labelled classes. This second step has been described above in Section 5.3.

The proposed method provides not only accurate architectures but also light, so that they can be easily ported to embedded devices. Stacked-NAS, illustrated in Figure 5.3, computes the optimal size of a number of kernels that captures patterns of the input data at different resolutions, obtaining feature-density maps that finally define the ROIs with the object candidates, as defined in [130]. We base our search strategy on separable and depthwise convolutions, which can obtain the same results as a normal convolutional operation but with less number of parameters and leverage on the additive property of convolution operations to merge several kernels with different compatible sizes into a single one, reducing thus the number of operations and the time required to estimate the optimal configuration. We evaluate our approach on several datasets to perform detection where Stacked-NAS is able to find optimal designs faster that other methods.

The pipeline we propose has two main steps:

- Object candidate extraction from a encoding-decoding architecture able to detect features at different resolution on the input data.

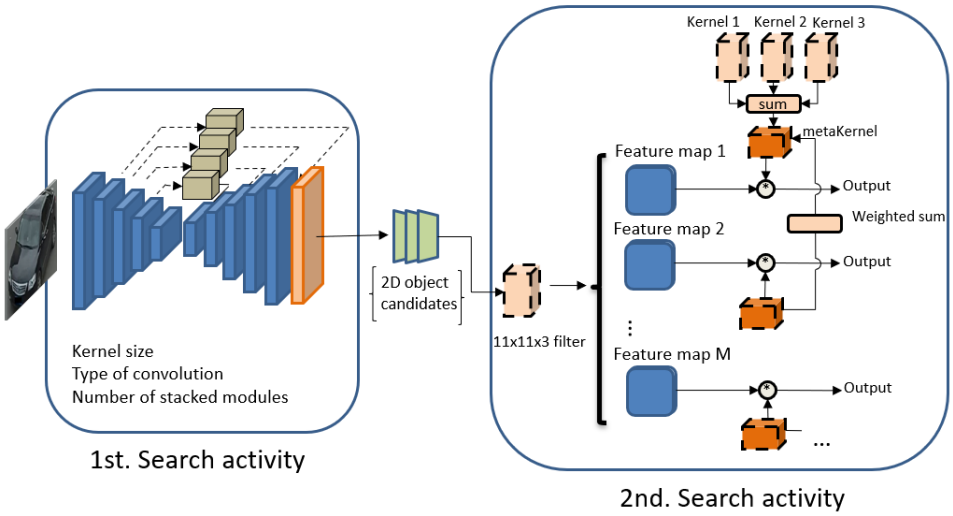


Figure 5.3: **General overview of Stacked-NAS.** Our approach has two steps: a multi-resolution feature extraction to create the object candidates and a second DNAS to find an architecture that classifies the extracted candidates in one of the labeled classes.

- DNAS approach to create a light and accurate network able to classify the detected objects on the previous steps, regarding network latency as the proxy of the computation consumption.

5.4.1 Object candidates extraction

In this section we will define the implementation details taken into consideration to develop the first step of the Stacked-NAS.

Problem formulation

As above mentioned, each convolutional kernel is responsible to capture a local image pattern. The larger the convolutional kernel is, the higher resolution patterns it can detect.

In order to detect objects on large datasets with large image resolutions and several objects on each image, we first need to define a method to extract candidates to be objects over the image, to later on, classify them. This is the objective of this first step.

On this regard, there is an important idea proposed by Stacked-Hourglass [53, 131], which presents an encoding-decoding architecture to detect multi-scale features with pixel-wise resolution, applied to find the joints of humans to, later on, predict their 3D pose. This work bases on this idea to propose an object-candidate extractor as first step of its pipeline. Our network is formed by convolutional and max pooling layers, used to process features down to low resolution (64×64 pixels in our case). At each max pooling step, more convolutions are applied until the lowest resolution that can be achieved. On that moment, the network begins the top-down sequence of upsampling and combination of features across scales to increase the precision of the up-scaling.

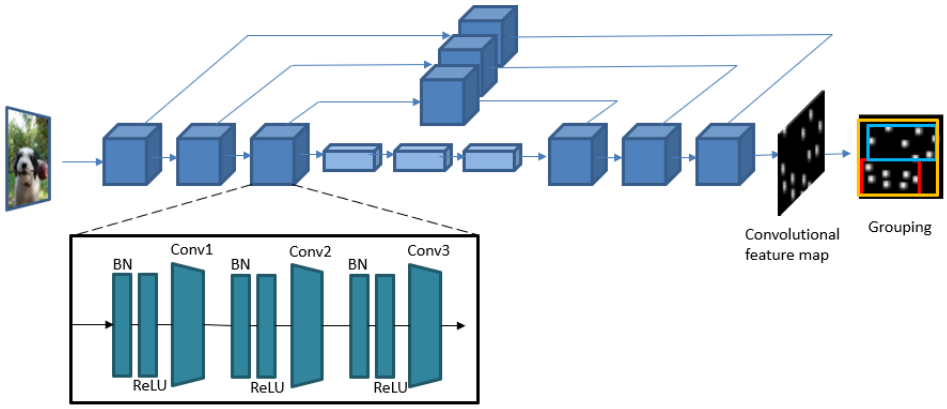


Figure 5.4: Illustration of step one of the proposed pipeline which aims to extract high feature-density areas as object candidates (marked in red). On each block, the type of convolution and the kernel sizes are parameters to be learned. During search, with each network candidate, a set of groups is calculated. Non-maximum suppression [14] is applied to minimize overlapping.

As proposed in [53,131] and in [74], we perform nearest neighbor upsampling of the lower resolution followed by an elementwise addition of the two sets of features. This architecture is, therefore, symmetric, so for every layer present on the way down there is a corresponding layer going up, [53]. As shown in Algorithm 5.2, the type of convolution and the kernel sizes are to be learned during search time. The target architecture shall find these parameters in order to get the group of features on the output feature map that has a higher IOU (intersection over union) with the labeled object, the less number of parameters and requires less time. These three factors are extracted and compared during search in order to choose the best network architecture.

The search space in Stacked-NAS

In this work we introduce a function $f : I \rightarrow M$ that converts input tensor with size $(W \times H \times C)$ into a convolutional feature map (**M**). As mentioned above, the target architecture is a combination of operations $g^{i,j}$ applied on the input m^i as:

$$m^{(j)} = \sum_{i<j} g^{(i,j)}(m^{(i)}). \tag{5.16}$$

In order to reduce the search time we constrain the search space so that the searched architecture has an skeleton as illustrated in Figure 5.4, that means, it is formed by a recurrent block with a similar layer disposition:

$$m^{(j)} = N * \sum_{i<j} \overline{g}^{(i,j)}(m^{(i)}). \tag{5.17}$$

Here in Eq. (5.17) we show the constrained search space, where N represents the number of times each convolutional block is repeated in the pipeline and $\overline{g}$ is a subset of the whole operation space since we only look for separable, dilated or conventional convolutions. The parameter N_r, the type of operation $\overline{g}$

Algorithm 5.2: The object candidate extraction methodology

Result: Find weights w_i to optimize the global loss function 5.18, given search space formed by a encoder-decoder architecture skeleton (Eq. (5.16)) and 3 types of operation candidates with an unknown kernel size (in order to simplify the search space only odd kernel sizes from 3×3 to 11×11 are possible).

while *not converge* **do**

Select one of the three operation candidates with random kernel sizes (always inside the pre-defined sizes). Create first architecture candidate by filling the pre-defined skeleton with selected operation and kernel size. Calculate feature map and group the features. Extract IOU between each candidate group and the labeled object and number of main features in the feature map contained by the candidate group. Calculate Loss through Eq. (5.18). Calculate $\partial L / \partial w_a$. Update weights.

end

Extract validation loss on a proxy dataset taken from the input image set, the kernel size for the minimum loss achieved in the execution and search time. These three indicators together with each group's IOU and number of contained features (calculated on each iteration) will be used to decide over the generated architecture candidates.

and each kernel size together with the number of times this encoding-decoding pipeline shall be repeated in a sequence (K_r) are to be learned during search time.

Before the input image enters the first pipeline illustrated in Figure 5.4 there is a preprocessing step formed by a sequence of convolutional and pooling layers that rescale and prepare the input images.

During the search stage of this first pipeline-step, the following parameters will be learned:

- **Number of stacked blocks**: As proposed by [53], the performance of this idea is better when we stack multiple hourglasses (or stacked blocks) end-to-end, feeding the output of one as input into the next. This provides the network with a mechanism for repeated bottom-up, top-down inference allowing for reevaluation of initial estimates and features across the whole image. In our case, the number of stacked blocks will be searched. To limit the search time, we limit this number between 1 and 10, since empirically we identified most efficient designed networks within this interval of stacked blocks.

- **Type of convolution**: Since we look for light and easy-to-port DNNs, during the search phase we aim to minimize the number of parameters. For that reason, during the search stage of this first step, there are three types of operations inside the search space that we consider: dilated, separable and normal convolution. Each of them followed by a max-pooling operation. Through this we aim two things: limit the search space, therefore reducing the search time, and use light convolution operations with minimal number of operations.

- **Number of filters on each Stacked block**: Inside of each encoding-decoding step, the number of filters or convolutional kernels as well as their size and stride will be searched during the search phase. In order to limit this activity, we constraint the search-space by not including kernel sizes

smaller than 3×3 (since we do not require pixel-wise accuracy on our case), bigger than 11×11 (since empirically we identified that this kernel size detects feature at a sufficient resolution for our purpose) and stride between 1 and 3.

The search strategy in Stacked-NAS

We use reinforcement learning during the search process of this method which consists of extracting child networks from the search space and train them in a proxy dataset (subset of training dataset with less training data to make training process faster) and validate it. The accuracies obtained in this validation are then fed back to the controller to improve the sample process over time.

On the first step of the presented pipeline we aim to minimize the loss function in order to find the object candidates:

$$\min Loss = \min_{\substack{m=1,..,M \\ n=1,..,N}} \sum_{i} \|P_{i,m} - C_{i,n}\| + \max IOU(candidate, GT), \tag{5.18}$$

where P is the predicted feature on the image i, C is the center of the m-th object and M and N are total number of predicted features and labeled objects on i-th image, respectively. During search time we aim to maximize the IOU of the candidate and the labeled object so that the outcome of the search network are groups or region proposals that contain as much are from the Ground truth (GT) or labeled object.

Grouping strategy

After the input image of the training dataset has been fed to the sequence of convolutional layers defined above, a feature map is generated from which we will calculate the object proposals (Figure 5.5).

Similar as defined in [46] when the authors introduced the concept of an "anchor" as a square region centered on the points from the feature map, we propose a grouping methodology of the most relevant features based on the generation of squares that fulfill two main premises at the same time:

- The squares shall contain as much main features as possible.

- The aspect ratio $width/height$ of the group or region shall never exceed 1:2 and 2:1, respectively.

Following these premises we look for square regions with a higher feature density since these will be our object proposals. In order to minimize redundancy in the candidates, we apply Non-Maximum Suppression [14] and characterize each group as $\boldsymbol{G_i} = (\boldsymbol{P_{tl}}, \boldsymbol{P_{br}})$, where each component is a tuple (x, y) that indicates the pixel coordinates of the top left and bottom right corner of the square group. For each trained network candidate the value of the defined loss function 5.18 applied on the candidates, their IOU, the network parameters, the search time together with the type of operation and kernel size are extracted in order to compare the performance of each network candidate.

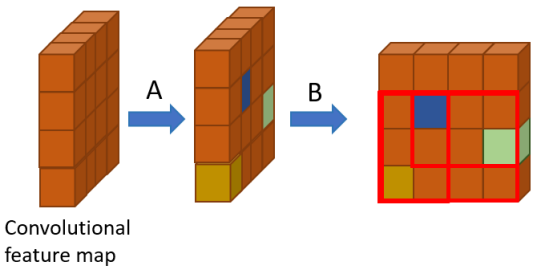


Figure 5.5: During the search time of the first step for each network candidate will be calculated a convolutional feature map predicted as result. From the filtered feature map the most relevant features will be extracted. The group candidates (marked in red) will be then based on them calculated.

5.5 Experiments for image classification

In this section we aim to demonstrate the performance and efficiency of the proposed method comparing the results obtained on different datasets.

We have conducted experiments to proof the accuracy of our work in classifying images on the commonly used ImageNet benchmark [132] and PascalVOC [79] datasets applying several important training techniques, as detailed in Section 5.5.1.

We perform experiments on the widely used ImageNet benchmark and the rest of common datasets mentioned before for a fair comparison with other state-of-the-art methods. We use the normal data augmentation including random horizontal flipping, scaling hue/saturation/brightness, resizing and cropping, following Section 5.5.1. For each experiment we train for a mean of 250 epoch with a batch size of 1024, as suggested in [38] and [22], using similar configurations as proposed in these works. During the architecture-search process, to evaluate each candidate, we randomly initialize the weights, train it from the beginning and check its performance on the test-set. For each candidate, an indicator for the precision, latency, number of parameters and search time are extracted, which will be then used to compare each network searched between each other. We aim to find an architecture with the minimum number of parameters, less inference latency on the hardware and the maximum precision together with a reduced required search time.

These results can be seen in Table 5.1. In this section, a comparison of the proposed method with state-of-the-art is also presented.

5.5.1 Implementation details

We have trained the models using 8 GPU NVIDIA Tesla V100. As proposed in [137] we have applied several implementation tricks to improve the training process, such as the following:

- Randomly sample an image and decode it into 32-bit floating point raw pixel values in [0,255].

- Random crop of a rectangular region.

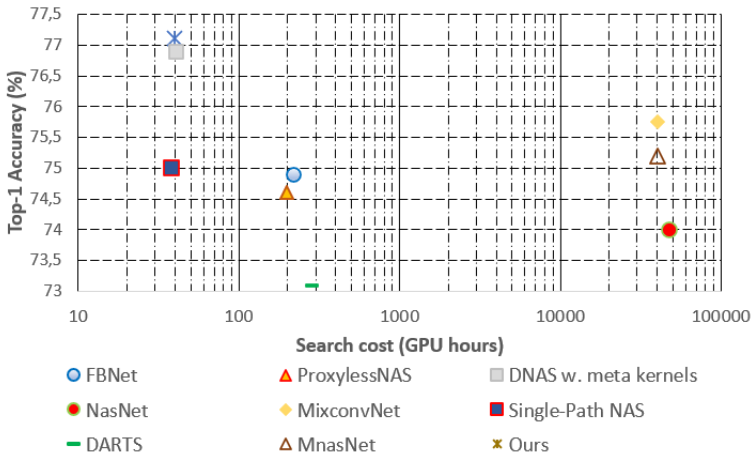


Figure 5.6: Comparison of several NAS and DNAS methods in terms of search cost. The data has been directly obtained from their papers. As also commented in [15], the search cost for MnasNet is estimated according to the description in [16]. The search cost of PNAS [17] is estimated based on the results claimed on that work that their method is eight times faster than NAS [18].

- Horizontal flip with 0.5 probability.

- Normalize RGB channels.

- Scale hue, saturation and brightness coefficients.

Due to the importance of the learning rate in the training process, in the conducted experiments we have applied a learning rate warm up (use small learning rate at the beginning and then switch back to the initial learning rate when training process is stable) followed by decaying cosine learning rate to improve the training process, as commented in [137] and proposed by [138].

In contrast to the typical exponentially decaying learning rate used $l_r = l_{r0} * e^{-Kt}$ in this work we have applied the formula in Eq. (5.19):

$$\begin{aligned} l_r &= \frac{1}{2}(1 + \cos\frac{b\pi}{B})l_{r0} \\ w^* &= w + l_r\frac{\partial L}{\partial w}. \end{aligned} \tag{5.19}$$

where B is the total number of batches, b is the actual batch during training, w^* the updated weight, w the weight before update and l_{r0} is the initial learning rate (in our case 0.65).

5.5.2 Target platform

The embedded platform targeted to check the effectiveness of the proposed method is the TDA2 system on chip which can accelerate deep neural network layers using the C66x DSP cores together with the

Model	Search Method	Search Space	Search Dataset	# Params(M)	FLOPs(M)	acc(%)
MNetV2 [32]	manual	-	-	3.4	300	72.0
CondenseNet(G=C=8) [133]	manual	-	-	4.8	529	73.8
EfficientNet-B0 [134]	manual	-	-	5.3	390	76.3
NASNet-A [18]	RL	cell	CIFAR-10	5.3	564	74.0
PNASNet [17]	SMBO	cell	CIFAR-10	5.1	588	74.2
DARTS [38]	gradient	cell	CIFAR-10	4.7	574	73.3
PDARTS [135]	gradient	cell	CIFAR-10	4.9	557	75.6
GDAS [125]	gradient	cell	CIFAR-10	4.4	497	72.5
MnasNet [16]	RL	stage-wise	ImageNet	3.9	312	75.2
Single-Path NAS [136]	gradient	layer-wise	ImageNet	4.3	365	75.0
ProxylessNAS-R [128]	RL	layer-wise	ImageNet	4.1	320	74.6
ProxylessNAS-G [128]	gradient	layer-wise	ImageNet	-	-	74.2
FBNet [15]	gradient	layer-wise	ImageNet	5.5	375	74.9
MNetV3 Large [24]	RL	layer-wise	ImageNet	5.4	219	75.2
MNetV3 Small [24]	RL	layer-wise	ImageNet	2.9	66	67.4
MixNet [23]	RL	kernel-wise	ImageNet	5.0	360	77.0
MetaKernels [22]	gradient	kernel-wise	ImageNet	7.2	357	77.0
Ours	**gradient**	**parallel kernel-wise**	**ImageNet**	**5.9**	**365**	**76.9**

Table 5.1: ImageNet classification performance compared with other state-of-the-art methods. The proposed approach in this work demonstrates a good Top-1 accuracy with less number of parameters and FLOPs. The number of parameters, FLOPs and Top-1 accuracy metrics presented in this table for the rest of the methodologies have been directly extracted from their respective papers. As it can be seen, the proposed method E-DNAS achieves similar accuracy results compared to other state-of-the-art methods, such as [22] and [23] in less time, as presented in Figure 5.6.

Texas Instruments Deep Learning suite (TIDL) to convert the trained network from floating point to fixed point and to enable the inference of the network on the embedded platform.

The mentioned TDA2 hardware has ARM Cortex-A15 cores running at up to 1.5 gigahertz, a dual-core DSP C66x processors that are capable of running deep learning inference, together with one embedded vision engine subsystem (EVE). It can run 16 times 16-bit (enough resolution for deep learning applications) MAC operations per cycle reaching up to 20.8 GMACs/s, [139].

To check the effectiveness of the proposed method we have mainly attended to MAC and FLOPs in order to compare the results with the theoretical performance of the target platform. To calculate this, based on the hadware specifications mentioned above the SoC has two C66x DSP cores running at 1000 GHz frequenz ($2 * 32GMACs$), other one EVE core running at 900 MHz ($16 * 900MMACs$) and other 2 ARM Cortex A15 cores running at 1500 MHz ($2 * 8 * 1500MMACs$). This results in 105 GMACs as theoretical performance, which means 210 GDLOPs, assuming DLOPs as 8-bit arithmetic or conditional operation (Multiply/Add/Compare). It can be assumed than $1MAC = 2DLOPS$.

For the experiments done on the mentioned TI and presented in Table 5.3, the trained networks using the proposed method in this work and ImageNet benchmark were converted from floating-point to fixed point to be then executed on the DSP dual core of the above mentioned hardware. In the floating-point

Model	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	bike	person	plant	sheep	sofa
MNetV2 [32]	75.8	84.5	83.4	76.1	68.3	58.7	78.9	84.8	86.5	54.4	80.7	70.9	84.0	85.0	83.6	76.8	48.7	78.7	72.8
MNetV3-S [24]	69.3	77.4	76.9	67.0	62.0	43.7	76.3	79.1	82.1	47.2	75.4	65.2	78.4	81.0	79.7	72.5	39.4	68.7	67.1
MNetV3-L	76.7	84.1	84.1	77.0	69.9	75.9	84.8	85.1	88.1	56.3	84.8	64.8	84.3	87.9	84.7	77.9	46.3	80.5	73.9
metaKernel [22]	77.3	86.1	84.8	76.8	68.6	59.2	83.6	86.3	87.1	56.9	85.2	67.2	86.6	87.2	86.0	77.7	49.1	80.8	74.5
Ours	**77.8**	**85.8**	**84.6**	**74.9**	**70.1**	**57.4**	**63.4**	**87.8**	**88.1**	**58.7**	**83.9**	**72.1**	**86.2**	**86.3**	**86.8**	**87.1**	**50.6**	**79.6**	**74.9**

Table 5.2: Comparison of the obtained results on the Pascal VOC2007 test set. As suggested by other works like [22], the VOC2007 trainval and VOC2012 trainval are combined as the training data and the PascalVOC2007 dataset is used as test-set. We demonstrate a better classification accuracy than other similar methods such as [22] or [24].

Model	# Params (M)	MACs (M)	Time (ms)
MNet [36]	4.2	569	75
NasNet-A [18]	5.3	564	183
Ours	**5.9**	**535**	**38**

Table 5.3: Results on ImageNet Benchmark comparing extracted multiply-accumulate operations from different methods and ours. The estimated inference latency on the described TI platform based on the calculated MACs is 38ms.

to fixed-point conversion an accuracy loss of around 3 % could be extracted from the results.

5.5.3 Results

The experimental results and comparison with other state-of-the-art methods are presented in Table 5.1. It is there presented that our method achieves a good Top-1 accuracy better than several methods with a lower number of parameters and FLOPs.

For the first variant of this method in which input data passes first through a convolution with 11×11 kernel we find that a better accuracy with a slight increment of the number of FLOPs can be achieved by increasing the size of this first filter to 13×13 until 17×17. Beyond that, the rate between accuracy and number of parameters decreases. After several tests, like showed in Table 5.2, we have empirically seen that the mentioned 11×11 kernel size gives the best trade-off between accuracy, number of operations and simplicity in the implementation.

Larger kernel sizes increase the model size with more parameters and also more operations and for this reason, using bigger kernels in the initial step of the pipeline would have led to a bigger network, not so suitable for embedded targets.

With regard to the search process speed, the experiments show that this proposal achieves an optimal architecture faster than other DNAS works, as it can be seen in Figure 5.6.

Our experiment results are summarized in Figure 5.6 where we compare our method with state-of-the-art efficient models both designed automatically and manually. In the case of the MnasNet, this paper does not disclose the exact search cost (in terms of GPU-hours or days) so in we have assumed the

Method	# Proposals	Data	mAP(%)	inference time (ms)
Selective Search [44]	2000	Pascal07	66.9	1830
Selective Search	2000	Pascal07+Pascal12	70.0	1830
RPN + VGG unshared [25]	300	Pascal07	68.5	198
RPN + VGG shared	300	Pascal07	69.9	198
RPN + VGG shared	300	Pascal07+Pascal12	73.2	198
Ours	75	Pascal07+Pascal12	**77.9**	**54**

Table 5.4: Results on PascalVOC07 and PascalVOC12 test set. As suggested by other works like [25], the training data is a combination of VOC2007 trainval and VOC2012 trainval.

prediction made for the search cost in MnasNet by [128] and [22].

5.6 Results for object detection

In this section we aim to demonstrate the performance and efficiency of the proposed method comparing the results obtained on different datasets.

We have conducted experiments on the commonly used COCO dataset [140], PASCAL [79] and ImageNet [132] datasets applying several important training techniques, as detailed in Section 5.5.1.

In Table 5.4 our method for region proposal is compared with other widely used approaches in terms of accuracy, number of proposals extracted and inference time. Moreover, in Table 5.5 our detection method is compared with state-of-the-art NAS frameworks with same objective.

In order to validate the proposed methodology for extracting object proposals, we have implemented the methodology explained in Section 5.4.1 with the grouping strategy proposed in Section 5.4.1. Some important considerations taken into account when designing these experiments are the following:

- Each object extraction network candidate is trained from scratch at least 100 epochs on the above-mentioned datasets.

- Each training and testing of the extracted network candidates are done through a set of 5000 and 1500 images respectively, obtained randomly from each target dataset.

- If the features found for at least three consecutive iterations during architecture search do not differ more than 20 pixels from each other (measured from the top-left and bottom-right corners), the architecture search process stops and the object candidates passed to the second steps of the pipeline are the output of the last iteration.

Model	Image size	FLOPs(G)	# Params(M)	inference time(ms)	AP
NAS-FPNLite MobileNetV2 [16]	320 ×320	1.52	2.16	310 (Pixel 1 CPU)	24.2
CornerNet Hourglass [141]	512 ×512	-	-	244 (Titan X)	40.5
FPN R-50 256 [142]	640 ×640	193.6	34.0	37.5 (P100)	37.0
FPN R-101 256	640 ×640	254.2	53.0	51.1 (P100)	37.8
NAS-FPN R-50 (7 256)	640 ×640	281.3	60.3	56.1 (P100)	39.9
DetNASNet [143]	640 ×640	3.8	60.3	-	40.2
NAS-FCO R-50 [144]	640 ×640	189.6	38.4	-	39.8
Ours	**640 × 640**	**275.6**	**57.1**	**56.0 (P100)**	**40.6**

Table 5.5: Results on COCO testset and comparison with other state-of-the-art methods.

5.7 Conclusions

In this work we present a network search approach to design light and optimal DNNs reducing the searching time. We propose a two-step pipeline that learns different meta-kernel sizes, able to treat different resolution patterns. We propose a pairwise searching with circular feedback on each iteration to speed up the process by updating the target weights and network parameters iteratively with, not only the loss calculated during its training, but also with the loss calculated on the parallel kernel being learned.

We demonstrate that our method provides good results in terms of accuracy and searching speed compared to other methods like [16] or [38] under similar computation resource constraints.

Furthermore, our proposed work Stacked-NAS proves accurate a fast results compared to other methods like [144] when running in similar hardware platforms, as shown in Tables 5.1, 5.5 and 5.2.

6 Conclusions

In this chapter we will detail the conclusions obtained during the development of this dissertation, summarizing the accomplished achievements. Furthermore, we will detail the goals obtained at the company FICOSA ADAS S.L.U. during the time this thesis took place, in which company this dissertation was held as part of the *Industrial Ph.D. Program*.

6.1 Conclusions

In this thesis, several theoretical contributions and application results on vehicle pose estimation and neural architecture search are presented. Since this is and industrial PhD, the objectives of this work focus on both the research's impact and the applicability of the developed approaches in real scenarios, paying special attention to the integration constraints of these algorithms into platforms typically used in mass-production projects. Specifically, the conclusions are summarized with respect to the thesis objectives as follows:

- **Investigate and develop vision-based algorithms to be integrated and applied on real scenarios.**

  One of the main objectives of this PhD is to investigate and develop algorithms that provide autonomy to vehicles and overcoming the limitations the automotive industry is facing nowadays in their integration. Over chapters 3, 4 and 5 the different barriers the companies are facing that prevent them to use deep learning based algorithms are presented and solutions to overcome these are proposed. These solutions attend to optimize important concepts that make them usable: light, accurate, platform-oriented. Over the before-mentioned chapters the usage of different sensors in order to create more precise applications has also been contemplated. Although in this thesis the mostly used type of sensors used are RGB cameras, more and more vehicles are being equipped with other light sensors that can complement the information extracted from RGB images to improve accuracy and decrease latency of autonomous driving functionalities. In the chapter 4 is presented an algorithm to detect the 3D shape of the surrounding objects from LIDAR

3D-point clouds. Together with this, the results are compared with image-only methods and the advantages/dis-advantages of such sensors are discussed.

- **Develop and test an AD functionality to estimate the 3D pose of the surrounding vehicles with the least number of sensors.**

  In Chapter 3 are presented two implementations for the detection of the 3D shape of the surrounding objects from an ego-vehicle using only 2D planar images as input. Avoiding the usage of several sensors is a trend in real automotive applications, since it leads to simpler and easier-to-manage systems, as well as it prevents storing and synchronizing data from different sources. According to the results presented in this chapter, the proposed methods provide an accurate and fast prediction of the location and shape of the surrounding objects during a driving maneuver, even when part of them are occluded. Such approaches as the ones presented in the above-mentioned chapter, aim to extract as much information as possible from single images, making less complex systems and lighter functions that can run on typical automotive SoCs (system on chips), with lower computational resources available.

- **Develop and test a scene understanding algorithm based on LIDAR point-clouds and executable on memory-constrained platforms.**

  In Chapter 4 is presented a method that detects object directly from 3D point-clouds provided by a LIDAR sensor mounted on the ego vehicle. Such optical sensors provide a measurement of the depth of the surrounding objects overcoming one of the biggest limitations of RGB cameras, since they only provide planar 2D information. In the before-mentioned chapters the results on a FPGA platform are also presented, comparing the accuracy and latency of the proposed approaches when training on a GPU and testing on a FPGA. These platforms are very suited for parallel computation, which can be very significant in Deep Learning, but on the other hand, they have low memory available. This limitation impacts on the design and the data handling inside the algorithm's pipeline. In this chapter it is presented a way to treat and store the data during training and testing, using the available resources of a FPGA together with an approach to port floating-point implementation (typical from GPUs) to fixed-point (required on FPGAs) without losing performance and precision.

- **Investigate and develop a methodology to design and train neural networks to be executed on resource-constrained platforms.**

  One of the biggest challenges tackled in this thesis is to investigate on how to generate accurate and light DNN architectures that can be executed on ARM, DSP or EVE cores, which are typical commercial hardware platforms. These are used in massed production projects and therefore, shall be cheap and small, not supporting big deep learning algorithms to run on them. In Chapter 5 is presented the investigation done in this direction, proposing a framework to search, train and validate automatically light and accurate deep neural architectures able to perform classification or detection tasks without manual intervention. This means a great breakthrough in the design and implementation of complex Deep Learning functionalities, since it speeds up the design process

and avoids the manual hyperparameter tuning usually needed to adapt networks to new data. Moreover, in this chapter an extension of this approach is presented which works for solving object detection tasks. The main contribution of the research presented in this chapter is the novel methodology for extracting object candidates from images to, later on, be classified into one of the labeled classes.

6.2 Connections with the company

FICOSA is an international supplier for electronic components mainly for the automobile industry. It was founded in 1949 in Barcelona (Spain) and through all this years it has established several headquarters around the world, like in Brazil, USA or China, achieving up to 6500 workers worldwide by 2010.

Although they started manufacturing cables and gears, mainly mechanical components, in the beginning of the 2000's FICOSA started opening new business units more focused on the development of intelligent cameras, battery management systems or infotainment systems. It was then when FICOSA ADAS S.L.U was created as part of FICOSA worldwide. Their development focuses on cameras and electronic control units (ECUs) to be sold to top manufacturers such as AUDI or Volkwsagen all over the world. It was inside this business unit that the work presented in this dissertation was done.

The main contributions of the company to this work have been:

- Specific RGB cameras manufactured internally at FICOSA to elaborate a dataset to perform the training and validation of the algorithms presented in Chapters 3 and 4.

- Hardware with enough computational resources to train, test and validate the implemented DNNs proposed in this work. In this sense, several types of processing units such as GPUs, FPGAs or typical industrial SoCs were provided by the company to extract the results mentioned in Chapters 4 and 5.

- Technical knowledge and support from the rest of the development team.

FICOSA has provided the required tooling and working environment to perform the recordings necessary to create a robust enough data base with which the 3D pose estimation algorithms proposed in Chapter 4 were trained and validated. Moreover, the DNAS and NAS methods obtained during the time that this thesis lasted at the company aimed to start an internal update of some classic computer vision algorithms developed in-house, such as pedestrian detection, with newly designed DNNs.

While many researchers become good experts on their field, they are sometimes quite away from the practical application of their research, what makes the gap between universities and private companies even bigger, since the limitations and daily work in both sites are completely different. What a company like FICOSA has meant for the author of this thesis is the opportunity of minimizing this gap, obtaining a direct feedback of the development done applied on a real scenario, obtaining a good overview of the limitation of both sides.

It is only when research centers and development companies find a common point that benefits both that they can progress and achieve breakthroughs. As commented in Section 3.1, Deep Learning has

reached a maturity point enough to be applied on many problems in a much straightforward way than years before. Together with this, over the last years, new powerful hardware platforms like GPUs or DSPs have been introduced in the market, what enables the introduction of heavy deep learning algorithms on many applications. It is due to this, that initiatives like the *Industrial Ph.D.* are now more interesting than ever, since any research work done in this field can be easier integrated and tested on the real world, shortening the time between new discoveries inside the academical world and its application on a real scenario.

A
List of publications

In this section, the reader can find the complete list of accepted and submitted publications since the beginning of the Ph.D.:

Journals

1. J. García López, A. Agudo and F. Moreno-Noguer. "*Stacked-NAS for object detection*". Computer Vision and Image Understanding, 2020 (under review).

Conferences

2. J. García López, A. Agudo and F. Moreno-Noguer. "*Vehicle Pose Estimation using G-Net: Multi-Class Localization and Depth Estimation*", 21st International Conference of the Catalan Association for Artificial Intelligence (CCIA), 2018. Roses, Catalonia, Spain.
3. J. García López, A. Agudo and F. Moreno-Noguer. Torras. "*Vehicle pose estimation via regression of semantic points of interest*", 11th Int'l Symposium on Image and Signal Processing and Analysis (ISPA), 2019. Dubrovnick, Croatia.
4. J. García López, A. Agudo and F. Moreno-Noguer. "*3D Vehicle detection on an FPGA from LiDAR point clouds*", 2nd. International Conference on Watermarking and Image Processing (ICWIP), 2019. Marseille, France.
5. J. García López, A. Agudo and F. Moreno-Noguer. "*E-DNAS: Differentiable Neural Architecture Search for Embedded Systems*", 25th International Conference on Pattern Recognition (ICPR), 2020. Milan, Italy.

Bibliography

[1] Directorate General of Traffic (DGT) of Spain 2016.

[2] Global Market Research Insight. https://globalmarketresearchinsight.blogspot.com/2016/07/global-automotive-sensor-market.html, 2016.

[3] Novus Light, "Sensors map the path to fully autonomous vehicles," 2018.

[4] Spectrum IEEE. https://spectrum.ieee.org/cars-that-think/transportation/self-driving/researcher-hacks-selfdriving-car-sensors, 2015.

[5] New Electronics. https://www.newelectronics.co.uk/electronics-technology/an-introduction-to-ultrasonic-sensors-for-vehicle-parking/24966/, 2010.

[6] Liu Xinchen, Liu Wu, Mei Tao, Ma Huadong, "A deep learning-based approach to progressive vehicle re-identification for urban surveillance," *European Conference for Computer Vision (ECCV)*, 2016.

[7] Adrien Gaidon, Qiao Wang, Yohann Cabon, Eleonora Vig, "Virtual world as proxy for multi-object tracking analysis," *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[8] Julieta Martinez, Rayat Hossain, Javier Romero and James J. Little, "A simple yet effective baseline for 3d human pose estimation," in *International Conference on Computer Vision (ICCV)*, 2017.

[9] Yin Zhou and Oncel Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4490–4499, 2018.

[10] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, Oscar Beijbom, "Nuscenes: A multimodal dataset for autonomous driving," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[11] Daniel H. Noronha, Bahar Salehpour and Steven J.E. Wilton , "LeFlow: Enabling Flexible FPGA High-Level Synthesis of Tensorflow Deep Neural Networks," *ArXiv e-prints*, July 2018.

[12] Andrew Canis, Jongsok Choi, Blair Fort, Bain Syrowik, Ruo Long Lian, Yu Ting Chen, Hsuan Hsiao, Jeffrey Goeders, Stephen Brown and Jason Anderson, "Legup high-level synthesis," *Chapter in FPGAs for Software Engineers, Springer*, 2016.

[13] Yecheng Lyu, Lin Bai and Xinming Huang, "Chipnet: Real-time lidar processing for drivable region segmentation on an fpga," *arXiv preprint arXiv:1808.03506*, 2019.

[14] Alexander Neubeck and Luc Van Gool, "Efficient non-maximum suppression," *International Association for Pattern Recognition (ICPR)*, 2006.

[15] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun,Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[16] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, Quoc V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[17] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, Kevin Murphy, "Progressive neural architecture search," *European Conference Computer Vision (ECCV)*, 2018.

[18] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le, "Learning transferable architectures for scalable image recognition," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[19] Wenhao Ding, Shuaijun Li, Guilin Zhang, Xiangyu Lei, Huihuan Qian, Yangsheng Xu, "Vehicle pose and shape estimation through multiple monocular vision," *International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[20] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research (IJRR)*, 2013.

[21] Florian Chabot , Mohamed Chaouch , Jaonary Rabarisoa , Celine Teuliere, Thierry Chateau, "Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[22] Shoufa Chen, Yunpeng Chen, Shuicheng Yan, Jiashi Feng, "Efficient differentiable neural architecture search with meta kernels," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[23] Mingxing Tan, Quoc V. Le, "Mixconv: Mixed depthwise convolutional kernels," *British Machine Vision Conference (BMCV)*, 2019.

[24] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam, "Searching for mobilenetv3," *International Conference on Computer Vision (ICCV)*, 2019.

[25] Ross Girshick, "Fast r-cnn," *International Conference on Computer Vision (ICCV)*, 2015.

[26] NVIDIA. http://www.nvidia.com/object/gpu.html, 1999.

[27] United Nations Economic Commission for Europe (UNECE) 2019.

[28] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[29] Karen Simonyan, Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*, 2012.

[30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich, "Going deeper with convolutions," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[31] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally and Kurt Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *International Conference on Learning Representations (ICLR)*, 2017.

[32] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[33] Barret Zoph, Quoc V. Le, "Neural architecture search with reinforcement learning," *International Conference on Learning Representation (ICLR)*, 2016.

[34] Song Han, Jeff Pool, John Tran, William J. Dally, "Learning both weights and connections for efficient neural networks," *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[35] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila and Jan Kautz, "Pruning convolutional neural networks for resource efficient inference," *International Conference on Learning Representation (ICLR)*, 2017.

[36] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[37] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, Hartwig Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," *European Conference on Computer Vision (ECCV)*, 2018.

[38] Hanxiao Liu, Karen Simonyan, Yiming Yang, "Darts: Differentiable architecture search," *International Conference on Learning Representation (ICLR)*, 2019.

[39] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, Joseph E. Gonzalez, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," *Computer Vision and Pattern Recognition (CVPR)*, 2020.

[40] Electronic Design. https://www.electronicdesign.com/markets/automotive/article/21805470/, 2017.

[41] Yann LeCun, Leon Bottou, Y. Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[43] Girshick, R., Donahue, J., Darrell, T., Malik, J., "Rich feature hierarchies for accurate object detection and semantic segmentation," *Internation conference in computer vision and pattern recognition (CVPR)*, 2014.

[44] Jasper R.R. Uijlings, Koen E.A. van de Sande, Theo Gevers, and Arnold W.M. Smeulders, "Selective search for object recognition," *Internation journal of computer vision (IJCV)*, 2012.

[45] Pedro F. Felzenszwalb, "Eficient graph-based image segmentation," *Internation journal of computer vision (IJCV)*, 2012.

[46] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[47] Matthew D Zeiler, Rob Fergus, "Visualizing and understanding convolutional networks," *ImageNet Large Scale Visual Recognition Competition (ILSVRC)*, 2013.

[48] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G. Berneshawi, Huimin Ma, Sanja Fidler, Raquel Urtasun, "3d object proposals for accurate object class detection," *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[49] Adrian Bulat and Georgios Tzimiropoulos, "Human pose estimation via convolutional part heatmap regression," *European Conference on Computer Vision (ECCV)*, pp. 4490–4499, 2016.

[50] S. M. Aiden Nibali, Zhen He and L. Prendergast, "3d human pose estimation with 2d marginal heatmaps," *IEEE's winter conference on applications of computer vision (WACV)*, 2019.

[51] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler and Raquel Urtasun, "Monocular 3d object detection for autonomous driving," *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[52] M.Zeeshan Zia, Michael Stark, Konrad Schindler, "Towards scene understanding with detailed 3d object representations," *International Journal of Computer Vision (IJCV)*, 2015.

[53] Alejandro Newell, Kaiyu Yang, Jia Deng, "Stacked hourglass networks for human pose estimation," *European Computer Vision Conference (ECCV)*, 2016.

[54] Roozbeh Anvari, "Fpga implementation of the lane detection and tracking algorithm," 2010.

[55] Lichao Huang, Yi Yang, Yafeng Deng, Yinan Yu, "Densebox: Unifying landmark localization with end to end object detection," *arXiv:1509.04874*, 2015.

[56] L. Novák, "Vehicle detection and pose estimation for autonomous driving," master's thesis, Czech Technical University in Prague, 2017.

[57] Jason Liang, Elliot Meyerson, Babak Hodjat, Dan Fink, Karl Mutch, and Risto Miikkulainen, "Evolutionary neural automl for deep learning," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2019.

[58] Yihui He and Song Han, "Adc: Automated deep compression and acceleration with reinforcement learning," *ArXiv*, vol. abs/1802.03494, 2018.

[59] Esteban Real, Alok Aggarwal, Yanping Huang, Quoc V Le, "Regularized evolution for image classifier architecture search," *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[60] Guilin Li, Xing Zhang, Zitong Wang, Zhenguo Li, Tong Zhang, "Stacnas: Towards stable and consistent differentiable neural architecture search," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[61] Xiaohan Ding, Yuchen Guo, Guiguang Ding, Jungong Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[62] Fisher Yu and Vladlen Koltun, "Multi-scale context aggregation by dilated convolutions," *International Conference on Learning Representations (ICLR)*, 2016.

[63] Kaiming He, Georgia Gkioxari, Piotr Dollár and Ross Girshick, "Mask r-cnn," *International Conference on Computer Vision (ICCV)*, 2017.

[64] Ashit Talukder and Larry Matthies, "Real-time detection of moving objects from moving vehicles using dense stereo and optical flow," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[65] David Eigen, Christian Puhrsch, Rob Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[66] J. Ivanecký, "Depth estimation by convolutional neural networks," master's thesis, Brno University of Technology, Faculty of Information Technology, 2016.

[67] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, Rob Fergus, "Indoor segmentation and support inference from rgbd images," *European Conference on Computer Vision (ECCV)*, 2012.

[68] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese, "Data-driven 3d voxel patterns for object category recognition," *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[69] Yu Xiang, Wongun Choi, Yuanqing Lin, Silvio Savarese, "Subcategory-aware convolutional neural networks for object proposals and detection," *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[70] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, Deva Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2010.

[71] Bojan Pepikj, Michael Stark, Peter Gehler, Bernt Schiele, "Occlusion patterns for object class detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2013.

[72] Bo Li, Tianfu Wu, Song-Chun Zhu, "Integrating context and occlusion for car detection by hierarchical and-or model," *European Conference for Computer Vision (ECCV)*, 2014.

[73] Alexander Toshev, Christian Szegedy, "Deeppose: Human pose estimation via deep neural networks," *IEEE International Conference on Computer Vision (ICCV)*, 2014.

[74] Tompson, J.J., Jain, A., LeCun, Y., Bregler, C, "Joint training of a convolutional network and a graphical model for human pose estimation," *Advances in Neural Information Processing Systems (NIPS).*, 2014.

[75] Xianjie Chen and Alan Yuille, "Articulated pose estimation by a graphical model with image dependent pairwise relations.," *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[76] Hyeonwoo Noh, Seunghoon Hong, Bohyung Han, "Learning deconvolution network for semantic segmentation," *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[77] Tompson, J., Goroshin, R., Jain, A., LeCun, Y., Bregler, C., "Efficient object localization using convolutional networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[78] Zhu, Derpanis, Yang, Brahmbhatt, Zhang, Phillips, Lecce, and Daniilidis, "Single image 3d object detection and pose estimation for grasping," *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[79] Mark Everingham, Luc Van Gool, Christopher KI Williams, JohnWinn, and Andrew Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision (IJCV)*, 2010.

[80] Roozbeh Mottaghi and Yu Xiang and Silvio Savarese, "A coarse-to-fine model for 3d pose estimation and sub-category recognition," *Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1504.02764, 2015.

[81] Francesc Moreno-Noguer, "3d human pose estimation from a single image via distance matrix regression," *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[82] Georgios Pavlakos, Xiaowei Zhou, Konstantinos G. Derpanis, Kostas Daniilidis, "Harvesting multiple views for marker-less 3d human pose annotations," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[83] Vasileios Belagiannis, Sikandar Amin, Mykhaylo Andriluka, Bernt Schiele, Nassir Navab, Slobodan Ilic, "3d pictorial structures revisited: Multiple human pose estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2016.

[84] Javier García López, Francesc Moreno-Noguer, Antonio Agudo, "Vehicle pose estimation via regression of semantic points of interest," *International Symposium on Image and Signal Processing (ISPA)*, 2019.

[85] Feng Zhang and Xiatian Zhu and Mao Ye, "Fast human pose estimation," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[86] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, Yaser Sheikh, "Convolutional pose machines," *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[87] Ankur Agarwal, Bill Triggs, "3d human pose from silhouettes by relevance vector regression.," *Computer Vision and Pattern Recognition (CVPR)*, 2004.

[88] Gregory Rogez, Jonathan Rihan, Srikumar Ramalingam, Carlos Orrite, Philip H.S. Torr, "Randomized trees for human pose detection.," *Computer Vision and Pattern Recognition (CVPR)*, 2008.

[89] Cristian Sminchisescu and Allan D. Jepson, "Generative modeling for continuous non-linearly embedded visual inference," *International Conference on Machine Learning (ICML)*, 2004.

[90] Gregory Shakhnarovich, Paul Violaand Trevor Darrell, "Fast pose estimation with parameter-sensitive hashing," *Computer Vision and Pattern Recognition (CVPR)*, 2004.

[91] Edgar Simo-Serra, Ariadna Quattoni, Carme Torras, Francesc Moreno-Noguer, "A joint model for 2d and 3d pose estimation from a single image," *Computer Vision and Pattern Recognition (CVPR)*, 2013.

[92] Edgar Simo-Serra, Arnau Ramisa, Guillem Alenyà, Carme Torras and Francesc Moreno-Noguer, "Single image 3d human pose estimation from noisy observations," *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[93] Javier García López and Antonio Agudo and Francesc Moreno-Noguer, "Vehicle pose estimation using g-net: Multi-class localization and depth estimation," *21st International Conference of the Catalan Association for Artificial Intelligence (CCIA)*, 2018.

[94] Tobias Nöll, Alain Pagani and Didier Stricker, "Markerless camera pose estimation - an overview," *Visualization of Large and Unstructured Data Sets (VLUDS)*, pp. 45–54, 01 2010.

[95] Shubham Tulsiani, Jitendra Malik, "Viewpoints and keypoints," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[96] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G. Berneshawi, Huimin Ma, Sanja Fidler, Raquel Urtasun, "3d object proposals for accurate object class detection," *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[97] J. Krishna Murthy, G.V. Sai Krishna, Falak Chhaya, K. Madhava Krishna, "Reconstructing vehicles from a single image: Shape priors for road scene understanding," *IEEE Int. Conference on Robotics and Automation (ICRA)*, 2017.

[98] Andreas Geiger, Philip Lenz, and Raquel Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite.," *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[99] Xinchen Liu, Wu Liu, Huadong Ma, Huiyuan Fu, "Large-scale vehicle re-identification in urban surveillance videos," *IEEE International Conference on Multimedia and Expo (ICME)*, 2016.

[100] Bo Li, Tianlei Zhang, Tian Xia, "Vehicle detection from 3d lidar using fully convolutional network," *Robotics: Science and Systems (RSS)*, 2016.

[101] Cristiano Premebida, João Carreira, Jorge Batista, Urbano Nunes, "Pedestrian detection combining rgb and dense lidar data.," *International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[102] Shuran Song, Jianxiong Xiao, "Deep sliding shapes for amodal 3d object detection in rgb-d images," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[103] Bo Li., "3d fully convolutional network for vehicle detection in point cloud," *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[104] Dominic Zeng Wang and Ingmar Posner., "Voting for voting in online point cloud object detection," *Proceedings of Robotics: Science and Systems (RSS)*, 2015.

[105] Charles R. Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation.," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[106] Charles R. Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space.," *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[107] Xiaolong Liu and Zhidong Deng, "A graph-based nonparametric drivable road region segmentation approach for driverless car based on lidar data," *Proceedings of the Chinese Intelligent Automation Conference (CIAC)*, 2015.

[108] Nicolas Soquet, Didier Aubert and Nicolas Hautiere, "Road segmentation supervised by an extended v-disparity algorithm for autonomous navigation," *Intelligent Vehicles Symposium (IV)*, 2007.

[109] Patrick Y Shinzato, Diego Gomes, and Denis F Wolf, "Road estimation with sparse 3d points from stereo data.," *Intelligent Transportation Systems (ITSC)*, 2014.

[110] Alejandro González, Gabriel Villalonga, Jiaolong Xu, David Vázquez, Jaume Amores, and Antonio M López, "Multiview random forest of local experts combining rgb and lidar data for pedestrian detection," *Intelligent Vehicles Symposium (IV)*, 2015.

[111] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia, "Multi-view 3d object detection network for autonomous driving," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[112] Javier García López, Francesc Moreno-Noguer, Antonio Agudo, "3d vehicle detection on an fpga from lidar point clouds," *International Conference on Watermarking and Image Processing (ICWIP)*, 2019.

[113] Franz Richter-Gottfried, Sebastian Hain, and Dietmar Fey, "Fpga-aware transformations of llvm-ir," *The First International Conference on Advances in Signal, Image and Video Processing (NIPS)*, 2016.

[114] Chris Lattner and Vikram Adve, "Framework for lifelong program analysis transformation," *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization (CCGO)*, 2004.

[115] John Demme, "A compiler infrastructure for fpga and asic development," *arXiv:2003.00151*, 2020.

[116] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, Tian Xia, "Multi-view 3d object detection network for autonomous driving," *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[117] Liang Xiao, Bin Dai, Daxue Liu, Tingbo Hu, and Tao Wu, "Crf based road detection with multi-sensor fusion," *Intelligent Vehicles Symposium (IV)*, 2015.

[118] Xiaofeng Han, Huan Wang, Jianfeng Lu, and Chunxia Zhao, "Road detection based on the fusion of lidar and image data," *International Journal of Advanced Robotic Systems (IJARS)*, 2017.

[119] Liang Xiao, Ruili Wang, Bin Dai, Yuqiang Fang, Daxue Liu, and Tao Wu, "Hybrid conditional random field based camera-lidar fusion for road detection," *Journal in Information Sciences*, 2015.

[120] Luca Caltagirone, Samuel Scheidegger, Lennart Svensson, and Mattias Wahde, "Fast lidar-based road detection using fully convolutional neural networks," *Intelligent Vehicles Symposium (IV)*, 2017.

[121] Tien-Ju Yang, Yu-Hsin Chen, Vivienne Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[122] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter., "Simple and efficient architecture search for convolutional neural networks.," *International Conference on Learning Representations (ICLR)*, 2018.

[123] Javier García López, Francesc Moreno-Noguer, Antonio Agudo, "E-dnas: Differentiable neural architecture search for embedded systems," *International Conference on Pattern Recognition (ICPR)*, 2020.

[124] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, Jian Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," *European Conference on Computer Vision (ECCV)*, 2018.

[125] Xuanyi Dong, Yi Yang, "Searching for a robust neural architecture in four gpu hours," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[126] Cui, Xiaodong and Zhang, Wei and Tüske, Zoltán and Picheny, Michael, "Evolutionary stochastic gradient descent for optimization of deep neural networks," *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[127] Sebastian Ruder, "An overview of gradient descent optimizationalgorithms," *arXiv:1609.04747*, 2017.

[128] Han Cai, Ligeng Zhu, Song Han, "Proxyless nas: Direct neural architecture search on target task and hardware," *International Conference on Learning Representation (ICLR)*, 2019.

[129] Eric Jang, Shixiang Gu, Ben Poole, "Categorical reparameterization with gumbel-softmax," *International Conference on Learning Representation (ICLR)*, 2017.

[130] Javier García López, Francesc Moreno-Noguer, Antonio Agudo, "Stacked-nas for object detection," *Journal on Computer Vision and Image Understanding*, 2020.

[131] Jing Yang, Qingshan Liu, Kaihua Zhang, "Stacked hourglass network for robust facial landmark localisation," *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[132] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," *Computer Vision and Pattern Recognition (CVPR)*, 2009.

[133] Gao Huang, Shichen Liu, Laurens van der Maaten, Kilian Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[134] Mingxing Tan, Quoc V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *International Conference on Machine Learning (ICML)*, 2019.

[135] Xin Chen, Lingxi Xie, Jun Wu, Qi Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," *International Conference on Computer Vision (ICCV)*, 2019.

[136] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, Diana Marculescu, "Single-path nas: Designing hardware-efficient convnets in less than 4 hours," *Machine Learning and Knowledge Discovery in Databases (pp.481-497)*, 2019.

[137] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, Mu Li, "Bag of tricks for image classification with convolutional neural networks," *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[138] Tilya Loshchilov, Frank Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *International Conference on Learning Representations (ICLR)*, 2017.

[139] TEXAS INSTRUMENTS. http://www.ti.com/tool/SITARA-MACHINE-LEARNING, 2018.

[140] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár, "Microsoft coco: Common objects in context," *European Conference for Computer Vision (ECCV)*, 2014.

[141] Hei Law, Jia Deng, "Cornernet: Detecting objects as paired keypoints," *European Conference for Computer Vision (ECCV)*, 2018.

[142] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie, "Feature pyramid networks for object detection," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[143] Yukang Chen, Tong Yang, Xiangyu Zhang , Gaofeng Meng, Xinyu Xiao, Jian Sun, "Detnas: Backbone search for object detection," *Conference on Neural Information Processing Systems (NIPS)*, 2019.

[144] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, Yanning Zhang, "Nas-fcos: Fast neural architecture search for object detection," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.