UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

DOTTORATO DI RICERCA IN
"TECNOLOGIE DELL'INFORMAZIONE"

CICLO XXXIII

Deep Learning-based Object Detection for Autonomous Driving

Coordinatore:
Chiar.mo Prof. Marco Locatelli

Tutore:
Chiar.mo Prof. Massimo Bertozzi

Dottorando: Andrea Zinelli

Anni 2018/2021

Contents

Introduction

Recent advancements in parallel computing hardware and frameworks, coupled with the enormous increase in publicly available annotated datasets, has caused a resurgence in interest from the research community on the topic of machine learning and, in particular, neural networks. In the last decade alone, an extremely wide variety of different neural architectures and models have been proposed to tackle the most disparate problems, including but not limited to sentence translation, image generation, image classification, detection, localization and planning. In particular, the release of increasingly powerful Graphical Processing Units (GPUs) allowed for the development of more complex neural network designs, giving rise to the research field that is currently known as *deep learning*. Amid these designs, perhaps the most important and notorious one is represented by Convolutional Neural Networks (CNNs), a category of neural network that utilizes multiple stacked layers of learnable convolutional kernels to build powerful hierarchical representations of input images and carry out complex visual tasks [1, 2, 3, 4, 5, 6, 7, 8]. All these solutions have shown remarkable performance, often surpassing "traditional" algorithms by wide margins while being more efficient due to their high parallelism. As a result, many "classical" algorithms are gradually being replaced by deep learning-based solutions, bringing about the so-called age of Software 2.0: the software engineer no longer hand-designs the logic of the algorithm directly, but rather designs the neural network model that learns the most suitable logic from the data.

Perhaps one of the industries most affected from this shift in paradigm is automotive: powered by this emergent technology, extensively researched and applied to the field of computer vision, Advanced Driver Assistance Systems (ADAS) and Autonomous Driving solutions have benefitted from tremendous leaps in performance. Among all the tasks that are required for successful autonomous navigation, one of the most critical is perception, which consists of interpreting the signals captured by the sensor suite displaced on the vehicle such that they can successfully be used for the subsequent tasks of tracking, planning and control. Most of the research on deep learning-based perception in the field of autonomous driving concentrates on two specific kinds of sensors, RGB cameras and LiDARs, which are characterized by complementary strengths and weaknesses. Cameras provide a much denser and richer type of signal, while also being considerably cheaper than the LiDAR solutions currently available on the market; on the other hand, they do not provide any information about the distance of the perceived objects from the sensor, and thus require stereoscopic setups to be able to infer the scene depth. Conversely, LiDARs provide an accurate reconstruction of the surrounding environment in the form of a point cloud, but such signal is considerably sparser and does not contain additional semantic information such as color.

Given this data, many deep learning models and techniques have been developed that carry out different types of perception tasks. A particularly important one towards fully automated navigation is *object detection*, which can be defined as follows: given an input, which is the signal returned by one or more sensors, the objective is to identify all entities of interest inside such input and determine their state. Such state can represent any properties of the detected object: image-based 2D detectors [9, 10, 11], for instance, usually estimate an image-aligned bounding box that contains said object, as well as the class it belongs to. 3D detectors [12, 13, 14, 15, 16, 17] estimate 3D bounding boxes that minimally enclose each object, which are represented by their position in the world, their dimensions and their orientation. By integrating additional information, like radar data or past measurements [18], other quan-

tities can be estimated, such as object velocity or trajectory. By summarizing the input signal via a finite set of elements, each one representing the state of a particular detection (e.g. an obstacle on the road), object detectors provide an output that is very simple and compact, and therefore immediately useful to the subsequent planning phase, as it requires little to no additional post-processing. Moreover, deep learning-based object detectors are extremely flexible, allowing to detect almost any category of interest, as long as there exist suitable training data for model optimization.

Given its central role in contemporary autonomous driving pipelines, this thesis focuses on the topic of deep object detection. More specifically, I propose three different systems, each one working on different input data and tackling a distinct detection problem.

The first system consists of an deep learning model for parking space detection and vacancy classification on surround-view images [19]. This approach builds upon the existing two-stage object detector Faster R-CNN [9], modifying its structure and logic to adapt it to the peculiar nature of the task of parking slot detection. Parking slots can be of different types (e.g. rectangular or slanted) and can be observed from different angles, rendering the original implementation, which predicts axis-aligned bounding boxes, ineffective for accurate slot prediction. As such, I propose a variation to the original model, removing the anchor-based region proposal and allowing generic quadrilaterals to be estimated instead of axis-aligned boxes. To train and evaluate the model, I collected and annotated a small dataset depicting different road scenes and parking lots, stitching together the bird's eye view projections of four fisheye cameras to construct surround-view images. Several experiments show the effectiveness of the proposed formulation in unobserved situations, as well as under noise and different observation conditions.

The second work consists in an extention of Faster R-CNN to the task of monocular 3D car detection. This is accomplished by introducing a simple additional neural network module into the original architecture which is tasked to perform image-based 3D detection, essentially by learning the mapping be-

tween object appearance on the image plane and the corresponding pose in the world. To train this module, I propose a novel loss formulation based off the Generalized Intersection-over-Union [20], disentangling the optimization of each degree of freedom of each 3D bounding box for additional training stability [21]. Experiments on the autonomous driving KITTI [22] dataset show the effectiveness of the method, despite it being simple and straightforward compared to other state-of-the-art solutions. Additional studies on the presented GIoU-based loss formulation, conducted by comparing it to different optimization strategies and by adopting it to optimize an entirely different detector [23], validate it as an effective cost function for 3D box prediction.

Finally, the third system consists in a neural network model for performing car, pedestrian and cyclist 3D detection on LiDAR point cloud data. For this work, I leverage the existing PointNet-based [24, 25] model Votenet [26], originally introduced to perform 3D object detection on RGB-D data of indoor scenes. First, to obtain better performance on the noisier and sparser LiDAR data, I introduce modifications to the model point sampling strategy as well as to the classifier logic. Then, I propose to leverage attention mechanisms [27] to explicitly model inter-point relationships and strengthen the feature extraction phase. Experiments on the KITTI dataset show competitive performance against state-of-the-art systems, validating both the proposed modifications and the choice of using attention to extract stronger point cloud representations.

This thesis is organized as follows. Chapter 1 provides a brief introduction to neural networks, CNNs and deep learning in general. It then follows up by reviewing the first deep convolutional models used for image classification and segmentation, as well as the most important state-of-the-art object detection techniques, considering both image-based and LiDAR-based approaches. The following three chapters illustrate the three proposed systems, detailing the methodologies, the optimization strategies, tha data used for training, as well as showing extensive experiments and their corresponding results. Finally, the

conclusions section summarizes the proposed approaches, highlighting and discussing the obtained results as well as outlining potential directions for future research.

Chapter 1

Prior Art

In this chapter I provide an intuitive introduction to Neural Networks starting from the basic linear regression problem. Then, I introduce Convolutional Neural Networks, explaining the differences compared to linear models and detailing the main building blocks and optimization techniques currently adopted in the literature. Follows a brief description of the first deep convolutional models for image classification, and how these evolved into standard components used in most modern architectures. Finally, I describe some of the most prominent approaches to Object Detection, both in 2D based on image inputs and in 3D using images and point clouds.

1.1 What are Deep Neural Networks?

1.1.1 Linear Regression

Suppose that we are given a set of data $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, in which each output $\mathbf{y}_i \in \mathbb{R}^O$ is generated by an unknown function $\mathbf{g}$ applied to its corresponding input $\mathbf{x}_i \in \mathbb{R}^I$ plus some additive noise ϵ, that is we have:

$$\mathbf{y}_i = \mathbf{g}(\mathbf{x}_i) + \epsilon_i \qquad i = 1, \cdots, N. \tag{1.1}$$

Suppose that we are asked, given this dataset, to estimate an approximation of the function $\mathbf{g}$. A common approach for tackling this problem is *linear regression*: this method consists of determining this approximation under the constaint that it is linear, leading to the following model:

$$\mathbf{f}^{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b}. \tag{1.2}$$

In this formulation, $\mathbf{W} \in \mathbb{R}^{O \times I}$ and $\mathbf{b} \in \mathbb{R}^{O}$ are unknown variables used to parameterize the linear function. Determining the optimal set of parameters that best fit the given data consists on solving a minimization problem, where the objective is to minimize the difference between the real output samples $\mathbf{y}_i$ in D and those generated by the model $\mathbf{f}^{\mathbf{W},\mathbf{b}}$:

$$\hat{\mathbf{W}}, \hat{\mathbf{b}} = \arg\min_{\mathbf{W},\mathbf{b}} L(\mathbf{f}^{\mathbf{W},\mathbf{b}}, D). \tag{1.3}$$

$L(\cdot,\cdot)$ is called cost function or loss function and quantifies how well the given model represents the data. A commonly adopted choice for the loss function in linear regression is the quadratic distance:

$$L(\mathbf{f}^{\mathbf{W},\mathbf{b}}, D) = \frac{1}{N} \sum_{i}^{N} \|\mathbf{f}^{\mathbf{W},\mathbf{b}}(\mathbf{x}_i) - \mathbf{y}_i\|^2. \tag{1.4}$$

Given the linear formulation with respect to the weights and fact that the chosen loss function is quadratic, the optimal parameters $\hat{\mathbf{W}}$ and $\hat{\mathbf{b}}$ can be found in closed form. Let:

$$\mathbf{x}_{i+} = \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix} \in \mathbb{R}^{I+1}, \qquad \boldsymbol{\theta}_j = \begin{bmatrix} \mathbf{b}_j \\ \mathbf{W}_j^T \end{bmatrix} \in \mathbb{R}^{I+1} \quad j = 1, \cdots, O, \tag{1.5}$$

where $\mathbf{W}_j$ is used to indicate the j-th row of matrix $\mathbf{W}$. The optimal set of weights $\boldsymbol{\Theta}$ can be obtained by solving the *normal equations*:

$$\hat{\boldsymbol{\Theta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}, \tag{1.6}$$

where

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_{1+}^T \\ \vdots \\ \mathbf{x}_{N+}^T \end{bmatrix} \in \mathbb{R}^{N \times (I+1)} \tag{1.7}$$

is the *design matrix* and

$$\hat{\boldsymbol{\Theta}} = \begin{bmatrix} \hat{\boldsymbol{\theta}_1} & \cdots & \hat{\boldsymbol{\theta}_O} \end{bmatrix} \in \mathbb{R}^{(I+1)\times O}, \qquad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_N^T \end{bmatrix} \in \mathbb{R}^{N\times O}. \tag{1.8}$$

Another approach for determining the optimal parameters $\hat{\mathbf{W}}$ and $\hat{\mathbf{b}}$ consists of *gradient descent*, which is an process that, given an initial value for the parameters, iteratively refines them by computing the gradients $\partial L/\partial \mathbf{W}$ and $\partial L/\partial \mathbf{b}$ and moves the parameter values in the opposite direction, as to reduce the loss function value:

$$\mathbf{b}_{t+1} \leftarrow \mathbf{b}_t - \gamma \cdot \frac{\partial L}{\partial \mathbf{b}_t}, \qquad \mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \gamma \cdot \frac{\partial L}{\partial \mathbf{W}_t}. \tag{1.9}$$

Generally, this procedure continues until either when a certain number of iterations is reached or when a stop condition is met, such as the variation in the loss function value is small enough. The quantity γ represents the *learning rate*, which controls the step of each update and must be chosen carefully depending on the problem. Differently from solving the normal equations (Eq. 1.6), gradient descent does not necessarily return the optimal solution, but rather an approximation. Despite this fact, there exist cases when the latter method is preferrable to the former: solving Eq. 1.6 involves inverting the matrix $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{(I+1)\times(I+1)}$, which might be computationally heavy if the dimensionality I of the data points is high; in these cases, using gradient descent will result in much improved execution times.

1.1.2 Linear Basis Function Regression

Given the imposed linearity constraint, the model resulting from this optimization is able to describe the data appropriately, and therefore be used effectively for estimations on new data points, only if the underlying function that generated the samples is indeed linear. The cases where this mapping is not linear, but we know the family of functions that it belongs to, can be tackled using

linear basis function regression. In this case, each input is transformed into a **feature vector** by a set of predetermined functions, known as basis functions $\mathbf{\Phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), ..., \phi_M(\mathbf{x})]$, before being fed to the linear model:

$$\mathbf{f}(\mathbf{x})^{\mathbf{W},\mathbf{b}} = \mathbf{\Phi}(\mathbf{x}) \cdot \mathbf{W} + \mathbf{b}. \tag{1.10}$$

The use of this intermediate representation allows to model more complex relationships between input and output. For instance, if we know beforehand that $\mathbf{x}$ and $\mathbf{y}$ are both scalars and the process $\mathbf{g}$ generating $\mathbf{y}$ from $\mathbf{x}$ is a polynomial of degree n, we could choose $\mathbf{\Phi}(\mathbf{x}) = [\mathbf{x}, \mathbf{x}^2, ..., \mathbf{x}^n]$ as basis functions, allowing the model to represent polynomial functions up to degree n. Optimizing via the minimization of Eq.1.3 would yield the polynomial that best fits the dataset D.

Despite no longer being linear with respect to the input, this new model is still linear with respect to the weights, which means that a closed form solution can be obtained using the normal equations if the quadratic loss (Eq. 1.4) is adopted: the only difference is that now the design matrix $\mathbf{X}$ does not contain the input values, but rather their feature representations obtained through the basis functions. Note that simple linear regression can be obtained by setting $\mathbf{\Phi}(\mathbf{x}) = \mathbf{x}$.

1.1.3 Parametric Basis Functions: The most basic Neural Network

For complex data it is often unfeasible to determine the nature of the underlying function $\mathbf{g}$, and therefore a suitable set of intermediate basis functions. These cases can be approached by utilizing *parametric basis functions* [28]:

$$\mathbf{\Phi}(\mathbf{x}) = [\phi_1^{\mathbf{W}^1,\mathbf{b}^1}(\mathbf{x}), ..., \phi_M^{\mathbf{W}^M,\mathbf{b}^M}(\mathbf{x})]. \tag{1.11}$$

Here, each $\phi_i^{\mathbf{W}^i,\mathbf{b}^i}(\mathbf{x})$ is commonly a scalar function composed of a parametric linear mapping followed by a non-linear function:

$$\phi_i^{\mathbf{W}^i,\mathrm{b}^i}(\mathbf{x}) = \sigma(\mathbf{x} \cdot \mathbf{W}^i + \mathrm{b}^i). \tag{1.12}$$

The non-linear function $\sigma(\cdot)$, commonly referred to as *activation function*, is necessary to allow the set of basis functions to model non-linear relationships between input and output.

In this formulation, the set of parameters $\{(\mathbf{W}^i, \mathbf{b}^i)\}_i^M$ is also optimized, allowing for the most suitable intermediate representation to be determined directly from the training data. Contrarily to the previous two approaches, however, this model is no longer linear with respect to the parameters, due to the presence of the activation function $\sigma(\cdot)$: as a result, the corresponding minimization problem (Eqs. 1.3, 1.4) is no longer convex. This has two implications: on the one hand, there is no closed form solution for the set of parameters; on the other hand, there is no guarantee that gradient descent techniques will reach the global minimum, as the loss function might now contain local minima and saddle points.

This formulation represents the most basic form of *neural network*:

- the set basis functions $\mathbf{\Phi}$ parametrized by $\{(\mathbf{W}^i, \mathbf{b}^i)\}_i^M$, is commonly referred to as *hidden layer*. This name stems from the fact that the intermediate representation produced by $\mathbf{\Phi}$ is generally abstract and not immediately interpretable by an external observer;
- each $\mathbf{W}^i \in \mathbb{R}^I$ represents a *weight* term, and each $\mathrm{b}^i \in \mathbb{R}$ represents a *bias* term;
- each element $\phi_i^{\mathbf{W}^i, \mathbf{b}^i}(\mathbf{x})$ of the feature vector is commonly referred to as *neuron*, and the number of neurons in the hidden layer is called *width*;
- the hidden layer $\mathbf{\Phi}$ as defined in Equation 1.11 represents a *fully-connected* layer, since each output neuron is a function of the *entire* input;
- the outward linear mapping $\mathbf{f}^{\mathbf{W},\mathbf{b}}$, parametrized by $\{\mathbf{W}, \mathbf{b}\}$, represents the *output layer* of the network.

The Universal Approximation Theorem [29, 30] states that this neural network is theoretically capable of approximating any continuous function, under the

assumption that the activation function $\sigma(\cdot)$ is non-constant and continuous and that the width of the network (i.e. the number of basis functions) is sufficiently high. Given that any algorithm can be represented by a function (e.g. classifying an image can be seen as a function that maps an input matrix of pixels to a probability distribution), this model could be used to solve any task. For many practical applications, however, obtaining a sufficiently accurate approximation of the desired function would require an extremely high width, which leads to prohibitive computational and memory costs. Moreover, this kind of model exhibits a tendency to *overfit* the data, expecially when the dataset is limited. Overfitting is a phenomenon where the model, instead of learning an approximation of the unknown underlying function generating the data, it memorizes the training set instead. As a result, the model performs well on samples belonging to the training set, but exhibits poor results on new, unobserved data points.

1.1.4 From Shallow to Deep Models

The basic idea in *Deep Learning* and *Deep Neural Networks* consists of extending basic neural networks by applying multiple hidden layers in a cascaded way, leading to the following formulation:

$$\mathbf{f}(\mathbf{x})^{\mathbf{W},\mathbf{b}} = \mathbf{W}^{\mathrm{T}} \cdot \mathbf{\Phi}_L \circ \mathbf{\Phi}_{L-1} ... \circ \mathbf{\Phi}_1(\mathbf{x}) + \mathbf{b}, \tag{1.13}$$

where $\mathbf{\Phi}_l(\mathbf{x}) = [\phi_1^{\mathbf{W}^{l,1},\mathbf{b}^{l,1}}(\mathbf{x}), ..., \phi_M^{\mathbf{W}^{l,M_l},\mathbf{b}^{l,M_l}}(\mathbf{x})]$ represents the l-th hidden layer (i.e.: the l-th set of basis functions) of the model. The number of hidden layers that comprise this model is commonly referred to as the *depth* of the neural network. A vast amount of research on a wide variety of different tasks highlighted how deeper models are capable of attaining comparable performance to shallow networks (i.e. networks with one or only few hidden layers) with considerably less parameters, while also being less prone to overfitting. The common consensus on why this is the case is attributed to how deep models extract information from the input compared to shallow ones. Despite the intermediate representation produced by a network with a single

hidden layer is enough to ensure the model is a universal approximator, it is believed to be highly inefficient [31]. Conversely, stacking a sequence of hidden layers one after the other allows a deeper network to learn a representation of the input at multiple scales and levels of abstractions, which in turn allows for more complex function families to be represented more efficiently. Moreover, this hierarchical structure of intermediate representations actively contributes to preventing overfitting, as it is more likely to capture meaningful patterns in the input rather than just memorizing the data points. Increasing the depth, however, is inevitably met with an higher difficulty in optimizing the model. Increasing the number of hidden layers leads to more cascaded non-linear activations, which in turn renders the cost function highly non-convex.

Early on, common choices for the activation function used to be the logistic function, or sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$, and the hyperbolic tangent function $\tanh(x) = (1 - e^{-2x})/(1 + e^{-2x})$. When increasing the depth of the model, however, these functions often cause optimization issues: this is due to the properties of the gradients of these functions, which is always ≤ 1 (in case of sigmoid $\leq 1/4$) and quickly saturates to zero when moving away from the origin. As a result, chaining multiple hidden layers that use this kind of non-linearity leads to gradients that get progressively smaller towards the early layers. This phenomenon is known as the *vanishing gradient* problem, and it hinders optimization as the parameters of these layers cannot update properly. As a result, most modern deep learning pipelines have gravitated towards non-saturating functions. The most widely adopted choice is the Rectified Linear Unit $\text{ReLU}(x) = \max(0, x)$ [32], whose gradient is constant to 1 in the active part of the function, thus avoiding the vanishing of the gradient. In order to have useful training signal for every input value, variations of the ReLU have been proposed, such as the Leaky ReLU, the PReLU [33] and the SELU [34].

1.1.5 Training Neural Networks

Despite the non-convexity of the loss function, gradient descent techniques can still be utilized to optimize Neural Networks: while there is no guarantee that

the optimization will converge towards the global minimum of the function, the hope is that the process will result to a local minimum that is appropriate enough for the task to perform. In order to compute the gradients to use for gradient descent, *backpropagation* [35] is often employed: this method relies on the ability to determine the gradient of simple functions, as well as on the chain rule of differentiability, to progressively determine the numerical value of the gradient of each weight with respect to the loss function. For instance, given the loss function value L, $\partial L/\partial \mathbf{f}$ can be computed directly, which can then be used to recover the derivatives of the weights using the chain rule:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{f}} \cdot \frac{\partial \mathbf{f}}{\partial \mathbf{W}}, \qquad \frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{f}} \cdot \frac{\partial \mathbf{f}}{\partial \mathbf{b}}. \tag{1.14}$$

Here, $\partial \mathbf{f}/\partial \mathbf{W}$ and $\partial \mathbf{f}/\partial \mathbf{b}$ are trivial to compute given the linear relationship between the weights and biases and the function $\mathbf{f}$. Given these quantities, the chain rule can then be enforced iteratively to compute the gradients with respect to the hidden layer parameters:

$$\frac{\partial L}{\partial \mathbf{W}_L} = \frac{\partial L}{\partial \mathbf{f}} \cdot \frac{\partial \mathbf{f}}{\partial \mathbf{\Phi}_L} \cdot \frac{\partial \mathbf{\Phi}_L}{\partial \mathbf{W}_L}, \tag{1.15}$$

$$\frac{\partial L}{\partial \mathbf{W}_l} = \frac{\partial L}{\partial \mathbf{f}} \cdot \frac{\partial \mathbf{f}}{\partial \mathbf{\Phi}_L} \cdot \frac{\partial \mathbf{\Phi}_L}{\partial \mathbf{\Phi}_{L-1}} \cdot \ldots \frac{\partial \mathbf{\Phi}_l}{\partial \mathbf{W}_l} \qquad l = 1, \ldots, L-1 \tag{1.16}$$

.

Again, all the partial derivatives in the above equations can be computed analytically, as the corresponding functions are either linear or depend on the non-linearity $\sigma(\cdot)$ whose derivative should be known.

Modern Neural Networks are often computationally expensive and they are usually optimized over very large datasets. As a result, applying gradient descent directly for their optimization is often intractable. This is due to the fact that gradient descent requires the model to be evaluated on all training samples (i.e. Eq. 1.4) every time a parameter update (Eq. 1.9) is to be performed, which leads to a prohibitive amount of time necessary to reach the local minimum. As a result, most current models are trained using Stochastic Gradient

Descent (SGD) instead: in this variant, each optimization step is performed using only $B \ll N$ randomly sampled data points at a time, where each group of B elements is often referred to as *batch* and B represents the *batch size*. Consequently, performing each parameter update is significantly faster, as it requires only a fraction of the data each time. Once the entire training set is used to perform updates, an *epoch* is completed, the training set is re-shuffled and new batches are created for further training. The theoretical drawback of this method is that it does not guarantee that even a local minimum will be reached during optimization. In pratice, however, SGD tends to reach a satisfactory value for the cost function rather quickly and, by introducing noise into the training process by randomly sampling batches of data each time, often acts as a regularizer, reducing overfitting and leading to models that perform better on new data samples.

Many variations of SGD have been proposed, with the objective of improving and speeding up training. Adding a momentum term [36] allows for a faster convergence towards a minimum and helps avoid oscillations in badly-conditioned regions of the loss function. Adagrad [37], instead of updating each parameter using the same learning rate, adapts the learning rate individually to each parameter. ADAM [38], one of the most used optimizers currently together with SGD with momentum, improves upon the idea of Adagrad while also keeping track of past gradient updates like in momentum SGD.

1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) represent perhaps the most well-known and widely-used category of deep learning models. These neural networks are designed to harness the properties of certain classes of signals of being organized in hierarchies and presenting local patterns. Examples of these kinds of signals include:

- *natural images*, in which neighboring pixels are correlated (the value of each pixel usually depends on its vicinity) and are organized to form

spacial patterns and structures;

- *videos*, where the correlation takes place not only in space but also through time;
- *natural text*, where neighboring characters form words and neighboring words form sentences.

1.2.1 The convolutional operator

Convolution is a fundamental mathematical operator which is widely used in image processing. It can be interpreted as a way to perform multiplication between two signals having different numbers of elements. For instance, given an input image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$, where $H \times W$ represent its *spatial dimensions* and C its number of *channels*, the convolution of this image with a *kernel* $\mathbf{W} \in \mathbb{R}^{K_1 \times K_2 \times C}$ can be defined as follows:

$$\mathbf{y}(h, w) = (\mathbf{x} * \mathbf{W})(h, w) = \sum_{(k_1, k_2) \in S} \mathbf{x}(h + k_1, w + k_2) \cdot \mathbf{W}(k_1, k_2), \quad (1.17)$$

where

$$S = \left[-\left\lfloor \frac{K_1}{2} \right\rfloor, \ldots, \left\lceil \frac{K_1}{2} \right\rceil \right] \times \left[-\left\lfloor \frac{K_2}{2} \right\rfloor, \ldots, \left\lceil \frac{K_2}{2} \right\rceil \right] \quad (1.18)$$

The output $\mathbf{y} \in \mathbb{R}^{H' \times W'}$ preserves the number of spatial dimensions of the input, and each element (h, w) is the result of a scalar product between the kernel and the neighborhood of (h, w) in the original image. In other words, convolution is a function that operates locally and that returns a map of local responses of the kernel to the input. Due to the fact that convolution cannot be computed for those positions in which the kernel is not completely inside the input, the spatial dimensions $H' \times W'$ of $\mathbf{y}$ do not necessarily coincide with those of the input. A commonly adopted technique, especially in deep convolutional models, to preserve the spatial dimensions consists of padding the input, commonly with zeros, such that every location can be processed. Note that, although Eqs. 1.17 and 1.18 describe a bidimensional convolution,

this operator can be applied to signals having any arbitrary number of spatial dimensions by choosing the kernels appropriately.

Even before CNNs, convolution has been widely used in computer vision for many different tasks. For instance, convolution is central in the well-known Canny edge detector algorithm [39] as it is used both for image smoothing (using gaussian kernels) and for gradient computation (typically done using Sobel kernels). Another direct application of convolution in computer vision is image sharpening, which makes use of high-pass kernels.

1.2.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) can be defined as a Neural Network in which at least one of its hidden layers is convolutional instead of fully-connected. A *convolutional layer* is a layer in which the basis functions in $\mathbf{\Phi} = \{\phi_i^{\mathbf{W}^i,\mathrm{b}^i}\}_{i=1}^{M}$ take the following form:

$$\phi_i^{\mathbf{W}^i,\mathrm{b}^i}(\mathbf{x}) = \sigma(\mathbf{x} * \mathbf{W}^i + \mathrm{b}^i). \tag{1.19}$$

The differences with respect to a fully-connected layers are as follows:

- each weight term $\mathbf{W}^i \in \mathbb{R}^{K_1 \times K_2 \times C}$ is now a kernel, applied to the input $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ (in case of bidimensional convolution);
- each bias term $\mathrm{b}^i \in \mathbb{R}$ is a scalar value which is added independently to *each* output of the convolution;
- the activation function $\sigma(\cdot)$ is applied independently to each element resulting from the above transformation.

As a result, the output of each function ϕ_i is no longer a scalar, but rather a matrix in $\mathbb{R}^{H \times W}$ (assuming proper padding is used), and the output of the convolutional layer $\mathbf{\Phi}$ is in $\mathbb{R}^{H \times W \times M}$. A bidimensional convolutional layer can therefore be interpreted as a layer that takes a signal having *spatial dimensions* $H \times W$ and C *channels* as input, and produces a new signal having the same spatial size and M channels in which each element is a feature representation

of the neighborhood of the same element in the input. This output signal is commonly referred to as *feature map*, while each of its elements represents a *neuron*. The convolutional kernels used to compute the feature maps, instead of being hand designed to perform specific operations (e.g. smoothing or gradient computation) are learned during optimization, yielding the set of local transformations that are most appropriate for the task at hand.

Convolutional layers bring several advantages over their fully-connected counterpart:

- the resulting model contains considerably less parameters. For example, if the input to a fully-connected layer contains I elements, the weights of each basis function must contain I elements. For high dimensional inputs, such as images, this would quickly lead to prohibitively large models. The number of weights in convolutional layers, on the other hand, depends only on the amount of channels of the input, which are generally limited compared to their spatial dimensions;

- the convolutional operation is *equivariant* to translations in the input. In other words, if the input signal is shifted along one or more of its spatial dimensions, the corresponding output value does not change but it is subject to the same shift. This property is very powerful, as it implies that convolution is able to capture local patterns within an input independently of where those patterns are. This is in contrast to fully-connected layers, in which a shift of the input signal leads to an entirely different output.

The depth of the model plays an especially important role for CNNs, as stacking multiple convolutional layers one after the other allows the neurons of the model to progressively expand their *receptive field* over the input. The receptive field of a neuron can be defined as the portion of the input that neuron is a function of. Suppose that the first hidden layer of a CNN is comprised of kernels having spatial size 3×3. The receptive field of the neurons produced

by this layer would be 3×3, as each neuron is a function of a 3×3 region of the input. If we now apply a second convolutional hidden layer, again with kernel size 3×3, to the feature map produced by the first layer, the resulting neurons would have a receptive field of 5×5 over the input (IMAGE). By stacking multiple convolutional layers, it is possible to progressively increase the receptive fields of the neurons arbitrarily, potentially allowing each one to "see" the entire input image. This has proven to be very powerful in practice as it allows for very "strong" hierarchical representations to be learned by the model: the neurons of the early hidden layers learn to recognize low-level general patterns, such as edges, dots or other basic shapes. Neurons from later layers become progressively more specialized and abstract, as they have access to more contextual information, and learn to recognize high-level structures that are useful for solving the specific task [40, 41]. This phenomenon is one of the most important contributors to the robustness of this class of models to overfitting, as well as their ability to perform well even when trained on limited data.

In practice, modern deep convolutional models do not keep the spatial dimensions constant throughout the entire model, but rather they progressively reduce them, generating increasingly smaller feature maps. This is achieved either by using *strided* convolutional layers or pooling operators. Strided convolutions operate exactly like normal convolutions, with the difference that some input locations are skipped. In particular, the *stride* of this operator determines how many positions the kernel "moves" when computing the next value: for instance, is the stride is equal to 2 this means that every other element in the input will be ignored, leading to an output whose spatial size is half of that of the input. Pooling operators can be seen as a special case of strided convolutions in which the kernels are not optimized, but rather perform a specific kind of operation over the input. The most widely used are Max Pooling and Average Pooling, in which the kernels perform a max and mean operation respectively. Normal convolutions can also be seen as a special case of strided convolution where stride is equal to 1. The advantage of progres-

sively compressing the feature map size throughout the model is twofold: on the one hand, it allows to rapidly increase the receptive field over the input; on the other hand, it lightens the model and speeds-up computation.

Despite being able to learn powerful and abstract representations, CNNs suffer from the drawback of being computationally expensive to train. Contrary to a fully-connected layer, where each basis function generates a single neuron, each function in a convolutional layer generates a feature map, that is a matrix of neurons. Since error backpropagation through the model requires the knowledge of all intermediate values in order to compute gradients, storing the entire hierarchy of feature maps requires a considerable amount of memory, especially for deep models and inputs high in spatial size. Moreover, the amount of multiplications and summation operations required by convolutional layers is significant. For these reasons, despite being theorized for many years [42, 43, 44], CNN models have been applied successfully to large-scale datasets and complex inputs only in recent years, as a result of the development of modern highly parallel hardware (i.e. GPUs) and programming languages [45].

1.3 The First Large Scale Models for Image Classification

The first truly large-scale deep Convolutional Neural Network can be identified in the pioneering work of Krizhevsky et al. [1]. In their work, the authors employed an 8-layer deep Convolutional Neural Network (later named AlexNet) to win the 2012 edition of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [46, 47], reducing the top-5 error rate from the 25.8 % of the 2011 edition down to 16.4 %. The ILSVRC classification challenge consists of classifying 100.000 RGB images of varying sizes in one of 1000 different classes, given a dataset consisting of 1.2 million and 50.000 annotated images for training and evaluation respectively. The impact that this work had on the computer vision community was considerable: on the one hand, it demonstrated the vast superiority of deep models compared to shallow ones on large

scale and complex datasets; on the other hand, it showed that the training of these kinds of models was feasible using GPU-enabled parallel computing.

Since this discovery, the field of deep learning has seen a spike in interest from the research community, and new and more powerful models were quickly developed. In 2014 two new models, VGG [2] and GoogLeNet [48], lowered the error rate to 7.3 % and 6.6 % respectively. The first one is similar in structure to AlexNet, but concentrates on smaller kernel sizes and is 19-layers deep. The second one is 22-layer deep and introduces the inception module, which takes advantage of kernels having multiple sizes applied to the same input in order to construct multi-scale representations at each layer as well as to reduce the computational cost.

In 2015, Ioffe et al. [49] introduced Batch Normalization, which quickly became a standard operation adopted by most deep models. The purpose of Batch Normalization was to reduce the internal covariate shift (i.e. varying distributions of activations within the model) in order to speed-up the training process. This is achieved by normalizing the activations at each layer to a zero-mean unit-variance distribution using the elements in each mini-batch as samples for the normalization. This leads to a better gradient flow through model, allowing for higher learning rates to be used and, ultimately, a faster convergence. Moreover, by allowing each sample to be used in conjunction with all other samples in the batch through the normalization process, Batch Normalization also acts as a regularizer, improving model generalization. By employing Batch Normalization in conjunction with a variant of GoogLeNet, the authors lowered the error rate on ImageNet down to 4.8 %.

The trend of improving performance by employing increasingly deep models, however, saturated quickly: after a certain number of layers, adding more depth did not bring further performance advantages. Not only that, but after a certain point the performance started to degrade. In particular, what was observed was that this degradation was not caused by overfitting as the models also exhibited higher error rates on the training set, but was rather due to the fact that deeper models are intrinsically more difficult to optimize. To

deal with this increased difficulty, He. et al. in 2015 proposed ResNet [3]. The intuition behind this work is rather simple: given a model, there exist deeper models whose training error is no worse than that of the original model. Such models can simply be obtained by copying the layers of the shallower model and adding any number of additional layers that perform an identity transformation. On the back of this intuition, the authors introduced *residual* layers, that is layers that encode the desired function in the form of a residual from the identity mapping. This is in contrast with the previous approaches, where each layer encoded the desired mapping directly. The hypothesis is that optimizing a specific function starting from the identity should be easier than starting from zero. To force the layers to encode a residual mapping, the authors employed shortcut connections, that is they sum the input of a layer to its output, introducing no extra memory or computational cost. An example of the residual block used can be seen in Fig.(FIGURA). ResNet to its core is simply a network in which multiple residual blocks are stacked one after the others. The authors demonstrated experimentally that this kind of model is indeed easier to optimize, and showed that a model as deep as 1000 layers can be successfully trained to convergence. ResNet-151, a version 151 layers deep, reached an error rate of 3.57 %, winning the 2015 edition of the classification competition.

Other than the pioneering works illustrated above, models that use different achitectures and techniques are continuously being developed. Two notorious examples are DenseNet [50] and MobileNet [51]. The first one builds upon the idea of ResNet, and introduces skip connections that connect each layer with every other layer, allowing for better gradient flow and feature reuse throughout the model. The resulting architectures require less computation, are lighter than ResNet counterparts while achieving comparable performance. The second one makes use of depth-separable convolutions to drastically reduce model complexity while retaining good performance. The resulting architectures are memory and computation light and are targeted towards embedded and mobile vision applications.

Despite the low-level differences between the presented architectures for classification, they all share the same high-level structure:

- an input image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ is processed by a set of convolutional layers which produces a feature map $\mathbf{\Phi}_L(\mathbf{x}) \in \mathbb{R}^{H' \times W' \times C'}$. This representation has lower spatial dimensions compared to the input ($H' < H$, $W' < W$) and can be thought of as a high-level representation of the input content, where each pixel encodes information about a different location. This convolutional part of the model is often referred to in literature as *feature extractor* or *backbone*;

- the feature map $\mathbf{\Phi}(\mathbf{x})$ is "unrolled" into a vector and processed one or more fully-connected layers, whose final output is another vector $\mathbf{y} \in \mathbb{R}^O$ which encodes a probability distribution and O represents the number of possible classes. These fully-connected layers can be interpreted as the task-specific portion of the model, which consumes the high-level representation extracted by the backbone and performs the classification task.

As already stated previously, in convolutional networks the neurons of each layer have access only to a limited scope of the input. Each convolutional layer build upon feature representations extracted by the previous layer, progressively increasing the receptive field over the input and extracting more abstract representations. This hierarchical structure is what makes features extracted by CNNs particularly robust: not having access to the entire input signal, but rather having to build upon low-level patterns to extract high-level representations, makes this class of architectures considerably less prone to memorize the data they are trained on. In other words, the extracted features show a high degree of generality, that is they tend to adapt well on new or unseen patterns, making the underlying models viable for *transfer learning*. Transfer learning consists of harnessing the knowledge already acquired to facilitate and improve the learning on new data and/or new tasks. This proves particularly useful when the task to be performed is complex and the training data is scarce,

as it allows to reuse information acquired on other, and potentially larger and more heterogeneous, datasets. Considering that ImageNet contains over 1 million images spanning 1000 different classes, the backbones of the previously illustrated models have historically been prime targets for *transfer learning*, and constitute to the present day a library of default pre-trained model architectures that are often used as starting point for developing new models and pipelines.

1.4 Semantic Segmentation

An important task for autonomous vehicles perception is image *segmentation*, which can be considered the natural evolution to image classification. In this task, the objective is no longer to assign a class to the input image, but rather to assign a class to each input pixel. That is, given an input image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ the corresponding output is $\mathbf{y} \in \mathbb{R}^{H \times W \times O}$, where O is the number of possible classes and each pixel contains a probability distribution over the classes for the corresponding pixel in the input. In general, the high-level structure of a semantic segmentation network is as follows:

- as in image classification, the input $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ is processed by a convolutional backbone, which extracts the feature maps $\mathbf{\Phi}_L(\mathbf{x}) \in \mathbb{R}^{H' \times W' \times C'}$;
- these feature maps are no longer vectorized and fed to fully-connected layers, but instead are directly processed by a second convolutional network which gradually upsamples these feature maps back to the original spatial size $H \times W$ and predicts a probability vector for each pixel. Commonly used operators to perform upsampling include interpolation followed by convolution and transposed convolution [52].

Segmentation data is considerably more costly to annotate compared to classification data, as it requires a label to be assigned to each pixel of every image. As a result, most publicly available datasets [53, 54, 55] only contain on average a few tens of thousands of samples, a fraction of the data present in ImageNet.

Therefore, the vast majority of research employs transfer learning, adopting the backbone of a pre-existing model trained on ImageNet and substituting the fully-connected layers previously used for classification with convolutional layers that perform segmentation. This way, the weights of the backbone are not learned from scratch, but are rather fine-tuned from an already good initialization, as it comes from the training on a much bigger dataset. This is proven to lead to faster convergence as well as better results, especially if the dataset for the new task is small. Due to the effectiveness of transfer learning, approaches that do not adopt pre-existing backbone architectures still pre-train their feature extractors on the ImageNet classification task before optimizing the entire model on the task at hand.

One of the first fully-convolutional CNNs for semantic segmentation is FCN (Fully Convolutional Network) [56]: in this work, the authors directly adapt AlexNet, GoogLeNet and VGG to the segmentation task by adopting their pre-trained backbones and replacing the fully-connected classifier with transposed convolutions that upsample the feature maps and predict probability distributions for each pixel. The lightest model, FCN32s, performs the upsampling using a single layer, which limits the ability of the network to segment finer details. 32s here indicates that the feature maps before upsampling have stride 32, that is they have resolution that is 32 times smaller than the input. To obtain a higher resolution segmentation, the authors propose two architectural variations, FCN16s and FCN8s, in which they use additional feature maps from earlier layers (and thus having higher resolution) to perform a more gradual upsampling, resulting in more fine-grained segmentation masks. Both of these networks are fine-tuned starting from the pretrained FCN32s model.

In U-Net [8], the authors build on the idea of FCN, and to perform upsampling and prediction they adopt a mirrored version of the backbone, in which each pooling operator is replaced by an upsampling operator. At each resolution, the upsampled features are concatenated and fused with the corresponding features from the backbone at the same resolution, allowing for more accurate localization.

The current state-of-the-art for segmentation models is represented by the DeepLab family of architectures [4, 57, 58]. In DeepLab [4], the authors adopt an ImageNet-pretrained VGG16 model as base for the segmentation. To deal with the limitation that VGG16 produces a feature map at 1/32 the original resolution (which in turn would produce very coarse segmentation maps), the authors remove the last two pooling layers of the model, and employ dilated convolutions in the following layers to make up for the loss of receptive field. The resulting segmentation map is then upsampled from 1/8 the resolution to the original input size using bilinear interpolation, and Conditional Random Fields (CRF) are used to further improve the result and better segment the finer details. DeepLabv2 [57], extends DeepLab by integrating ResNet as additional backbone for better performance and by introducing the Atrous Spatial Pyramid Pooling (ASPP) module, where multiple kernels having multiple dilation rates are used to capture objects at different sizes and scales. DeepLabv3 [58] augments the ASPP module with image-level features and removes the costly CRF post-processing, while still performing considerably better than its predecessors.

1.5 Object detection

Semantic segmentation provides very powerful information for autonomous driving systems. For example, it can be applied to determine the free driving space, or to locate traffic lanes or other kinds of horizontal traffic signs. It represents, however, redundant information when the targets for detection are obstacles such as cars or pedestrians or more complex entities such as vertical traffic signs. In these cases, since semantic segmentation does not provide an instance-level distinction but only returns the class each pixel belongs to, additional post-processing steps would be required to determine the exact location of each single object.

A class of problems that can be considered complementary to semantic segmentation is *object detection*. Object detection can be defined as follows:

given an input $\mathbf{x}$ (e.g. an image, a LiDAR scan etc...), the goal is to locate within this input all the entities of interest as well as to describe their states $\{S_i\}_{i=1}^{N}$. This state might include the position of the entity, its class and other properties such as size or velocity. What makes object detection particularly interesting and challenging is that the number of detections, and therefore the number of outputs to be returned by system, greatly varies depending on the input. For instance, one image might display several tens of cars to be detected or none at all. This is in contrast with the previously presented problems, where the dimension of the output was independent of the content of the input: in classification the output is always a single probability vector; in image segmentation the output is always a probability vector per image pixel. To deal with this complication, the general approach adopted in deep learning consists on estimating a very high number of possible objects and progressively suppress redundant and negative information to obtain the final set of high-quality estimations.

1.5.1 2D Object Detection on Images

Probably the most researched variant of object detection in deep learning is 2D object detection on images. In this case, the state for each object is represented by its class and by the smallest axis-aligned 2D bounding box containing it, that is $S_i = (c_i, x_i, y_i, w_i, h_i)$, where c_i represents the class of the i-th object, x_i and y_i the center of its bounding box in the image and h_i and w_i its dimensions. Pioneering research in the direction of fully deep learning-based object detection is represented by R-CNN (Region-based CNN) [5]. In this work, object detection is achieved via a combination of Selective Search [59] and ImageNet-pretrained AlexNet. Selective Search is an algorithm that, given an image, returns a set of bounding boxes that are likely to be located in interesting regions of the image. The generated set of boxes has high recall (it is likely to contain all interesting objects), low precision (most of the boxes are wrong, inaccurate or duplicates) and is generally very large (around $\sim$ 2000 boxes are generated for a 600x600 image). Moreover, the algorithm is class-agnostic, that

is it does not specify what kind of object is within each of the boxes. RCNN uses Selective Search to determine an initial set of detections, called *region proposals*; each proposal is then used to generate a new image, of size 227×227, by cropping and reshaping its content, that is subsequently given as input to a pretrained AlexNet model, which is tasked to classify the proposal content as well as to compute a correction to the box location and size. In particular, the last fully-connected layer of AlexNet is replaced with two new parallel layers: a layer having $N + 1$ outputs, which represents the probability vector for the desired N classes plus the background class for negative boxes, and a layer for bounding box regression, which returns four values corresponding to the box corrections in position and dimensions. While surpassing all other methods by a large margin, however, RCNN is extremely slow, requiring over 40 seconds per image. This is due to the fact that AlexNet must be applied to each of the 2000+ region proposals, resulting in a system that is impractical for autonomous driving situations, where high frame rates are essential.

A step towards improving upon this limitation is the follow-up work Fast R-CNN [60]. The major contribution of this work consists in feature sharing among different proposals: here the convolutional backbone, which is now VGG16, is applied to the entire input image, obtaining a feature map that is shared for all objects. Then, for each region proposal returned by Selective Search, the corresponding area *in the feature map* is determined by projecting the box and warped to a standard 7x7 size using the so called RoI Pooling operation. This pooled set of features is finally used for classification and box regression. This formulation has two major advantages over the original design: on the one hand, applying the convolutional backbone to the entire image instead of once per proposal results in a considerably faster computation, over 200 times faster than RCNN. On the other hand, sharing the same set of features for all proposals results in a stronger intermediate representation that is more aware of the context, which leads to improved performance.

Despite being considerably faster than RCNN, Fast R-CNN is still bottlenecked by Selective Search, which requires around 2 seconds per image to

generate the proposal boxes, slowing down the entire pipeline considerably. A solution to this problem is introduced by Faster R-CNN [9]. In this work, a specific subnetwork, called Region Proposal Network (RPN), is used instead of Selective Search to estimate the set of proposal boxes. RPN generates the proposal boxes by assigning to each pixel of the feature map generated by the backbone a set of predefined boxes, called *anchors*. For each anchor, it then estimates whether it is an interesting or a background box, as well as computing a correction for it. The proposal boxes are finally obtained by removing the background anchors and by filtering duplicates via Non-Maximum Suppression. These proposals are then processed as in Fast R-CNN to obtain the final detections. Further details about this approach will be reviewed in the next chapter, as Faster R-CNN is central to both the proposed Parking Slot Detection network as well as the 3D Object Detection network. By avoiding Selective Search, and performing both region proposal and bounding box estimation in a unified forward pass of the model, Faster R-CNN is considerably faster than its predecessors, allowing for more than 15 images to be processed per second. Moreover, it achieves top performance, making it one of the current state-of-the-art approaches. This method is further improved in a subsequent work [11], where a Feature Pyramid Network is used as backbone in order to extract feature maps at multiple scales. Anchors and proposal boxes are then assigned to the proper scale depending on their size.

The R-CNN approaches are widely regarded as *two-stage* approaches. This is due to the fact that the final detections are obtained through two separate phases: first, a set of proposal boxes is generated. Then, each proposal is analyzed individually (i.e. through the pooling of its features) in order to compute its class and refine its state. Another class of object detection techniques is represented by *single-stage* methods: here, the final detections are obtained directly, without exploiting an intermediate set of object proposals.

One of the first works in this direction is represented by YOLO (You Only Look Once) [6]. Here, object detection is cast as a single convolutional (ndr: there are 2 FC layers tho) neural network that estimates the bounding boxes

directly from the input image. To achieve this, the input image is divided into an $S \times S$ grid of cells, and the network is tasked to estimate B potential boxes per cell. To do this, the input is processed by a stack of convolutional and pooling operations such that an $S \times S \times (B * 5 + C)$ feature map is obtained. Each pixel of this map is responsible for returning the boxes for the objects whose centers lie in the corresponding grid cell of the input: in particular, the network returns B possible boxes per cell with the respective confidences, center coordinates and dimensions, as well as a C-class probability vector that indicates the category of the detected object. The unified design adopted by this approach allows for extremely fast predictions: the base model, which is a modified GoogLeNet architecture, is able to perform detection at 45 frames per second, while the light model, which is similar to the base model but with less layers, reaches 155 frames per second.

Similarly to YOLO, SSD (Single-Shot Multibox Detector) [7] approaches object detection using a single, unified, convolutional neural network. Instead of predicting the boxes directly, however, in SSD the detections are obtained as refinements of a set of predetermined boxes (similarly to the anchors in Faster R-CNN). Moreover, instead of using a single feature map to perform detection, multiple feature maps at different scales are used for prediction: the low-resolution maps are adopted for bigger predetermined boxes, while the high-resolution ones are used for smaller predetermined boxes. Harnessing features at multiple scales for differently sized boxes allows the resulting detections to be more accurate, considerably improving performance over YOLO.

YOLO was subsequently improved, leading to two new models: YOLOv2 [10] and YOLOv3 [61]. YOLOv2 is similar to its predecessor with a few tweaks and changes aimed at improving recall and localization accuracy of the boxes. Firstly, they improve upon the network architecture: they adopt a new model similar to VGG16 as base, called Darknet-19, and they integrate Batch Normalization, which speeds up training and improves performance. Moreover, they strengthen the pretraining on ImageNet. Secondly, they adopt an anchor-based design, where the predictions are obtained from refining predetermined boxes

instead of being computed directly. In particular, instead of hand-picking the anchor dimensions like SSD or Faster R-CNN, they obtain them by running k-means clustering on the datasets' ground truth boxes; this way the distribution of anchors is closer to that of the ground truth boxes, which simpifies optimization and leads to improved performance. Finally, they increase the robustness of the system to different object scales by adopting multi-scale training: every 10 epochs, a new random scale is selected and the model is trained on images resized to that scale. YOLOv3 further boosts performance by adopting a much deeper model with residual connections and by performing prediction using feature maps at multiple scales.

All previous approaches share a common problem: on average, the number of overall background grid cells or anchors is far greater than those actually containing objects. This discrepancy often leads to suboptimal optimization of the classifier, since the gradient value is overwhelmed by background examples. Some approaches, such as the RCNN family and SSD, deal with this problem by forcing the loss function to be computed on a balanced set of positive and negative samples. Even in this case, however, most of the training signal is dominated by easily classified background examples.

A technique often adopted to deal with this limiatation is OHEM [62] (Online Hard Example Mining): here, the loss function is applied to all the samples, but the gradient is computed only for the n samples for which the network performs the worst, that is the n samples for which the loss value is highest. This has the effect of forcing the training to focus only on the more difficult cases, ignoring the easier ones.

Another solution is proposed in [63]. In this work, the authors introduce RetinaNet, a single-stage object detector. This model adopts ResNet with FPN as backbone, and performs object detection at multiple scales by classifying and refining anchors, similarly to SSD. To deal with the inbalance between positive and negative classes, they introduce the focal loss: in this formulation, instead of computing the loss value as the average of cross-entropy losses of each sample, the loss value is computed as the sum of the cross-entropy losses

for each element, but each one is downweighted depending on how well the model performs on it. Hard examples will receive high weights, whereas simple examples will be assigned progressively lower weights. As a result, during optimization, training automatically concentrates on the hard examples; this is contrary to the original cross-entropy formulation, in which the contribution to the training of the few hard examples is decimated by the averaging operation over all samples. The focal loss, moreover, carries an advantage over OHEM: in OHEM easy samples are ignored; instead, when using focal loss all samples contribute to the optimization, albeit with reduced impact if easy.

1.5.2 Tradeoff between One-stage and Two-stage techniques

The previous paragraph introduced some of the main state-of-the-art object detectors, categorizing them in two-stage and single-stage pipelines.

Two-stage detectors perform detection in two distinct phases: first, they generate a set of proposals, that is a set of intermediate detections having high-recall (most required entities are detected) but low precision (most proposals are wrong or inaccurate). Then, a second part of the model analyzes each proposal by pooling its features and outputs a classification decision as well as a correction to its state. Proposal generation can either be performed using existing algorithms, such as Selective Search, or by employing a Region Proposal Network. On the other hand, single-stage methods perform estimation directly, without making use of an intermediate set of proposals.

There is no clear winner between the two classes of methods; which paradigm to use depends on the specific use-case and situation. Single-stage methods tend to be faster than their two-stage counterpart while also being simpler from a logical standpoint, requiring less complex code and being more straightforward to implement and debug. Two-stage methods, on the other hand, have an edge performance-wise. Where the greatest advantage of two-stage methods lies, however, is their far greater flexibility: by adopting region proposals and by pooling their features, it is considerably easier to integrate additional tasks into the original pipeline.

The most notorious work in this direction is represented by Mask R-CNN [64]. Mask R-CNN expands Faster R-CNN to include a segmentation mask prediction for each proposal, on top of the classification and position refinement. This is achieved by adding an extra convolutional module which processes the pooled feature map of each object and returns an upsampled binary mask for that object. To further improve performance and better align the pooled features for the semantic task, the authors propose RoIAlign, an upgrade to the RoIPooling operator, which avoids the quantization effects of the latter by adopting adopting bilinear interpolation.

Another interesting line of research that directly takes advantage of the two-stage structure is Tracking without Bells and Whistles [65]. In this work, the authors propose a small modification to Faster R-CNN to directly perform tracking, without the need for additional training or tracking specific data. In particular, the detections of each frame are added to the proposal pool for the successive frame and corrected by the refinement network to directly create trajectories, while the original set of proposals is used to identify new objects and initialize new tracks.

Given this extra flexibility and performance, two-stages methods, and especially Faster R-CNN, constitute the foundation for two of the solutions proposed in this thesis: the parking slot detection network and the monocular 3D object detection network.

1.6 3D Object Detection

While being useful for applications such as monitoring systems or security cameras, in autonomous driving detecting objects at image level is often insufficient, as it gives no information on where each instance actually is in the world. As a result, new systems and algorithms are continuously being developed that perform detection in 3D. Formally, 3D object detection consists of locating in the input all instances of objects of interest and return, for each one:

- its class c;
- its 3D bounding box, expressed with respect to some frame of reference. This box is identified by its center point (x, y, z), its dimensions (h, w, l) and its orientation expressed as roll, pitch and yaw angles (ϕ, ψ, θ).

In other words, the state of the object i is now represented by the vector

$$S_i = (c_i, x_i, y_i, z_i, h_i, w_i, l_i, \phi_i, \psi_i, \theta_i). \tag{1.20}$$

When operating in driving scenarios, a common assumption is that all objects of interest lie on a plane, and can therefore be subject only to rotations around one axis, simplifying the state to $S_i = (c_i, x_i, y_i, z_i, h_i, w_i, l_i, \theta_i)$.

Input data used for 3D object detection systems often include images, either coming from monocular or stereo camera setups, and point clouds, obtained by disparity map triangulation or LiDAR sensors.

1.6.1 LiDAR-based 3D Object Detection

LiDAR (Light Detection And Ranging) sensors are commonly adopted choices for autonomous driving as they are able to provide highly accurate depth information about the environment. They operate on the time-of-flight principle: distance from an object is calculated from the time required by a light impulse, emitted by the sensor itself, to reach the object and be reflected back to the sensor receiver. The resulting raw data returned by this class of sensors is a point cloud, that is a set of 3D points in space. Some sensors might also return additional information for each of these points, such as the reflectivity of the corresponding objects. The most typically adopted choice in automotive is represented by Velodyne sensors, due to their ability to rapidly perform 360° scans while also having reasonable vertical Field-of-View and point density through the use of multiple emitter-receiver pairs (verify). The resulting scans usually contain from several tens of thousands to a few hundred thousand points, depending on the sensor model and the number of scan planes.

Point clouds are considerably different from images: the latter are dense grids of elements, organized in a specific structure where position matters. On the other hand, point clouds are sparse and are not characterized by a specific ordering: a permutation of a point cloud is equivalent to the original point cloud. As a result, convolutional models, which are based on the assumption that the input is dense and regular, cannot be directly applied to this data type.

An early work in deep learning-based 3D object detection [66] attempts at bridging the gap between image and point cloud data modalities by projecting the LiDAR point cloud on a Bird's Eye View (BEV) plane, which is then used as input to a convolutional model that directly performs 3D bounding box prediction and classification. Another similar work is represented by MV3D [67]: here, both LiDAR and image data are fused together to perform detection. In particular, three separate convolutional backbones are used to extract feature maps: one from the image, one from a Bird's Eye View projection of the LiDAR point cloud and one from a frontal view projection of the same cloud. Then, the Bird's Eye view features are used to generate a set of initial 3D proposal boxes via an RPN. Finally, these proposals are projected back on all three feature maps, whose features are pooled accordingly, fused and used to estimate the final boxes. AVOD [12] adopts a similar strategy for fusing image and LiDAR information, but only uses the Bird's Eye View projection of the point cloud, generated similarly to MV3D. In this method, however, the proposal boxes are generated using both image and Bird's eye view LiDAR features, resulting in improved recall for small instances.

Despite enabling all previous methods to harness the representational power of convolutional models and pre-existing architectures, point cloud projection inevitably leads to information loss, limiting the ability of such models to reason in 3D and ultimately compromising performance. Currently, the state of the art approaches for 3D Object Detection can be classified in two macro categories: voxel-based methods and point-based methods.

In voxel-based methods, the input point cloud is first converted into a voxel

grid, which is then processed using 3D convolutional operators. Considering the sparsity of a LiDAR scan, however, voxel conversion often results in a very high ($> 90\%$) percentage of empty voxels. To considerably reduce computational and memory requirements, voxel grids are often represented and stored as sparse tensors and 3D convolutions are computed only at locations that contain non-empty voxels, also known as **active sites**; these convolutions are often referred to as sparse convolutions or sub-manifold convolutions as they can be seen as operating on low dimensional data living in a high-dimensional space (e.g. surfaces in 3D) [68, 69].

Conversely, point-based methods aim at processing the raw point cloud directly, without any conversion or quantization. This is achieved using PointNets, specific models designed to handle the permutation invariance property of point clouds: a permutation of a point cloud corresponds to the same point cloud, meaning that feeding different permutations to a model must not change the output of the model. The idea behind PointNets, first introduced in [24], is rather simple: first, each point is processed independently and identically using a series of fully-connected layers; then, a global representation, or signature, for the point cloud is computed by applying a symmetric function (i.e. a function whose result does not depend on the order of its inputs) to the processed points. For classification, this representation is further processed by additional fully connected layers, which convert it into a probability vector. For segmentation, the signature is concatenated to the features of each point, in order to enrich them with global information. Then, each point is processed separately in order to estimate the class. While exhibiting very promising results on both classification and segmentation in controlled environments (e.g. single object, small rooms), PointNet suffers from a limitation that inhibits its applicability to more complex scenarios and tasks: by applying the symmetric function only once to the entire point cloud, this model in unable to reason hierarchically and capture local patterns. The follow-up work PointNet++ [25] improves upon this aspect, allowing hierarchies of local representations to be built from the input. The core component introduced in this work is the *Set*

Abstraction layer, which takes as input a point cloud containing N points and performs the following operations:

- first, it applies the Farthest Point Sampling algorithm to sample S centroids from the input cloud;
- second, it creates groups of points by assigning to the centroids the points that are close enough to them;
- third, it applies a simple PointNet to each group to extract global information.

The resulting output of this layer is therefore a new point cloud containing S points, located at the centroids positions, and whose features are a function of the local neighborhood of each centroid. By stacking multiple set abstraction layers one after the other with increasing grouping thresholds, it is possible to model local relationships between points with increasing receptive fields similarly to how convolutions do for images. To restore the original point cloud resolution (e.g. to perform point segmentation), *Feature Propagation layers* are used, in which the features from the low-dimensional point clouds are propagated back to the original point locations using distance based interpolation.

One of the first voxel-based approaches can be identified in VoxelNet [70]: here, a 3D feature volume is computed by subdividing the space into a grid of equally spaced voxels and by applying a PointNet-like network, called Voxel Feature Encoding (VFE), to the points inside each voxel. Then, the resulting voxel grid is processed by several 3D convolutions, which aggregate information in a progressively larger receptive field in order to capture shape information. Finally, the processed volume is aggregated along the height direction to generate a 2D feature map, which is fed to an RPN to perform 3D box prediction and classification.

SECOND [71] builds upon VoxelNet, proposing an algorithm for the generation of the indices corresponding to active sites that runs on GPU, leading to drastically reduced execution times for sparse and sub-manifold convolutions. It also introduces a novel loss function for angle estimation that allows

to better handle the cases in which the predicted orientation differs from the ground truth value by $\approx \pi$, and a novel data augmentation procedure that consists of adding to each point cloud objects from other scenes in order to obtain richer samples and accelerate and improve training. PointPillars [13] aims at performing 3D object detection employing only 2D convolutional layers. This is achieved by operating on pillars instead of voxels: the point cloud space is discretized only along the width and length axes, while no binning is performed in the height direction. The points inside each pillar are then encoded using a simple PointNet, and the resulting encoded pillars are used as pixels to create a bidimensional feature map. Finally, this map is processed by a 2D convolutional backbone followed by a detection head similar to that in SSD [7] to perform 3D bounding box detection. By adopting this particular encoding, this approach surprasses in performance the previous two methods while avoiding the computational bottleneck represented by 3D convolutions, allowing it to operate at over 60 FPS.

Among the early works on point-based 3D object detection is Frustum-PointNets [23]. This approach splits the prediction process into two steps by leveraging both image and 3D data: first, a high-performing 2D object detection model (e.g. Faster R-CNN) is applied to the image in order to extract a set of 2D bounding boxes. These boxes are then used to cast viewing frustums into the 3D space, determining regions that contain objects. The points inside each frustum are then selected and given as input to three PointNet-like models which perform, in sequence: background points removal, object center estimation plus point cloud normalization and, finally, 3D box prediction. This method, while outperforming other fusion-based pipelines such as MV3D, suffers from several drawbacks: first of all, both image data and 3D data are required, as the former is necessary to generate frustums for the latter. Second, the overall system is bottlenecked in performance by the 2D object detector: if a 2D detection is missed, no frustum is generated and therefore no 3D box is computed. Likewise, a wrong 2D detection always leads to an erroneous 3D box, as a 3D detection is always returned for each frustum.

The first successful purely point-based 3D detection approach on autonomous driving data is represented by Point R-CNN [14]. In this work, the authors first leverage PointNet++ to extract a feature representation for each point in the point cloud. Then, a 3D proposal as well as an objectness propability are estimated for each point from this representation. Positively classified proposals are propagated to a second stage, which pools points that are located in the vicinity of each proposal and uses this information to perform box correction and confidence prediction. Despite using only LiDAR data, this approach vastly outperforms all previously mentioned fusion-based schemes. It also performs better than the illustrated voxel-based approaches, albeit being slower, as no quantization (and therefore information loss) is involved. A similar approach, STD [72], introduces spherical anchors in the first stage to estimate the initial set of 3D proposals. In the second stage, the points inside each proposals are converted to a grid of voxels and processed by a VFE before being passed to the estimation subnetwork. Finally, the estimation subnetwork, other than performing classification box correction, also contains an additional branch which is supervised to predict the 3D Intersection-over-Union between the predicted and the ground truth box. This information is used as an additional confidence measure about the quality of the predicted boxes when performing NMS, improving the overall performance of the pipeline.

Another point-based method can be identified in Votenet [26]. Differently from Point R-CNN, here no proposals are used and prediction is performed in a single stage: PointNet++ is again used as backbone, but instead of computing feature representations for the entire point cloud, these are computed only for a subset of points, called *seed points*. Seed points are then processed by a small fully-connected network, which is supervised to estimate a shift that pushes the seeds close to the object centers, creating clusters around potential object. The resulting shifted points, called *votes*, are clustered together and processed by a second PointNet that estimates whether each cluster corresponds to an object and the corresponding 3d bounding box. This formulations outperforms approaches such as Frustum PointNets in scenarios that are dense of objects,

such as indoor rooms, but struggles on autonomous driving data, where the majority of points do not belong to objects of interest. 3DSSD [73] operates similarly to Votenet: the seed points are again shifted to generate candidate points, but these are no longer clustered together, but are rather used to pool the original set of seed points. Arguably the most significative innovation introduced in this work, however, concerns point sampling: instead of adopting the simple euclidean distance to sample the points at each abstraction layer of PointNet++, the authors propose a mix of euclidean distance and feature distance. This has the effect of retaining a higher percentage of "interesting" points while discarding similar points (i.e. points belonging to the road plane), leading to an overall improvement to the recall of the system.

Voxels and raw points constitute two fundamentally different representations for the same data, and as such they share complementary strengths and weaknesses: voxels are generally easier to manage as they are ordered structures, akin to images. This makes it relatively simple to adopt similar techniques for their processing, consisting mostly of 3D convolutions applied in a cascaded way to progressively enlarge the receptive field over the space. Moreover, sparse and sub-manifold convolutions make processing even large volumes extremely efficient. The quantization process involved in the conversion from points to voxels, however, limits the resolution of the data, which in turn might lead to a degradation of the performance. Raw points, on the other hand, are subject to no quantization so they are able to provide precise localization of all objects of interest. In turn, they prove to be more challenging to process as they are sets, and therefore they have no internal organization and are invariant to permutations. As such, some recent work focuses on exploiting both modalities in order to improve efficiency and performance. SA-SSD [74] adopts a 3D convolutional backbone similar to VoxelNet to extract a pyramid of 3D features at multiple scales. As in VoxelNet, the last set of features is converted into a 2D feature map by aggregating the features along the height direction, which is then processed by a detection head that performs anchor classification and refinement in order to obtain the final detections. In addition to the above,

this method also employs a point-based auxiliary network in order to encourage the voxel-based backbone to learn the fine-grained structure of the point cloud: each feature map in the 3D convolutional pyramid is used to generate point-level features at the original point cloud resolution. This is achieved by identifying the position of each voxel according to its index and by propagating the features at the original point locations using a feature propagation layer. The resulting enhanced point cloud is then used to perform two auxiliary task: foreground point segmentation and object center prediction. Once the model is trained, the auxiliary network can then be discarded, leaving only the enhanced voxel-based detector. Another recent work leveraging both modalities is PV-RCNN [75]: here, a VoxelNet-like model is used to produce an initial set of region proposals. At the same time, a set of keypoints is sampled from the original point cloud using Farthest Point Sampling, and the precise locations of keypoints are used to aggregate features from the original point cloud, the multiscale voxel features extracted by the backbone and the 2D features generated by the Region Proposal Network. Finally, the resulting enhanced keypoints are used in a second stage to refine the proposals estimated from the voxels.

1.6.2 Image-based 3D Object Detection

Range sensors, while particularly suited to 3D detection due to the accuracy of the data they return, tend to be expensive: a single Velodyne sensor, for instance, might exceed alone the cost of the rest of the vehicle, which in turn renders consumer-level products impractical. As a result, a lot of research has concentrated on performing 3D sensing using cheaper alternatives. In particular, in autonomous driving it is widespread the use of cameras: this type of sensor is orders of magnitudes cheaper compared to LiDARs, it returns rich and dense informantion and, by adopting stereo-camera setups, it can be used to percieve the 3D structure of the environment, albeit less accurately compared to the dedicated alternatives.

Currently, there are two main classes of image-based deep learning ap-

proaches to 3D detection: stereo-based methods, which use pairs of rectified images as input, and monocular-based methods, which aim at detecting 3D objects from a single image.

Stereo-based Detection

A recent work on deep learning-based stereo 3D detection can be identified in Triangulation Learning Network [76]. This approach first proposes a monocular baseline that performs 3D detection from a single frame, and then extends it to the stereoscopic case. The baseline operates similarly to Faster R-CNN, with the difference that all estimations revolve around 3D boxes: in particular, 3D anchors are displaced in the 3D space and then projected on the image to determine their 2D counterparts. Features from the 2D anchors are then pooled and processed by the RPN, which determines 3D proposals. Finally, the same process is repeated using the 3D proposals, whose features are passed to the detection portion of the model to determine the final predictions. To extend this framework to stereoscopic data, both left and right frames are processed in parallel by the network, the 3D anchors and proposals are projected on both frames and the respective features are fused using an ad-hoc scheme that accounts for potential mismatches due to different depths. The addition of the second image and the fusion scheme leads to moderate improvements in performance; despite this, however, this method performs considerably worse than other contemporary stereo approaches and is even outperformed by some monocular pipelines.

Another similar approach is Stereo R-CNN [17], which also builds upon Faster R-CNN. Instead of relying on 3D anchors or proposals, however, this method fuses the left and right features computed by the same backbone to predict matching left-right 2D proposals from 2D anchors. These proposals are then used to pool the corresponding left-right features, which are finally employed to estimate the left-right 2D boxes, a set of image keypoints, as well as the corresponding 3D object dimensions and orientation. Using this information, the final 3D box is obtained via triangulation. To further improve

performance, a final alignment phase is performed, in which the depth of the detections is corrected by minimizing the photometric error of objects in the left and right images. This correction, in particular, proves to be central to the method, accounting for most of the performance gain.

A seminal work towards accurate stereoscopic (and monocular) 3D object detection is represented by Pseudo-LiDAR [77]. The idea behind this research is surprisingly simple: the performance gap that exists between LiDAR-based and Stereo-based detectors is not to be attributed solely to the technological differences between the two types of sensors, but also to the data representation that is used to train the models. In fact, what the authors observed is that the point clouds resulting from disparity triangulation are indeed not inaccurate enough to justify such a wide difference in result quality. To validate this hypothesis, they introduced a two-step pipeline: first, state-of-the-art approaches are used to extract a disparity map from the stereo pair, which is then converted into a point cloud; second, state-of-the-art LiDAR-based 3D detectors, such as AVOD and Frustum-Pointnets, are applied to the point cloud to perform detection. The resulting system decisively outperforms all purely image-based stereo systems, validating the claim. One of the main reasons as to why a point cloud representation allows for increased performance compared to an image-based one is as follows: processing point-cloud data, either by using 3D convolutions, 2D convolutions on BEV representations or PointNets, ensures that the elements that are operated upon together are physically close in space. Convolutions on images, on the other hand, operate identically on patches corresponding to objects at different scales (far away objects are smaller on images compared to nearby objects) or patches in-between objects and background (and therefore involving entities very far away in physical space), making them less suitable for reasoning in 3D.

Monocular Detection

Contrary to LiDAR-based or Stereo-based 3D detection, monocular detection is an ill-posed problem, as a single image does not provide enough information to

recover the scale of the portrayed scene, and therefore the depth of the objects. One way to recover a good enough approximation of the position of interesting objects is to use a priori knowledge about them, such as their dimension. For instance, if the height of a specific traffic sign is known then it would be possible, given the camera intrinsic parameters, to infer the approximate distance from the sensor. Another possible way is by learning the relationship between the way objects appear in the image plane and their corresponding state in the world using accurate ground truth and deep learning models. This would be similar to how humans with one eye covered would still be able to estimate the 3D structure of the world, despite not having, theoretically, enough information to do so.

Due to the challenging nature of the problem, to ease learning and improve performance most monocular 3D detection pipelines embed some form of a priori knowledge or 3D reasoning mechanism directly within their models. In Mono3D [15], the authors leverage the assumption that all objects should lie on the ground plane in order to generate object proposals. In particular, they use camera calibration information to determine a fixed plane, they generate 3D candidate proposals on this plane and they project them on the image in order to score them and keep only the most promising ones. The overall score for each candidate is determined using semantic, instance, shape, location and context cues, which are computed using external methods. The best candidate proposals are then processed further by a second stage similarly to Fast-RCNN: a VGG16 model is used to extract features from the input image, RoIPooling is used to obtain the features for each proposal and fully-connected layers are used to estimate the class, the position of the object as well as its orientation.

OFTNet [78] adopts a ResNet backbone to extract multi-scale feature maps from the input image. Then, an orthographic feature transform is introduced to map the image-level features to a BEV representation, in order to allow further computation to reason in 3D without perspective effects. To obtain this representation, a voxel grid fixed to the ground plane is generated, then each voxel of the grid is projected onto the feature maps and all features

within the projection are accumulated into the voxel. Finally, the voxel grid is collapsed into a 2D representation by accumulating features along the height direction. As last step, each location of the BEV feature map is processed in order to classify whether there is an object as well as to determine the center, dimensions and orientation of the object.

In MonoGRNet [79] the 3D bounding box estimation process is split into four sequential subtasks performed by a single, unified neural network. In the first step, image features are computed using a VGG16 backbone and 2D bounding boxes are extracted by adopting a two-stage object detection pipeline. Given the 2D boxes, RoIAlign is used to pool the respective set of features which are used for the remaining three steps. First, the depth of each instance is estimated. Then, given the depth, the true coordinates of each instance center are regressed. Finally, the positions of the eight vertices for each box are computed with respect to each box local frame of reference placed in each estimated center.

MultiFusion [80] leverages a pretrained model for monocular depth prediction to compute a depth map for each image. This map is then concatenated to the image itself and fed to a VGG16 backbone followed by an RPN to extract region proposals. Given the 2D proposals locations, RoI Max Pooling is used to extract the corresponding features from the feature map and RoI Mean Pooling is used to extract a feature representation from the point cloud generated from the depth map. These two sets of features are then concatenated and used to classify each proposal and determine their corresponding 3D box.

Deep MANTA[81] detects vehicles via part estimation followed by template matching. First, a standard VGG16 backbone followed by an RPN are applied to the input image to obtain region proposals. Then, these proposals go through two cascaded refinement stages, involving RoIAlign on the proposal coordinates followed by box correction. Besides the correction, the second refinement stage also outputs, for each box, its classification score, a vector of 2D image coordinates corresponding to the object parts (i.e. tyres, headlights etc...) as well as an estimation of the similarity of its 3D dimensions with re-

spect to a set of fixed templates. Follows a final 2D/3D matching phase in which the predicted similarities as well as the 2D object parts locations are used to recover the 3D pose of the object as well as the 3D coordinates of its parts by matching them against a database of templates using the PnP algorithm [82].

MonoPSR [16], builds upon pre-existing high performance 2D detectors and uses LiDAR data as additional information during the training of the model to improve performance. First, a pretrained 2D object detector is applied to the input image in order to compute the image-level boxes. For each detection, a set of features is then extracted by fusing together the full-image features pooled at the box location and a second set of features obtained by applying a ResNet backbone to an image-level crop of the object. These features are then fed to a proposal generation module, which estimates the orientation, dimensions and position of the object. Follows a proposal refinement module, which further corrects the location of the detection. Finally, this information as well as the available LiDAR data during training are used to guide an additional instance reconstruction module to estimate a point cloud representation for each object, which is then used to setup additional auxiliary loss functions.

Chapter 2

Object Detection for Parking Slot Detection

In this chapter I present the proposed deep learning method for parking slot detection from surround view images. I first provide motivation for the research, highlighting the importance of the problem as well as detailing other methodologies adopted in literature and their weaknesses. I then briefly review Faster R-CNN, as it constitutes the baseline object detection model adopted as starting point for this work. I follow up by illustrating the approach and the dataset used for optimization, including the data preparation process used to create it. Finally, I describe the experiments performed to validate the effectiveness and robustness of the system and I discuss the obtained results.

2.1 Prior Art and Motivation

Advanced Driver-Assistance Systems (ADAS) are experiencing a spike in interest from the research community and are currently one the most researched technologies. Among these systems are, for instance, Adaptive Cruise Control, which allows the vehicle to automatically maintain a specific distance from the vehicle in front, or Lane Keeping, which automatically keeps the vehicle

centered in its lane. Another technology frequently available on modern cars is the Parking Assistant which monitors the surrounding space and, once it locates an available parking spot, it assists the driver throughout the maneuver. The localization of free parking space is often performed using sonar sensors by detecting enough unoccupied space to allow the maneuver to be completed successfully. While such a system might work well under the supervision of a human driver, however, it is unsuitable in the context of a fully automated vehicle: sonars are only capable of detecting vacant space, not parking slots directly, and therefore are only useful under the assumption that the vehicle is in proximity of a parking lot. On the other hand, cameras are able to percieve horizontal traffic signs and therefore can enable fully automated parking slot detection.

A lot of research has focused on vision for parking slot localization and occupancy classification. Many algorithms have been developed that exploit static cameras to monitor occupancy in order to manage parking lots [83, 84, 85]. From an autonomous driving system point of view, however, these approaches are unsuitable as they all rely on the a priori knowledge about the location of each slot, information that is unavailable when the cameras are dispaced on a moving vehicle.

Research in the direction of automatic parking detection during navigation started with [86], where color was used as cue to segment parking slot markings directly in the image. More recent approaches perform detection on a Bird's Eye View representation of the image instead, in which horizontal road markings are mostly free of perspective effects: for instance, in BEV rectangular slots always appear rectangular, and line thickness does not depend on the vicinity of the slot to the sensor. Moreover, in order to obtain a complete 360° perception of the surrounding environment, most setups involve multiple calibrated cameras, whose BEVs are then stitched together to obtain surround view images. These representations are used in approaches such as [87, 88], in which parking slots are identified using low level visual features, such as corners and lines. [89] uses boosting [90] in order to classify cross-points between parking-line segments,

and then determines the entry point to each slot from those. The classifier is trained on a dataset comprised of 8600 images, in which the position of each cross-point and the orientation of each slot is annotated. In [91], sobel filters followed by a probabilistic Hough transform are used to extract lines from the surround view images. Then, the available parking slots are detected by exploiting relations between parallel lines.

All the above approaches, however, suffer from a common limitation: they all rely on hand-designed visual features, and therefore they tend to perform well only for the specific and controlled environments for which they are designed. [89], for example, is only able to detect horizontal and vertical slots, so it tends to fail in presence of slanted parking slots on during the execution of the parking maneuver. [91] is relatively robust to different observation conditions, but is unable to detect slanted slots and requires a computationally expensive post processing phase. Color-based approaches such as [86] might fail in presence of occlusions, noise or variations in illumination conditions. To deal with these weaknesses, the method proposed in this thesis performs parking slot detection and occupancy classification from surround-view images directly using a deep convolutional neural network. The core idea is that, by allowing the model to automatically learn from data what features are useful for detecting parking slots, the resulting system should, given enough heterogeneous training data, show higher robustness and adaptability to different observation conditions and slot types. Before going into detail about the approach, I briefly review the 2D object detector Faster R-CNN [9], as it constitutes the foundation for the proposed pipeline.

2.2 Faster R-CNN

As already stated in Section 1.5.1, Faster R-CNN is a two-stage neural network for 2D object detection from images. In particular, for each detected object, it returns:

- its class c_i;

- its axis-aligned 2D bounding box, described by its center coordinates (u_i, v_i) and its dimensions (h_i, w_i).

More specifically, this systems is composed of three main modules:

- a *Backbone Network*, which is used to extract generic feature representations from the input;
- a *Region Proposal Network* (RPN), which estimates a high-recall initial set of potential bounding boxes, called *proposals*;
- a *Detection Head*, which analyzes each proposal in order to determine whether it contains an object, classify the object and compute a correction to the proposal box to make it better fit the object.

Backbone Network As first step, the image is processed by the backbone, which is responsible for extracting a high-level feature representation of the input. The most commonly adopted choice for this network consists of a ResNet model whose weights are initialized via a training on the ImageNet dataset for the classification task. Recently, the ResNet model is often enhanced with a Feature Pyramid following FPN [11], which generates a pyramid of feature maps at different resolutions by progressively upsampling the last feature map produced by ResNet and by fusing it with early layers maps.

Region Proposal Network Given the generated feature map, the RPN estimates an initial set of bounding boxes potentially enclosing regions of interest. To do so, it exploits a set of predefined boxes, called *anchors*, as well as the relationship between each pixel of the feature map and the center of its receptive field in the input image. The chosen anchors usually span multiple sizes and aspect ratio, as to be able to cover the vast majority of potential objects. More specifically, the RPN operates as follows:

- each pixel of the feature map is assigned the same set of k anchors,

except that the anchors are shifted at the center of the corresponding pixel receptive field in the image;

- the RPN maps the feature map, using a 3×3 convolutional layer followed by two parallel 1×1 convolutional layers, into two outputs: a classification map and a regression map. Both maps have the same spatial size as the feature map and $2 \cdot k$ and $4 \cdot k$ channels respectively.

- the classification map contains the classification decision for each set of anchors at each location, expressed as a discrete probability vector $p = (p_0, p_1)$, where p_0 represents the probability that the anchor contains background and p_1 the probability that it contains an object; likewise, the regression map contains the shift and scale corrections to apply to each anchor (u_a, v_a, h_a, w_a) in order to generate the corresponding proposal box (u, v, h, w), encoded as follows:

$$t_u = \frac{u - u_a}{w_a} \qquad t_v = \frac{v - v_a}{h_a} \qquad t_w = \log \frac{w}{w_a} \qquad t_h = \log \frac{h}{h_a} \tag{2.1}$$

In case a feature pyramid is used instead of a single feature map, anchors at different scales are assigned to different pyramid levels: bigger anchors are assigned to low-resolution feature maps, as these maps tend to focus more on the global context of the input; conversely, smaller anchors are assigned to high-resolution feature maps, as these embed more local information. Using bigger feature maps for smaller anchors also means that more of these anchors are present, which improves recall for smaller objects. This architectural choice ensures that objects can be effectively detected even if they appear at considerably different scales in the input. Note that the same RPN model is used to process all feature maps in the pyramid.

Detection Head Given the generated proposals and the feature maps from the backbone, the detection head is tasked to predict the final set of objects. In particular, this second stage operates the following way:

- Non-Maximum Suppression is applied to the proposal boxes to remove redundant detections, using their Intersection-over-Union to compute similarity and the predicted objectness probability p_1 to determine which boxes to keep;

- the top-scoring N proposal boxes are selected, their coordinates are projected onto the feature map and the set of features specific to each one is extracted using RoIPooling. RoIPooling is an operator that aggregates features in a region of arbitrary size into a region of fixed size by quantizing the region, subdividing it into bins and aggregating the information in each bin via a max pooling operation. Similarly to the RPN, in case a feature pyramid is used, each proposal is projected onto the appropriate feature map depending on its size;

- the features pertaining to each proposal are processed by the detection head, which returns a classification decision as well as a refinement for each one. This subnetwork is generally quite simple, encompassing a couple of fully-connected layers followed by two more parallel fully-connected layers that perform classification and refinement respectively. Assuming there are M possible object classes, the output of the classification branch is a discrete probability distribution $p = (p_0, p_1, \ldots, p_M)$, where p_0 represents the probability for the background class, which should be high in case the proposal does not contain any object of interest. The regression branch, on the other hand, estimates a shift and scale correction for *each class*, leading to $4 \times M$ outputs. Which correction to apply is then determined by the estimated probability distribution. The corrections are encoded the same way as they are for the RPN (Eq. 2.1), where (u_a, v_a, h_a, w_a) represent the proposals centers and dimensions;

- a class-wise Non-Maximum Suppression is performed in order to remove duplicates and obtain the final detections.

The additional correction performed in the second stage by the detection

head allows for more accurate detections to be obtained, as the classification and refinement decisions are based on regions of the feature map instead of single pixels, allowing for more information to be used.

2.2.1 Training Procedure

Commonly, all three blocks are optimized simultaneously.

In order to provide the RPN with a balanced set of positive and negative examples, at each iteration only a subset of N anchors is used for training. Commonly, 256 anchors are selected per image, with a ratio of up to $1:1$ between positive and negative anchors. Positive anchors are defined as the anchors which share a sufficiently high Intersection-over-Union with at least one ground truth bounding box. Conversely, if an anchor has low enough Intersection-over-Union with all ground truth boxes, it is labelled as negative. The overall loss function for training the RPN is as follows:

$$L_{RPN} = \frac{1}{N} \cdot L_{cls}\left(p, c^*\right) + \frac{\lambda}{N_{pos}} \left[c^* >= 1\right] L_{reg}\left(t, t^*\right), \tag{2.2}$$

where $L_{cls}\left(p, c^*\right) = -\log p_{c^*}$ is a standard multi-class cross-entropy loss function and the loss function for training the refinement takes the following form:

$$L_{reg}(t, t^*) = \sum_{i \in \{u,v,h,w\}} \text{smooth}_{L_1}(t_i - t_i^*), \tag{2.3}$$

where

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise.} \end{cases} \tag{2.4}$$

Here, c^* represents the index of the ground truth class (in this case $c^* \in \{0, 1\}$), t^* represents the regression targets for box refinement encoded as in Eq. 2.1, N_{pos} is the number of positive anchors and λ is a hyperparameter used to balance the two loss contributions. The Iverson bracket function $[c^* >= 1]$ evaluates to 1 for all positive anchors, and 0 otherwise. This has the effect of

ignoring the regression loss for negative anchor boxes, as they have no associated ground truth box.

To ensure an equally balanced optimization procedure for the detection head, during training the second stage operates slightly differently. Instead of considering the top-scoring N proposals after suppression, a balanced set of positive and negative proposals is used instead. Like for the RPN, positivity and negativity are defined according to the Intersection-over-Union of the proposals with the ground truth boxes. In this case, 64 proposals per image are chosen such that the ratio between positives and negatives is approximately 1:3. The loss function L_{DET} used to train the detection head has the same formulation as that for the RPN (Eq. 2.2), with some minor differences:

- c^*, instead of representing positivity or negativity, now represent the ground truth class index (0 if background);
- since the detection head outputs a correction for each possible class, the regression loss L_{reg} is computed only on the predictions corresponding to the ground truth class.

The final loss for training the system is simply given by the sum of the loss functions for the RPN and the detection head:

$$L = L_{RPN} + L_{DET}. \tag{2.5}$$

2.3 Parking Slot Detector

The method ptoposed in this thesis [19] frames parking slot detection as a 2D object detection problem. In particular, given a surround view image as input, the task is to locate all parking slots visible in the image, classify whether they are vacant or occupied and identify their image coordinates. Differently from the common approach to 2D object detection, however, where each object is described by an axis-aligned bounding box, in this case each parking slot is identified by the image coordinates of its four vertices. Formally, the state for

the i-th parking slot is represented by the vector $S_i = \left(c^{(i)}, q_1^{(i)}, q_2^{(i)}, q_3^{(i)}, q_4^{(i)}\right)$, where $c^{(i)}$ represents the class of the slot (i.e. occupied or vacant) and $q_j^{(i)} = \left(u_j^{(i)}, v_j^{(i)}\right)$, $j = 1, \ldots, 4$ represent the ordered set of vertices for the slot. The reason why the vertex-based representation was chosen is twofold: on the one hand, it allows the network to model all parking slot types, including slanted ones which are not rectancular; on the other hand, it enables the model to operate in scenarios in which the cameras are not perfectly calibrated, which leads to distortions in the generated surround-view images.

To allow for generic quadrilateral prediction, the model structure of Faster R-CNN has been adapted accordingly. A visual representation of the various steps of the proposed pipeline is displayed in Fig. 2.1.

Backbone Network Like in the original model, the backbone processes the input, which is now a surround view image, in order to extract a feature representation. As backbone network of choice I adopted ResNet-34 enhanced by the first upsampling layer of FPN such that the last feature map has resolution 1/16 of that of the input. This resolution allows for enough proposal boxes to be generated without being too computationally expensive to process. At the same time, adopting a pyramid of features at multiple resolutions would bring close to no benefit to the overall performance, as all parking slots are expected to have roughly the same size when represented in bird's eye view.

Anchor-Free Region Proposal Network While an anchor-based region proposal approach might work well when the boxes to be predicted are axis-aligned rectangles, it struggles in case the detections are generic quadrilaterals, as it would require a complex design of the set of anchors, encompassing multiple scales, shapes and orientations [92], to work well. Therefore, inspired by [93], I opted for an anchor-free solution in which the coordinates of each proposal are estimated directly. In particular, the proposed Anchor-Free RPN operates as follows:

- each pixel of the feature map is assigned a specific location in the input image, called reference point, which corresponds to the center of its receptive field (Fig. 2.2a);
- for each reference point, the RPN determines whether it is inside a parking slot and estimates the coordinates of its four vertices as a residual from the reference point location (Fig. 2.2b). More specifically, if $r = (u_r, v_r)$ is the position of the reference point and $q_i = (u_i, v_i)\,,\ i = 1, \ldots, 4$ represent the estimated parking slot vertices, the output is encoded as follows:

$$t = \left\{ t_i = \frac{q_i - r}{\gamma}, i \in \{1, 2, 3, 4\} \right\}, \tag{2.6}$$

  where γ is a normalization constant, empirically chosen to be equal to 50. This logic is implemented as two sibling convolutional branches, a classification branch and a regression branch, each comprised of two convolutional layers. In both branches, the first layer is represented by a 3×3 convolution having 256 output channels followed by batch normalization and ReLU activation. The second layer is a 1×1 convolution with 1 output channel for the classification branch, representing the estimated confidence that the corresponding reference point is inside an instance, and 8 output channels for the regression branch, representing the estimated coordinate residuals t.

Detection Head Given the generated parking slot proposals, the detection head is responsible for estimating the final set of detections from them. The logic is similar to that of Faster R-CNN, with a few modifications:

- all proposal boxes having an estimated confidence below 0.5 are discarded (Fig. 2.1c). Then, Non-Maximum Suppression is applied to the remaining detections in order to remove duplicates (Fig. 2.1d). For computational simplicity and efficiency, NMS is performed using the Intersection-over-Union of the axis-aligned rectangle minimally enclosing each proposal, and in case of IoU ≥ 0.5 the proposal having the higher score is kept;

- the features pertaining to each proposal are extracted by projecting the proposals on the image plane and by applying RoIAlign [64] on the axis-aligned rectangle minimally enclosing each proposal. The decision of adopting RoIAlign over RoIPooling is dictated by need of performing vertex estimation as accurately as possible. Indeed, while in standard 2D object detection slight inaccuracies in the predicted boxes positions can be justifiable, the harsh quantization introduced by RoIPooling could lead to performance degradation in case of parking slot detection, where pixel-accurate estimations are required. RoIAlign circumvents this problem by removing the quantization of bins and performing feature pooling via bilinear interpolation.

- the detection head processes the extracted set of features in order to perform classification and vertices estimation (Fig. 2.1e). This part of the model is comprised by two fully connected layers, having 2048 neurons each, batch normalization and ReLU activation, followed by two parallel fully-connected layers for classification and estimation respectively. The classification layer outputs a discrete probability distribution $p = (p_0, p_1, p_2)$, representing the probabilities of no-slot, vacant slot and occupied slot. The estimation layer adopts the same parametrization as the RPN (Eq. 2.6), with the difference that the residual is now computed with respect to center of each proposal aligned rectangle. Correcting with respect to this center instead of the centroid of the four estimated vertices ensures that the predictions performed by the detection head are consistent with the information that it processes, which is determined by RoIAlign on the aligned rectangles.

2.3.1 Training Procedure

The optimization procedure mostly follows the original work (Sec. 2.2.1).

The RPN is trained by selecting, at each iteration, 256 reference points per image with a ratio of 1:1 between positive and negative samples. If not

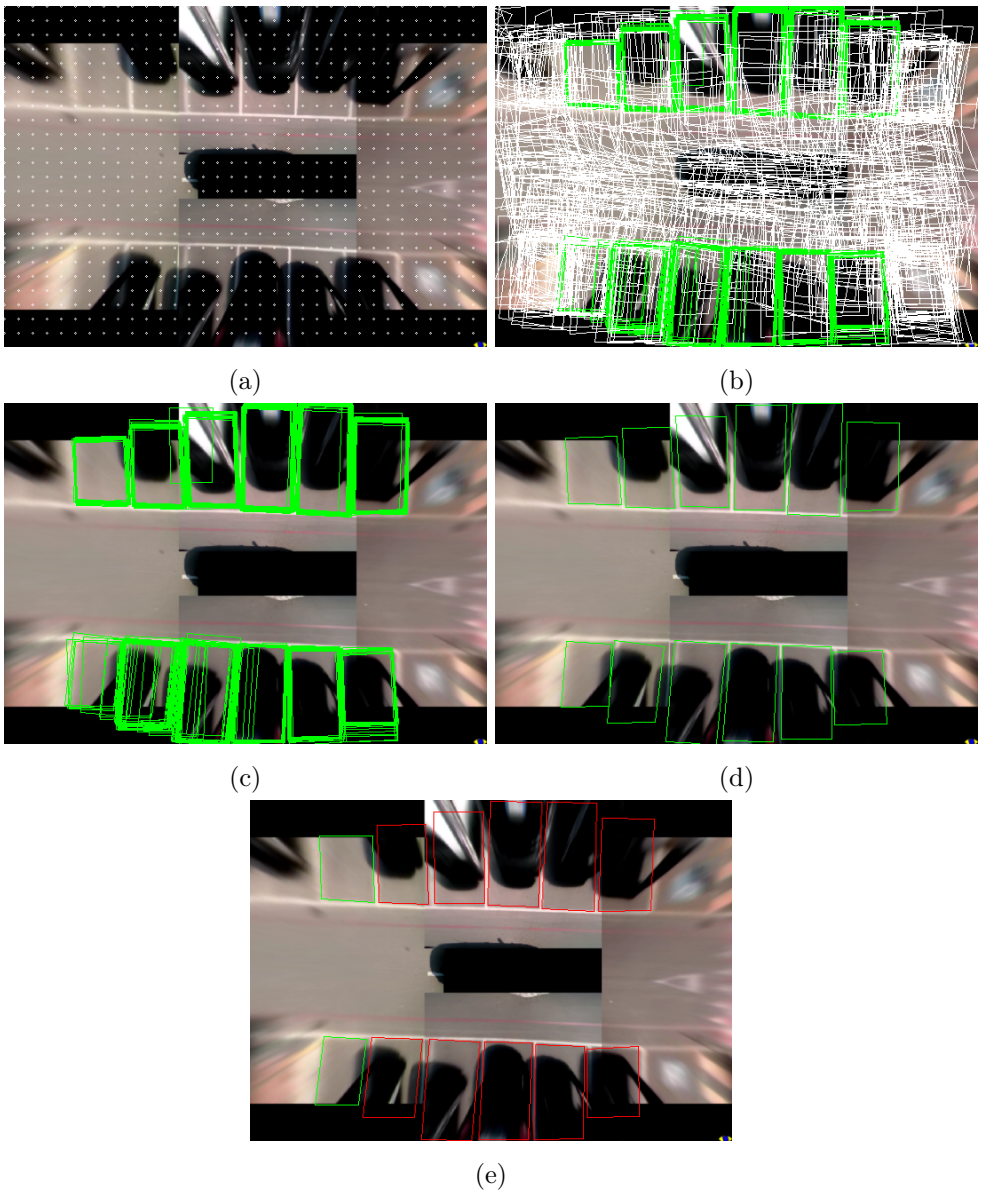


Figure 2.1: Visualization of the intermediate steps of the network. (a): Locations of the reference points on the image. (b) Region proposals estimated from the reference points. The green color indicates proposals having objectness score above 0.5. (c): Pruning of the negative region proposals. (d) Remaining positive region proposals after NMS. (e): result of the detection head on the remaining region proposals.

enough positive samples are present, enough negative samples are selected to reach 256 elements total. Since no anchors are used at this stage, positivity is determined simply by whether each reference point is contained in a ground truth parking slot or not: if it is, it is labelled as positive and associated to that slot; otherwise, it is labelled as negative. The loss function for training the RPN is given by:

$$L_{RPN} = \frac{1}{N} \cdot L_{cls}\left(p_{in}, c^*\right) + \frac{\lambda}{N_{pos}} \left[c^* = 1\right] L_{reg}\left(t, t^*\right), \tag{2.7}$$

where L_{cls} is a standard binary cross-entropy loss:

$$L_{cls}(p_{in}, c^*) = -c^* \cdot \log p_{in} - (1 - c^*) \cdot \log\left(1 - p_{in}\right) \tag{2.8}$$

and L_{reg} follows Eqs. 2.3 and 2.4. p_{in} and t represent, respectively, the predicted confidence probabilites and vertices residuals (parametrized as in Eq. 2.6), while t^* contains the corresponding ground truth residuals, and c^* is set equal to 1 for positive reference points and 0 otherwise. The balancing hyperparameter λ is empirically set to 3.

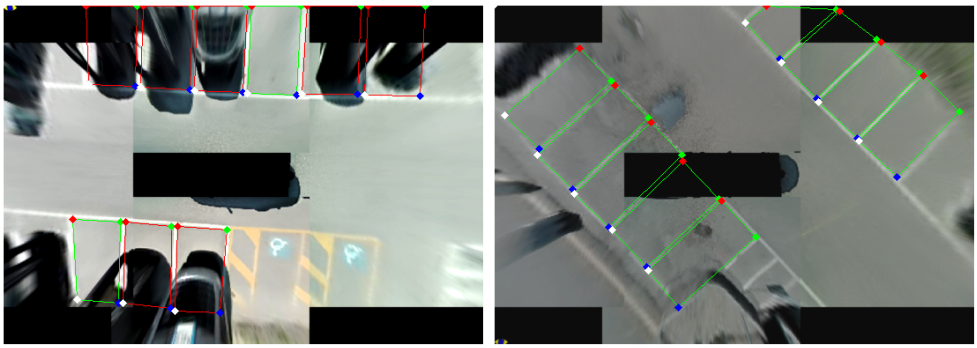
The training of the detection head follows Faster-RCNN, with the only difference that, in this case, a set of 128 proposals per image are used, with a ratio of 1:1 between positive and negative proposals. In this stage positivity is determined by measuring the intersection over union between the minimum enclosing rectangle of each proposal and the minimum enclosing rectangle of each ground truth quadrilateral. In particular, for each proposal, the ground truth box having maximum IoU with it is determined. Then, if the IoU is greater than 0.5, it is labelled as positive and associated with that ground truth. Otherwise it is marked as negative. The loss function L_{DET} for the detection head is equal to Eq. 2.2 with $\lambda = 1$, and the total loss for the system is given by the sum of the RPN loss function and the detection head loss function: $L = L_{RPN} + L_{DET}$.

The model is trained jointly using Stochastic Gradient Descent with momentum 0.9, batch size 16 and learning rate 10^{-3} for 10.000 iterations. During training, data augmentation is employed in order to enrich the data samples

and reduce overfitting. In particular, each input image is subject to both horizontal and vertical flipping, applied independently each with probability 0.5, and is perturbed in brightness, saturation and contrast in the range ±20%, in order to simulate additional observation conditions.

2.3.2 Dataset Construction and Data Preparation

In order to train the model, a small training dataset composed of 467 surround-view images was created and manually annotated with the image coordinates of the corners of each parking slot, as well as the information regarding the occupancy of the slots. The vertices of each slot have been preprocessed in order to ensure a consistent ordering among different examples. More specifically, the vertices are sorted clockwise with respect to the parking slot centroid, identified as the mean of its four corners. Ensuring the consistency of the vertices order is crucial to successful optimization, as the model must be able to attribute a semantic meaning to each one. If ground truth vertices were not ordered, very similar instances would correspond to different training objectives, which would lead to instability and inability to reach convergence. Examples of annotation are visible in Fig. 2.2.



(a) (b)

Figure 2.2: Examples of annotated images. Colors are used to highlight both vertex ordering and occupancy.

To obtain the training images, a vehicle setup with four calibrated and synchronized fish-eye cameras was used to collect several sequences, both of road scenes and parking lot areas. The images from each camera were then cast to a Bird's Eye View representation using the cameras projection model and stitched together using the calibration to create the surround-view images. Meaningful frames depicting different scenarios, parking slot types and observation conditions were then manually selected for annotation. To avoid biasing the network towards situations in which parking slots are always present, 167 of the 467 chosen frames contain no parking slots. The generated images have shape 1100×900, which is downsampled to 544×384 before being given as input to the model. The reason for downsampling the input is twofold: on the one hand, reducing the spatial size reduces the computational and memory costs of the system considerably. On the other hand, the chosen resolution is divisible by 32 on both dimensions, which simplifies the computation of the reference points positions.

2.4 Experimental Results

To validate the effectiveness of the presented approach, the trained model was tested on new sequences acquired under different observation conditions and containing parking lots unobserved during training. Some qualitative results are visible in Fig. 2.3. As can be observed, despite the extremely limited training set, the model exhibits a remarkable capability to generalize to new, unseen scenes and it is able to correctly identify parking slots that are similar to those observed during training. In particular, the network is currently capable of handling parking slots displaying different patterns (Fig. 2.3b) as well as rotated slots (Figs. 2.3c, 2.3d). Moreover, despite the fact that the training set contains only as few as 20 examples of slanted parking slots, the network is able to detect them with acceptable accuracy, as shown in Fig. 2.3e. The system is also able to withstand noise in the data: in Figs. 2.3c, 2.3d and 2.3e some dirt is visible in the lenses. Also, the cameras are not perfectly calibrated,

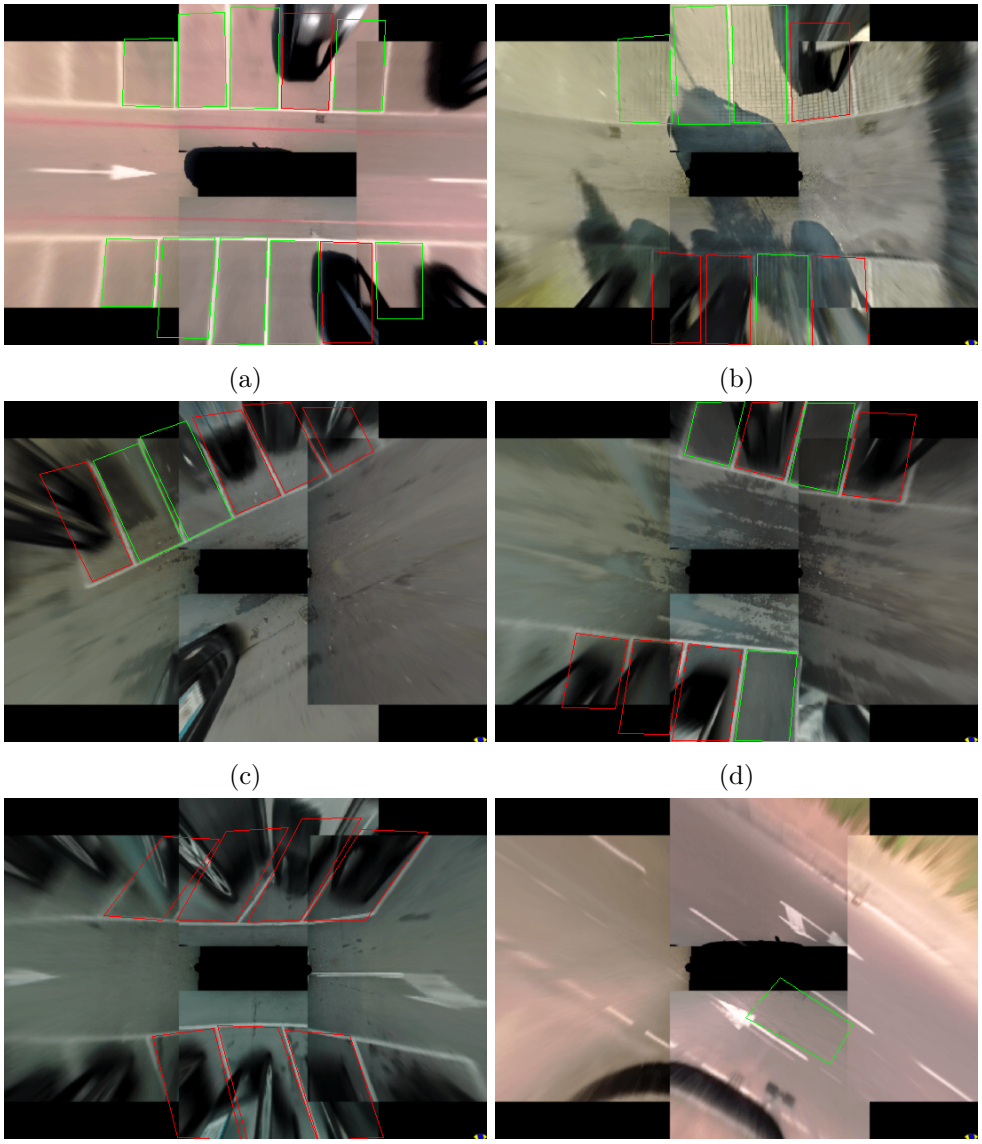


Figure 2.3: Results of the network on different types of parking slots under different observation conditions: (a) the most common scenario. (b) Different pattern. (c-d) Different rotations. (e) Slanted parking slots. (f) Failure case.

which leads to misalignments between the four bird's eye view images. Nevertheless, the network is able to predict the observable slots well enough. The scarsity of training data, however, might lead to incorrect predictions, where unobserved horizontal traffic signs and patterns are erroneously interpreted as parking slots, such as in Fig. 2.3f.

The proposed system is able to run at over 13 frames per second (fps) on a NVIDIA Geforce GTX 1080 GPU.

2.4.1 Semantic Shift Problem

In the data preparation section (2.3.2) I mentioned the importance of ensuring a consistent ordering among the four vertices of each ground truth bounding box, as the network must be able to associate a semantic meaning to each point (e.g. the first prediction is the top-left point, the second is the top right and so on) in order to be able to converge. In order to guarantee this consistency, the vertices were sorted in a clockwise order with respect to the centroid of each bounding box. There are, however, some specific observation angles for which the result of the sorting rule changes abruptly, causing what I call a *semantic shift* (see Fig. 2.4a for a schematic representation of the problem). As a consequence, the network exhibits erratic behavior when asked to perform predictions for instances that are very close to this critical angle (see Figs. 2.4b, 2.4c, 2.4d). This is due to the fact that the model is unable to identify which is the correct order of the points and, as a result, tends to predict coordinates that are in between the right ones. Note that this problem is not due to the specific ordering rule chosen, but rather to any ordering rule. Different ordering strategies might correspond to different configurations at which the semantic shift occurs, but the overall problem remains. The hope is that, by increasing the number of training examples close to the semantic shift, the network can learn to handle these cases more effectively and narrow down the range of angles for which the confusion happens. Of course, the ideal way to handle this problem would be to adopt a representation for the output that does not depend on any specific ordering, bypassing the semantic shift altogether.

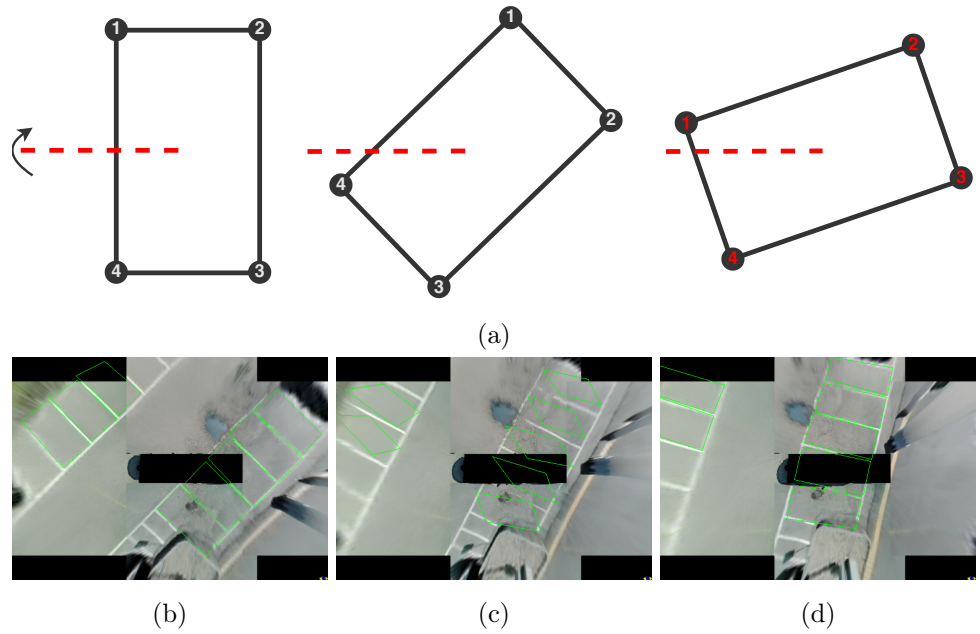


Figure 2.4: Semantic shift problem. (a) Schematic illustration of the problem. (b) Output before the shift angle. (c) Output around the shift angle. (d) Output after the shift angle.

One such representation might take ispiration from the very recent work Poly-YOLO [94], in which the authors extend the latest YOLOv3 model to also predict the semantic mask for each detected object by interpreting each mask as a set of vertices that are estimated using a polar grid.

2.4.2 Ablation Study

To validate the choice of adopting a two-stage detection approach for added localization accuracy, I carried out an ablation study that is made up of two distinct experiments:

- first, I removed the second stage, entrusting the RPN to perform detection directly. To achieve this, the classification branch of the RPN was modified so that it predicts the class of each reference point (e.g. background, occupied, vacant) instead of just determining whether each reference point is contained in a parking slot. The cost function for the classification was changed accordingly to a standard multi-class cross-entropy formulation;
- second, the sampling procedure for picking a balanced set of foreground and background examples at each iteration was removed. Instead, all reference points are used to compute the loss function L_{RPN}.

To evaluate each experiment, a small test set of 107 unobserved parking spaces was annotated.

As evaluation metric I chose the Average Precision (AP), utilized to evaluate Object Detection performance in the PascalVOC benchmark [95]. To compute this metric, the outputs of the system are ranked according to their confidence and are then used to determine the precision/recall curve. The AP value is then obtained by computing the mean precision value at a set of 11 equally spaced recall intervals. To determine whether it is a true or false positive, each prediction is checked against the ground truth according to a rule. Commonly, in traditional Object Detection this rule consists in the Intersection-over-Union

between the detection and the ground truth boxes. More specifically, in order to be considered a correct detection, an output of the model must have an IoU above a certain threshold with at least one ground truth element of the same class as the prediction. Duplicate detections of the same ground truth instance are handled as false positives (e.g. if the same ground truth object is detected 3 times, one is considered as true positive and the rest are false positives). Differently from the training procedure, where the IoU computation has been approximated using the minimum enclosing rectangle for efficiency, here the computation is exact, as performance is not a concern.

The results of this study are shown in Tab. 2.1. The displayed mAP (mean AP) scores are obtained by averaging the AP values for the vacant and occupied classes. The subscript indicates the IoU threshold used to determine the positivity or negativity of each detection, while no subscript means that the correspoding values are obtained by averaging the mAP values over multiple thresholds, from 50% to 95% in intervals of 5%.

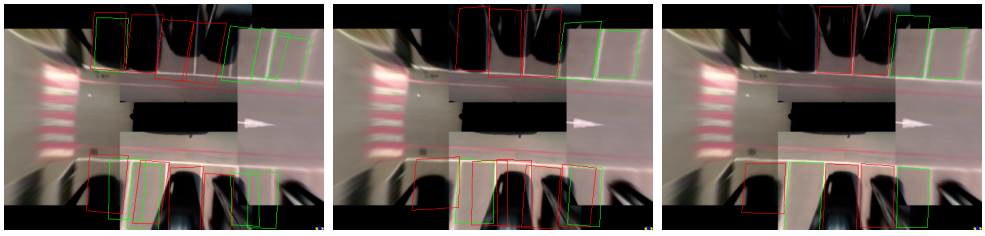
Point sampling	Second stage	mAP	mAP_{50}	mAP_{70}
✓	✓	**44.9**	**60.2**	**54.8**
✓		20.1	36.5	25.4
		14.1	29.6	17.4

Table 2.1: Results over the test set. mAP_{50} and mAP_{70} represent the mean Average Precision using IoU threshold of 50% and 70%, respectively. mAP represents the mean Average Precision averaged over multiple thresholds (from 50% to 95%, in intervals of 5%).

As can be observed, the introduction of the second stage to the pipeline accounts for most of the performance gain of the system. This difference in performance can be justified by the fact that, when the second stage is not used, the predictions are no longer based on ad-hoc set of features extracted by RoIAlign, but rather on generic patches of the feature map. These patches

are not as accurately localized as feature crops, are less informative and have a smaller receptive field, which hinders the ability of the model to produce accurate vertex predictions.

Removing the sampling strategy for the reference points during training further degrades the quality of the results. The reason for this is twofold: on the one hand, using all reference points at each training iteration inflates the classification loss with background samples, slowing down progress for positive reference points; on the other hand, the sampling process provides stochasticity during optimization, meaning that the same input provides different feedback every time it is presented to the model. This has a regularizing effect, favoring generalization especially when the training data is limited. Moreover, the improvement in performance induced by the sampling strategy as well as the second stage is more pronounced at higher IoU thresholds (i.e. mAP_{70}), which is representative of a better localization accuracy of the full model. An example of the different test-time behavior of the model in the three cases can be observed in Fig. 2.5.



(a) (b) (c)

Figure 2.5: Results of the ablation study. (a): Network without the detection head and using all reference points. (b): Network without the detection head using the usual sampling strategy for the RPN. (c): Full pipeline

It is worth noting that the blurring at the edges of the images caused by the bird's eye view transformation, might lead to detection instability in those regions, resulting in mAP scores for the full model that are not representative of its true performance. Moreover, some false positives might still be detected

due to the network being exposed to unobserved road markings (e.g. Fig. 2.3f). These detections, however, tend to last no more than a couple of frames, and can therefore be filtered out easily in a post processing phase. Such uncertainties of the model are also very likely to be explained away by adding more heterogeneous samples to the training data, which is currently quite limited.

2.4.3 Model Simplification and Sparsification

The proposed full two-stage pipeline is comprised of a total of 56.2 million parameters, distributed as follows:

- the feature extractor, which consists of FPN with ResNet-34 backbone, contains 23.9 million parameters;
- the RPN contains 2.36 million parameters;
- the fully-connected detection head contains the remaining 29.9 million parameters.

In this section I illustrate two experiments aimed at obtaining a lighter model (i.e. a model with a reduced number of parameters) and I analyze the impact that the proposed modifications have on the performance of the system.

The first thing to notice when looking at the distribution of weights within the model is that just the detection head comprises more that 50% of the total number of weights. Such a high number of parameters is mostly due to the first fully-connected layer, which is tasked to map an input of size $7 \times 7 \times 256$ down to 2048 dimensions. Therefore, to lighten the model I propose a fully-convolutional variant to the detection head: this structure processes the 7×7 input returned by RoIAlign using stacked convolutional layers that progressively enlarge their receptive field to cover the entire input, and only contains 1.8 million parameters, a fraction of those contained in the original architecture. More specifications about the two variants and their structure are displayed in Tab. 2.2.

	Fully-Connected		Convolutional	
Features	2048×12544 2048×2048	29.9M	$256 \times 256 \times 3 \times 3, 0$ $256 \times 256 \times 3 \times 3, 1$ $256 \times 256 \times 3 \times 3, 0$	1.77M
Classifier	3×2048	6.14K	$3 \times 256 \times 3 \times 3, 0$	6.91K
Detector	8×2048	16.4K	$8 \times 256 \times 3 \times 3, 0$	18.4K

Table 2.2: Comparison between the original fully-connected architecture of the detection head (left) and its convolutional counterpart (right), displaying the layers involved and the corresponding number of parameters. Fully-connected layers are expressed in the format $C_{\text{out}} \times C_{\text{in}}$, where C_{in} and C_{out} represent the number of input and output channels. Convolutional layers are displayed as $C_{\text{out}} \times C_{\text{in}} \times k \times k, p$, where k is the kernel size and p represents the number of rows and columns of zero-padding applied to the input.

To further simplify the system, I propose to sparsify the model. Sparsification consists of forcing a certain percentage of the weights of the network to be equal to 0 while trying to preserve the original performance as much as possible. Different techniques have been introduced to perform model sparsification [96]. The approach adopted in this work is quite straightforward and operates in the following way:

0. the chosen model is trained until convergence normally. A target sparsification percentage $S \in [0, 100]$ is chosen;

1. for each layer of the model, the weights of the layer are sorted according to their magnitude (i.e. their absolute value). Then, the $S\%$ of weights having the lowest magnitude are set to 0. Note that this process involves both the backbone network, the RPN and the detection head simultaneously and is only applied to the weights of the convolutional and fully-connected operators, ignoring biases;

2. the model obtained from step 1. is trained normally for one iteration;
3. if training has not converged yet, return to step 1., otherwise go to step 4.;
4. the final, sparsified model is obtained by performing step 1. one last time, in order to force $S\%$ of the weights to be equal to 0.

This scheme assumes that the magnitude of each weight is a good approximator of its importance within the model. It is also simple, requiring no complex logic or additional hyperparameters aside the sparsification percentage S. Applying the sparsification procedure to each layer individually is crucial to avoid situations in which all the weights of a layer are set to zero, which would compromise the optimization process and cause divergence. These situations are not unlikely, as weights of different layers are usually drawn from different distributions, and therefore some layers might contain weights that have, on average, lower magnitude compared to other layers. Also, allowing deactivated weights to receive updates at each iteration opens up the possibility for the set of zeroed out weights to change over time and to recover deactivated weights, if needed.

The benefits that model sparsification brings are twofold: on the one hand, the resulting models can be stored more efficiently and occupy less memory, which is especially advantageous when memory requirements are strict. On the other hand, by adopting dedicated hardware or sparse representations, such as those introduced in Sec. 1.6.1 for dealing with sparse voxel volumes, zero weights can effectively be skipped during computation, leading to considerable runtime improvements. Note that, differently from voxel grids, in this case sparsity is a property of the model, not the input.

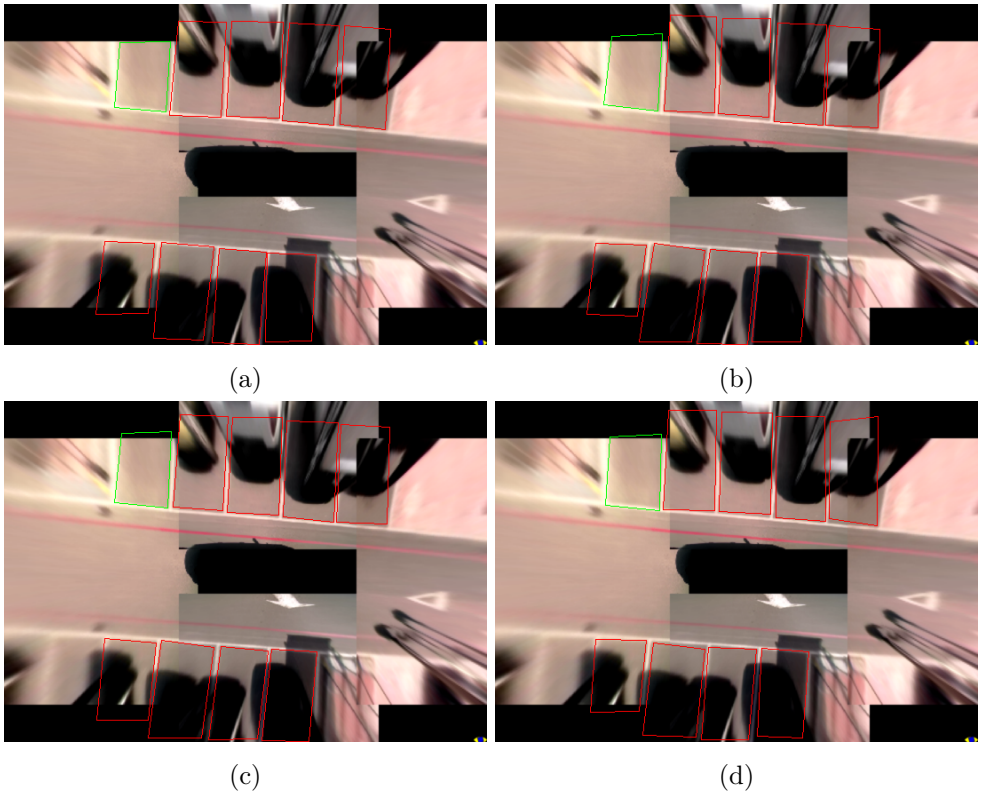
The results obtained from these experiments are summarized in Tab. 2.3. As expected, the convolutional detection head exhibits worse performance compared to its fully-connected counterpart. Such difference is especially marked for very high IoU thresholds (i.e. mAP_{90}) implying that, on average, the convolutional variant returns detections having lower localization accuracy. It is

also interesting to note that the performance degradation is more pronounced for mAP_{50} compared to mAP_{70}, which partly contradicts the previous statement. A possible explanation for this behavior might be that the convolutional model has more difficulty handling noisier cases, such as blurred regions and distorted or occluded areas, compared to the FC model. When a higher overlap is required in order to label a detection as correct (i.e. mAP_{70}), however, the FC model also falls short in predicting hard examples, which leads to a greater performance loss between mAP_{50} and mAP_{70} compared to the convolutional model. Conversely, the latter appears to have comparable performance on easier examples, as proven by the smaller gap for the mAP_{70} case, despite being slightly less accurate overall as demonstrated by the faster drop when working with mAP_{90}.

FC	Conv	Sparsity	mAP	mAP_{50}	mAP_{70}	mAP_{90}	params
✓		0%	**44.9**	**60.2**	**54.8**	**15.6**	56.2M
	✓	0%	38.7	55.6	52.1	6.39	28.1M
	✓	50%	41.1	56.5	54.3	8.56	14.0M
	✓	85%	39.4	55.4	53.1	6.34	4.2M

Table 2.3: Results of the model simplification and sparsification experiments. FC and Conv are used to indicate the original and the convolutional detection heads respectively. The sparsity column indicates the percentage of model weights set to zero by the sparsification process. Note that such sparsification is applied to the whole model, and not only on the detection head.

About sparsification, results are shown for two different sparsity values: 50% and 85%. In particular, to aid convergence, the 85% sparse model is obtained by further sparsifying the 50% sparse model. Surprisingly, the 50% sparse model convincingly outperforms the original, almost matching the FC model on the mAP_{70} metric despite having $1/4$ of the total parameters. This result can be explained by keeping in mind two aspects about the system:



(a) (b)

(c) (d)

Figure 2.6: Output of the different versions of the model on the same input. (a): Original fully-connected model. (b) Convolutional model. (c): 50% sparse convolutional model. (d): 85% sparse convolutional model.

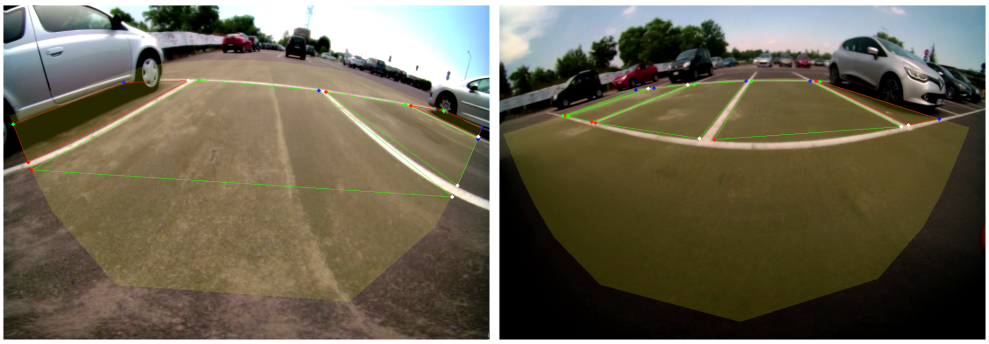
first, the overall network architecture is quite large for the complexity of the data it is applied on; second, the training set is very limited as it contains just below 500 samples. As a result, it is not unreasonable to assume that the hidden representation learned by the model might have a tendency to overfit the training data. Forcing the model to use only half of its weights could encourage more general representations to be learned, which translates to better generalization, while still retaining, given the model size, enough capacity to explain the data. Also, the process of repeatedly setting weight values to 0 during the sparsification procedure introduces additional stochasticity to the optimization, which acts as a deterrent to overfitting. The same benefits are retained when further sparsification is performed, allowing the 85% sparse model to perform on par with the original one despite having approximately 1/7 of its weights (and 1/14 with respect to the FC architecture). Relative to 50%, however, the performance begins degrades, likely due to the limit of minimum number of required parameters being approached. An example of output of the four different network variants on the same input is shown in Fig. 2.6: it can be observed that, despite the difference in number of parameters, the overall quality of the detections is comparable.

The decision to adopt ResNet+FPN as backbone of choice was dictated mostly by its availability and the fact that it constitutes arguably the most commonly used setup for Faster R-CNN based systems. The results obtained during the sparsification experiments, however, highlighted how such a model might be oversized with respect to the complexity of the distribution of inputs it is applied on. It would be therefore interesting to investigate alternative design choices involving lighter models to begin with, such as MobileNets [51].

2.4.4 Prediction Directly on Spherical Images

To further test the adaptability and effectiveness of the proposed approach to parking slot detection, I performed additional qualitative experiments where the model operates on the original images instead of their surround view reconstruction. More specifically, such images are obtained by projecting the raw

data acquired by fish-eye lenses on a hemisphere using a spherical projection model. To obtain the ground truth parking slots in this new representation, I directly reused the previously generated annotations on the bird's eye view images, inverting the transformation to obtain their corresponding coordinates in the spherical images. The generated bird's eye view images, however, depict only a limited portion of the observed space, as far away regions are cropped out due to being too noisy. As a result, since the annotations are reused from the bird's eye view case, there exist parking slots that are clearly visible in the spherical images which are not labelled; also, parking slots that are cut off in the bird's eye view but are fully visible in the spherical images would have their annotations also cut off (see Fig. 2.7 for an example). If such data were to be used directly to train the system, it would lead to optimization instability and suboptimal convergence, as the model would receive contraddictory training signal.



(a) (b)

Figure 2.7: Projections of the annotations generated on surround view images onto the corresponding spherical images. The yellow polygon is used to highlight the portion of image visible from its bird's eye view.

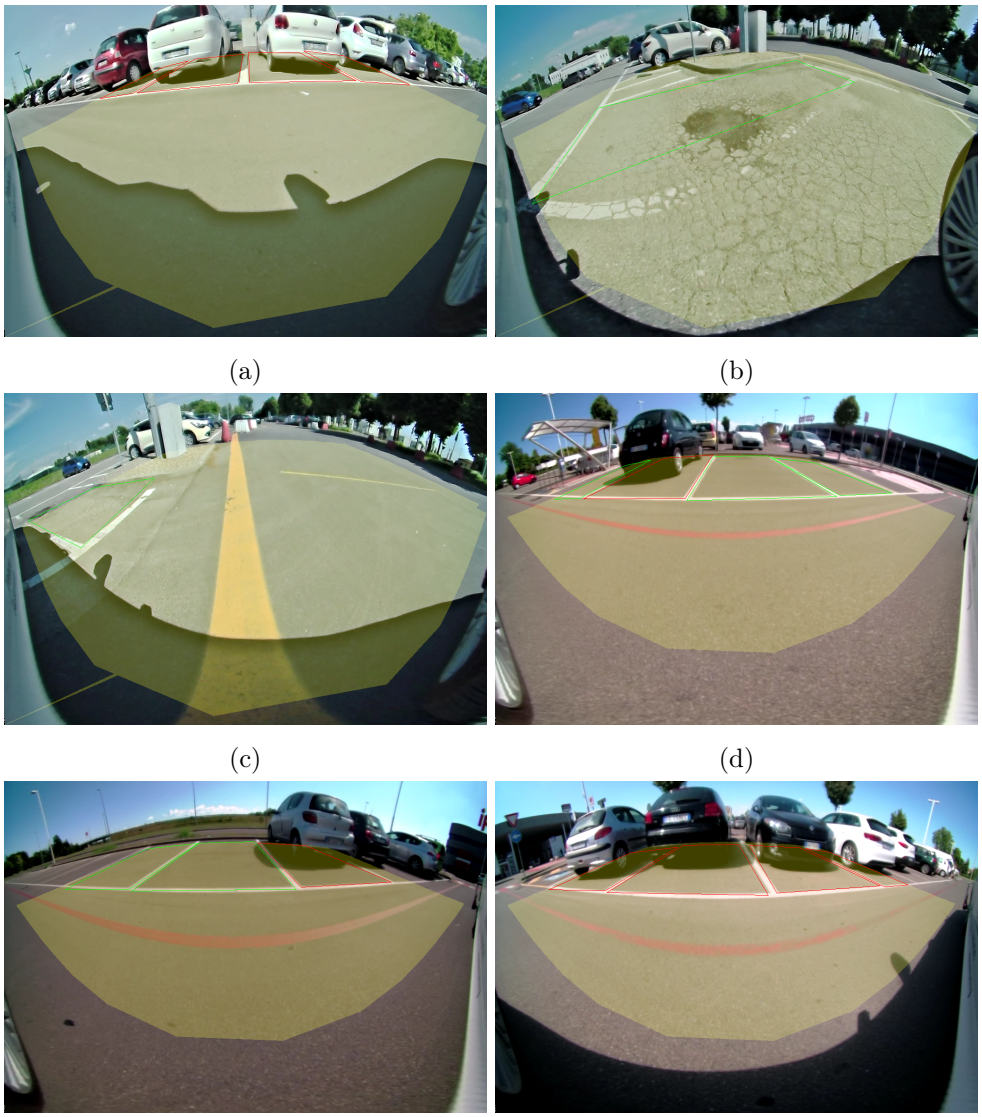
To overcome this limitation, each input image is appended a fourth channel which is a binary mask highlighting the portion of the image that is visible in its corresponding bird's eye view representation. This solution allows the

model to have knowledge about regions that are valid for prediction while being considerably cheaper to implement than simply annotating all the missing slots. An alternative solution to the binary mask could consist in simply zeroing out all pixels that are not visible in bird's eye view. This approach, however, is suboptimal as it would deprive the model from being aware of contextual information that might prove useful for the final task.

Overall, the resulting training set consists of 1868 images total, as each original surround-view image is generated from a total of 4 spherical images. Each spherical image has an original resolution of 1024×992, which is cropped to 1024×704 by removing the rows corresponding to the sky, before using it as input. The adopted model is identical to the one used on surround view, with the exception that the first layer of the backbone is modified to accept 4-channel inputs.

Qualitative results on unobserved parkings scenes are visible in Fig. 2.8. It can be seen that the model is capable of handling relatively well parking slots observed from very different point of views, and therefore characterized by very different shapes, sizes and appearances due to the perspective projection. Moreover, the network appears to have properly learned the semantic meaning of the input binary mask, as it only returns detections within its area.

Note that the experiments on spherical images were conducted exclusively to evaluate the performance and robustness of the proposed system to different conditions and points of observations. Indeed, operating on spherical images (or even pinhole ones) instead of surround view representations is suboptimal for the task of parking slot detection, for several reasons. Parking slot markings, and road markings as a whole, are generally well-behaved in bird's eye view as they are mostly free of perspective effects and preserve their shape and size independently from their distance from the observation point. Also, the bird's eye view projection naturally eliminates all information that lies above the chosen plane, which is mostly useless for the task. For both of these reasons, surround view images represent a much more suitable domain for the task, which leads to better model optimization and increased performance. Another



(a) (b)

(c) (d)

(e) (f)

Figure 2.8: Results of the model on spherical images. It can be noted that the network has learned to utilize correctly the information about the field of view.

non negligible advantage of surround view is that it allows to cover a field of view of 360° with a single input, while at least four inputs would be required in cases spherical images are used. This translates into higher computational requirements, as every image would need to be processed by the model to obtain a 360° aware detection.

2.5 Discussion

In this chapter I presented an end-to-end deep learning-based approach to parking slot detection and occupancy classification on surround-view images. More specifically, I built upon the existing 2D object detection framework Faster R-CNN, redesigning it to allow for generic quadrilateral prediction instead of axis-aligned bounding box estimation.

To train and evaluate the system, two small datasets, containing 467 and 107 surround-view images respectively, were collected and manually annotated with the location and the occupancy of visible parking spaces. The system displayed promising results, exhibiting a remarkable capability to adapt to unseen scenarios containing parking slots of the same type as those observed in the training set while being robust to noise and misalignments between the stitched images that constitute the surround-view representations. It also showed the ability to function properly on an entirely different and more difficult domain, as proven by its effectiveness when applied on the native spherical images. Model simplification and sparsification experiments highlighted the fact that the proposed model is capable of preserving comparable performance by actively using only 1/14 of its total number of trainable parameters, which leaves plenty of room for efficiency improvements.

Chapter 3

Monocular 3D Object Detection via Generalized Intersection-over-Union Minimization

3.1 Prior Art and Motivation

A key challenge in ADAS and autonomous driving systems consists of perceiving the surrounding environment and locating obstacles in it, such that planning and control can be performed accurately. Particularly critical is the detection of moving entities such as cars, pedestrians and cyclists, as they pose a major challenge for safe navigation.

Amongst the most researched solutions to 3D object detection are the LiDAR-based ones, as these kinds of sensors are capable of providing a very accurate, albeit sparse, reconstruction of the surrounding environment. These sensors, however, are generally quite expensive, often making up a big fraction of the total cost of the vehicle. As a result, cameras are usually adopted as a cheaper, more consumer-friendly, alternative to perception. Cameras have the

advantage of perceiving richer information compared to LiDAR, as it is denser and contains color, and can be used to reconstruct the geometry of the environment by adopting stereo setups in conjunction with disparity computation methods [97, 98, 99], albeit not as accurately as LiDARs, especially at long ranges. This information can then be used either implicitly [100, 76, 17] or explicitely [77] to estimate the 3D locations of the objects of interest.

Perhaps the most interesting variant of 3D object detection is, however, the monocular one. Here, the task is to determine a 3D bounding box for every object of interest using as input only a single image. This problem is evidently underconstrained, as a single image alone does not provide enough information to determine the overall scale of the scene and, therefore, the distance of the objects from the sensor. To determine the depth, additional information about the scene is required, such as the a priori knowledge about the dimension of the observed objects. Most contemporary state of the art monocular systems leverage the availability of this knowledge, which often comes in the form of ground truth 3D bounding boxes, to train deep models, with the objective of implicitly encoding the relationship between object appearance on the image plane and the corresponding position in the world within its parameters, such that the model can be used to perform detections in new scenarios. Obviously, for such a system to be able to function accurately, it requires a considerable amount of training data, and the new environments that it is exposed to must belong to a domain that is similar to the one it is trained on. For instance, if the camera intrinsic parameters change from the training set, the system is unlikely to produce accurate localizations, as the learned underlying mapping between appearance and position is no longer valid for the new camera model. Similar problems arise if the camera is positioned differently or if the portrayed objects are visually very different.

Due to the difficulty of this task, most current 3D object detection frameworks opt for dedicated models that integrate 3D reasoning mechanisms directly into their architectures [15, 78, 79, 81, 16], in the hope that the resulting system learns features that are more suitable for 3D tasks and generalize bet-

ter to new situations. Please refer to section 1.6.2 for a more in-depth review of such approaches. Conversely, in the proposed method, I argue that explicit 3D reasoning directly encoded into the network structure is not mandatory for good monocular 3D detection performance, as long as the underlying model has sufficient capacity. To this end, I propose an extention to the 2D detector Faster R-CNN in which a small subnetwork is added to the detection head to perform 3D box estimation. This subnetwork is simple, does not contain any kind of explicit 3D reasoning in its structure and is trained jointly with the rest of the model. See Fig. 3.1 for a schematic representation of the proposed system.

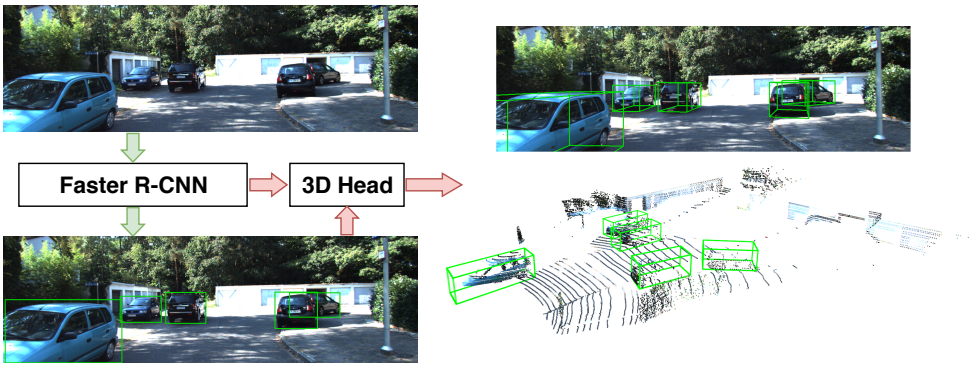


Figure 3.1: **Overview of the proposed 3D detection pipeline**: I extend Faster R-CNN with an additional module responsible for estimating 3D bounding boxes given the 2D detections. This extra module is trained end-to-end with the rest of the network using a novel loss based on the Generalized Intersection-over-Union.

Commonly, 3D estimators are trained via a loss function that minimizes the error between the predicted box parameters (i.e. center, dimensions, orientation) and their corresponding ground truths directly. Instead, I propose a novel objective function that allows to reason in terms of boxes as a whole via the minimization of an approximation of their Generalized Intersection-over-

Union [20]. The experiments show that this formulation leads to considerably better results, likely due to better feature representations induced by a more suitable choice of the loss function.

3.2 Baseline Model

As already stated in Sec. 3.1, the proposed method consists of an extention to the vanilla Faster R-CNN model for 2D detection (refer to Sec. 2.2 for an illustration of this model).

More specifically, as backbone of choice I adopt the standard FPN built upon an ImageNet-pretrained ResNet-50 model, and I extend it, following [63], with an additional downsampling stage consisting of a 3×3 convolution having stride 2 applied to the last feature map returned by ResNet. Formally, the resulting feature extractor generates a feature pyramid comprised of feature maps at 5 different resolutions. These maps are usually labelled as P_2 to P_6, where P_l identifies the feature map having resolution $1/2^l$ of that of the input.

The Region Proposal Network follows the original implementation. To handle objects having different sizes, the RPN comprises 5 different anchor scales having areas $\{16^2, 32^2, 64^2, 128^2, 256^2\}$ which are assigned to the levels P_2 to P_6 of the feature pyramid. Each scale is made up of three different anchors having aspect ratios $\{0.5, 1, 2\}$, for a total of 15 anchors over the entire pyramid.

Likewise, the detection head follows standard procedure. As pooling method I adopt RoIAlign, which extracts a 7×7 fixed-size feature map from the pyramid for each of the top 300 scoring proposals (post NMS) returned by the RPN. These features are then propagated to the detection head for object classification and 2D box refinement. Again, in order to handle objects having different sizes, each proposal is assigned to the proper level of the feature pyramid before performing RoIAlign, according to the following rule:

$$l = \left\lfloor l_0 + \log_2 \left(\frac{\sqrt{w \cdot h}}{224} \right) \right\rfloor. \tag{3.1}$$

Here, w and h represent the width and height of the region proposal and l_0 represents the level in the feature pyramid that a proposal having area $w \cdot h = 224^2$ should be mapped to. Following the original implementation of FPN, I set $l_0 = 4$.

3.3 3D Detection Module

To allow the system to perform detection of 3D bounding boxes, I propose to extend Faster R-CNN with an additional 3D module. In particular, given the final 2D detections produced by the detection head, a second RoIAlign step is performed to extract their specific sets of features, following the same assignment rule illustrated in Eq. 3.1. Then, given each 2D detection and its corresponding set of features, the 3D module is responsible for estimating the 3D bounding box $\mathbf{B} = (x, y, z, h, w, l, \theta)$ corresponding to that object, where (x, y, z) are its center coordinates with respect to the camera frame of reference, (h, w, l) are its height, width and length respectively and θ is its orientation, expressed as a rotation angle around the camera y-axis. See Fig. 3.2 for a bird's eye view illustration of the targets to be estimated.

In order to simplify and stabilize training, these values are not estimated directly by the 3D module, but are rather encoded as follows.

Object Dimensions To estimate the object dimensions, the detection head outputs the following quantities:

$$\left(\log \frac{h}{\bar{h}}, \log \frac{w}{\bar{w}}, \log \frac{l}{\bar{l}}\right), \tag{3.2}$$

where $\bar{h}$, $\bar{w}$, $\bar{l}$ represent class-specific prior values obtained by averaging the dimensions of each ground truth object across the entire training set. This formulation allows to frame the estimation of the dimensions in terms of a relative correction, where negative values correspond to a reduction in size with respect to the prior and positive values to an increase in size.

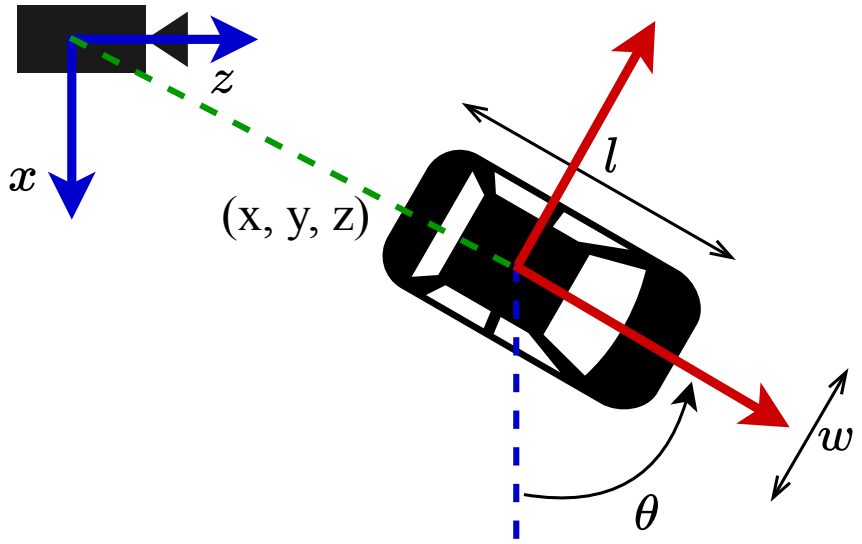


Figure 3.2: Visualization of the 3D detection targets from a bird's eye view perspective.

Object Center Inferring the position of the object directly from the feature crops might prove problematic, as positional information is lost when performing RoIAlign. Therefore, the center estimation problem is decomposed into two: the estimation of the center depth and the estimation of the center projection onto the image plane. Particularly, the center depth is encoded as follows:

$$\frac{z - \mu_z}{\sigma_z}, \tag{3.3}$$

where z represents the true depth and μ_z and σ_z represent the mean and standard deviation for depth values computed across the training set. This ensures that the range of ground truth values follows a distribution with zero mean and unit variance, simplifying the optimization process. To estimate the coordinates of the center projection, the following representation is adopted:

$$\left(\frac{u - p_u}{p_w}, \frac{v - p_v}{p_h} \right), \tag{3.4}$$

where (u, v) represent the image coordinated of the projected center (x, y, z), (p_u, p_v) is the center of the estimated 2D detection and (p_w, p_h) its dimensions on the image plane. In other words, the projected center prediction is framed as a correction with respect to the 2D box center, normalized by the box size. Having estimated the center projection (u, v) as well as its depth z (both obtainable by inverting the encodings above), the true object center in 3D can be obtained by exploiting the pinhole camera model:

$$x = \frac{z \cdot (u - u_0)}{f}, \tag{3.5}$$

$$y = \frac{z \cdot (v - v_0)}{f}, \tag{3.6}$$

where f represents the focal length of the camera and u_0, v_0 its principal point.

Object Orientation Similarly to the object center case, the object orientation θ is not directly observable from feature crops alone: indeed, identical

objects having the exact same orientation but a different position appear differently on the image plane due to the perspective projection. See Fig. 3.3 for an illustration of the problem.

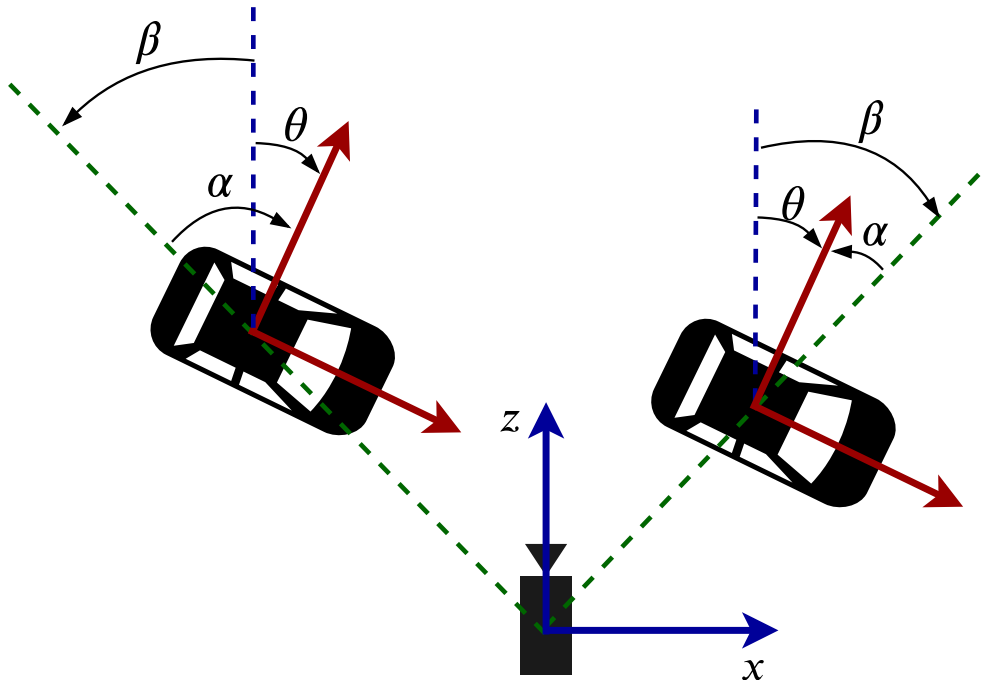


Figure 3.3: The two objects clearly appear differently once projected onto the image plane, despite having the exact same orientation θ. To avoid ambiguities, I estimate the observation angle $\alpha = \theta - \beta$ instead, where $\beta = \text{atan2}(-x, z)$, as this angle correlates with image appearance.

Therefore, instead of predicting the orientation directly, following [101] I predict the so-called observation angle:

$$\alpha = \theta - \text{atan2}\left(-x, z\right). \tag{3.7}$$

This transformation compensates for the perspective effect by considering the angle with respect to the ray that goes through the center of the object instead

of its absolute orientation More specifically, in order to avoid ambiguities in the angle prediction, the 3D module is trained to estimate the pair $(\sin\alpha, \cos\alpha)$. Given these values, as well as the estimation of the center coordinates, the final object orientation can be reconstructed by inverting Eq. 3.7.

The architecture of the 3D detection module is relatively simple and generic, consisting of two fully-connected layers, each having 2048 output neurons and ReLU activation function, followed by a third fully-connected layer returning 8 values per class, encoded as illustrated above. No normalization techniques such as Batch Normalization are used in this module.

3.4 Model Optimization

Before going into detail about the proposed training procedure, I briefly review the concept of Generalized Intersection-over-Union, first introduced in [20], as it is central in the optimization of the 3D detection module.

3.4.1 Generalized Intersection-over-Union

As already anticipated in Sec. 2.4.2, the most common and widespread metric utilized to evaluate object detectors is the Average Precision, a numerical value that summarizes the Precision-Recall curve of the system. To determine this curve, model predictions are associated to the ground truth depending on their Intersection-over-Union. When optimizing their models, however, most approaches opt for minimizing the distance between predicted and ground truth box parameters directly, which does not necessarily strongly correlate with increasing their IoU. As such, in [20] the authors propose to utilize the IoU directly as cost function for training the model: theoretically, this approach should lead to performance gains, given that it is an optimization objective that is closer to the evaluation metric.

Utilizing the IoU directly as cost function is possible in case all bounding boxes are axis-aligned, as their IoU can be computed in closed form only through the use of minimum, maximum, multiplication, summation and ReLU

operators, all of which are differentiable almost everywhere at worst. In case the predicted and target boxes do not overlap, however, their IoU would be equal to 0 and provide no gradient, rendering the optimization of this case impossible. As such, the authors propose the Generalized Intersection-over-Union (GIoU), an extention to the IoU which accounts for the disjoint case by introducing an extra term that measures the similarity in space between the two boxes. Formally, given two boxes P and Q having intersection area $\mathcal{I}$ and union area $\mathcal{U}$, their GIoU is defined as:

$$\text{GIoU} = \text{IoU} - E, \tag{3.8}$$

where

$$\text{IoU} = \frac{\mathcal{I}}{\mathcal{U}} \qquad E = \frac{\mathcal{A}^c - \mathcal{U}}{\mathcal{A}^c}. \tag{3.9}$$

$\mathcal{A}^c$ is used to indicate the area of the smallest axis-aligned bounding box containing both P and Q. Intuitively, the extra term E is equal to 0 if one of the boxes is completely contained in the other, and increases asymptotically to 1 as the two boxes grow apart in space. Moreover, like the IoU, it can be computed analytically using only differentiable operators, rendering the GIoU as a whole differentiable. As a consequence, the GIoU can be used directly as loss function:

$$\mathcal{L}_{\text{GIoU}} = 1 - \text{GIoU}. \tag{3.10}$$

This formulation is always within the range $[0, 2)$ and ensures that there is gradient even in case of disjoint boxes. Also, it is shown that, for boxes sharing low overlap, the GIoU has the potential of providing a steeper gradient compared to the traditional IoU, making it a more suitable cost function than the latter. By replacing the original loss functions with the IoU and GIoU-based ones, the authors improve the overall performance of both YOLOv3 and Faster R-CNN on two different large-scale datasets, validating the claim.

3.4.2 Training Procedure

The training of the baseline Faster R-CNN network mostly follows the original implementation, already introduced in Sec. 2.2.1.

Region Proposal Network To train the RPN, a balanced set of 256 anchors per image are selected for loss computation, with a ratio of 1:1 between positive and negative samples. More specifically, an anchor box is assigned to a ground truth box and labelled as positive either if its IoU with that box is above 0.7 or if it is the anchor having the highest IoU with that box. This second condition is necessary to ensure that no ground truth box is ignored during optimization. If the previous conditions are not met and an anchor has IoU below 0.3 with all ground truth boxes, it is labelled as negative. Otherwise, it is considered an ambiguous case and ignored during training. The loss function follows the original formulation (Eq. 2.2), with the exception that L_{reg} uses the GIoU instead of a smooth-L1 based direct minimization.

Detection Head Likewise, to train the detection head, 256 proposals per image are selected from the top 2000 scoring proposals after NMS, with a ratio of 1:3 between positive and negative samples. Again, the label assigned to each proposal is based off its overlap with ground truth boxes: if its IoU is below 0.5, it is marked as negative, otherwise it is considered positive and assigned to the ground truth box with which it shares the highest overlap. Given the selections, their features are extracted by the appropriate level of the pyramid (Eq. 3.1) and propagated to the detection head for loss computation. Like for the RPN, the loss computation follows the original work with the difference that the GIoU loss is used instead of the standard smooth-L1 objective for optimizing the box parameters.

3D Detection Module The 3D detection module is optimized together with the rest of the network. In this case, unlike at test time, instead of performing a second RoIAlign step on the final detections returned by the detection head,

the 3D module operates on the same set of positive proposals also used by the detection head.

Generally, the commonly adopted approach for training 3D box predictors consists of minimizing directly the adopted parametrization (in this case Eqs. 3.2, 3.3, 3.4, 3.7) against the ground truth using some sort of distance function (e.g. Eq. 2.4). Instead, I propose to extend the GIoU formulation to the 3D case. As already stated in Sec. 3.4.1, the IoU between two axis-aligned bounding boxes has closed form solution, and this is due to the fact that their intersection is still an axis-aligned box. Moreover, by approximating the minimum enclosing box as another axis-aligned box, the GIoU can also be calculated analytically and has a well-behaved gradient. 3D boxes, however, are subject to rotations and their intersection is a cuboid only if they have a relative orientation which is a multiple of $\pi/2$. In all other cases, their intersection would be a generic, irregular, polihedron. To ensure the existance of an analytical solution, I disentangle the estimation of the angle from the rest of the dimensions, which are optimized via the GIoU by considering their respective boxes at a canonical orientation. The loss function for this module is therefore comprised of two distinct components:

$$\mathcal{L}^{3D} = \mathcal{L}_{ang} + \mathcal{L}_{3IoU}, \tag{3.11}$$

where $\mathcal{L}_{ang}$ is responsible for optimizing the orientation and $\mathcal{L}_{3IoU}$ is tasked to optimize position and dimentions through the GIoU.

Formally, let $\mathbf{B} = (x, y, z, h, w, l, \theta)$ be the 3D box predicted by the module and $\hat{\mathbf{B}} = \left(\hat{x}, \hat{y}, \hat{z}, \hat{h}, \hat{w}, \hat{l}, \hat{\theta}\right)$ its assigned ground truth box. Let $\hat{\alpha} = \hat{\theta} - \text{atan2}\left(-\hat{x}, \hat{z}\right)$ be the ground truth observation angle. $\mathcal{L}_{ang}$ is defined as the smooth-L1 loss between the estimated and the target observation angles:

$$\mathcal{L}_{ang} = \text{smooth}_{L1}\left(\sin\hat{\alpha} - \sin\alpha\right) + \text{smooth}_{L1}\left(\cos\hat{\alpha} - \cos\alpha\right). \tag{3.12}$$

In order to optimize the position and dimensions of the boxes through the GIoU, I first prerotate them such that their orientation angle is equal to 0, yielding:

$$\mathbf{B}_0 = (x, y, z, h, w, l, 0), \tag{3.13}$$

$$\hat{\mathbf{B}}_0 = (\hat{x}, \hat{y}, \hat{z}, \hat{h}, \hat{w}, \hat{l}, 0). \tag{3.14}$$

Under this assumption, the boxes can be directly defined in terms of their opposing corners:

$$x_{1,2} = x \pm l/2 \qquad y_{1,2} = y \pm h/2 \qquad z_{1,2} = z \pm w/2, \tag{3.15}$$

$$\hat{x}_{1,2} = \hat{x} \pm \hat{l}/2 \qquad \hat{y}_{1,2} = \hat{y} \pm \hat{h}/2 \qquad \hat{z}_{1,2} = \hat{z} \pm \hat{w}/2. \tag{3.16}$$

Given these values, and by approximating the minimum enclosing box as another cuboid having $\theta = 0$, computing the intersection area $\mathcal{I}$, the union area $\mathcal{U}$ and the minimum enclosing area $\mathcal{A}^c$ is a trivial extension of the 2D case (see Alg. **??** for the complete formulation). Finally, given the GIoU value, the loss function is obtained using Eq. 3.10.

More specifically, instead of minimizing the GIoU loss function between $\mathbf{B}_0$ and $\hat{\mathbf{B}}_0$ directly, inspired by the disentangling transformation introduced in [102] I split the optimization into six separate contributions, each responsible for a single degree of freedom:

$$\mathcal{L}_{3IoU} = \frac{1}{6} \sum_{i \in \{x,y,z,h,w,l\}} \left(1 - GIoU\left(\hat{\mathbf{B}}_0, \mathbf{B}_0^i\right)\right). \tag{3.17}$$

Here, $\mathbf{B}_0^i$ is used to represent the box obtained from $\mathbf{B}_0$ by replacing all values except for i with the ground truth (e.g. $\mathbf{B}_0^z = (\hat{x}, \hat{y}, z, \hat{h}, \hat{w}, \hat{l}, 0)$)). This formulation leads to a considerable optimization speedup, especially early in the training where most predictions are disjoint from their corresponding ground truth boxes. Further analysis will be presented in Sec. 3.5.3.

Training Details The model is trained end-to-end on full resolution images for 90k iterations, using Stochastic Gradient Descent with batch size 4, weight

decay 5e-4 and momentum 0.9. The learning rate is initially set to 10^{-2} and is reduced by a factor of 10 every 30k iterations. The ResNet backbone is initialized with ImageNet pretraining values and its Batch Normalization layers as well as its first two convolutional blocks are kept fixed during training. To enrich the training data, each sample is independently augmented by random horizontal flipping with probability 0.5 as well as by jittering its saturation, brightness and contrast by ±30%.

3.5 Experimental Results

In this section I introduce KITTI [22], the autonomous driving dataset used to train and evaluate the proposed approach. Then, I perform a quantitative comparison against current state of the art monocular 3D object detectors. Finally, I analyze the loss function used for optimizing the 3D module and compare its effectiveness against other alternatives.

3.5.1 The KITTI Dataset

The proposed system is trained and evaluated on the KITTI [22] dataset, which currently constitutes the *de facto* choice of the autonomous driving community for research.

This dataset provides both image, LiDAR and odometry data, as well as ground truth annotations for a wide variety of tasks including semantic segmentation, instance segmentation, visual odometry/SLAM, 2D object detection and tracking, 3D object detection, depth estimation and optical flow. Image data is acquired using two stereo camera setups, one for greyscale and one for color, both displaced at the front of the vehicle and providing images at a resolution of 1382×512. Due to rectification, the images provided for training are smaller and have an approximate resolution of 1240×375. LiDAR scans are recorded using a 64-planes Velodyne spinning at 10 frames per second and capturing approximately 100k points per revolution. The cameras are

synchronized with the Velodyne and capture images at the beginning of each revolution, also at 10Hz.

The 2D/3D Object Detection dataset is gathered by annotating dissimilar frames from several recorded sequences with the corresponding observable 2D and 3D bounding boxes, for a total of 7481 training samples and 7518 test samples. The test set annotations are not publicly available and are used exclusively by the online evaluation server for performance evaluation. More specifically, KITTI provides box annotations for 7 different classes, that is Car, Pedestrian, Cyclist, Van, Truck, Sitting Person and Tram, but only the first three are considered for evaluation by the official benchmark, as the others are too scarce in number for proper model training. Still, like many current methods I only consider the Car class for prediction, as it is considerably more frequent and evenly distributed within the dataset compared to Pedestrians and Cyclists. Also, each annotated box is attributed one of three categories, *easy*, *moderate* or *hard*, depending on its size on the image plane and on how much it is occluded and truncated.

Following previous work [67, 17, 76], I split the available 7481 annotated images into a training and a validation set, comprised of 3712 ad 3769 samples respectively. It is important to note that, in order to ensure proper performance evaluation, these two splits are originated from two disjoint sets of sequences, such that no similar scenes are shared between training and validation.

3.5.2 Comparison with the State of the Art

I evaluate the 3D localization and detection performance of the system using the KITTI Average Precision metric for bird's eye view (AP_{BEV}) and 3D detection (AP_{3D}). For a exhaustive comparison, I consider both the official 0.7 IoU threshold and the more permissive 0.5 IoU threshold. The results for the two tasks are shown in Tab. 3.1 and Tab. 3.2 respectively.

The proposed method exhibits state-of-the-art performance on the validation set, surpassing all other monocular approaches by a good margin on the official 0.7 IoU threshold, while also being competitive with the stereo-based

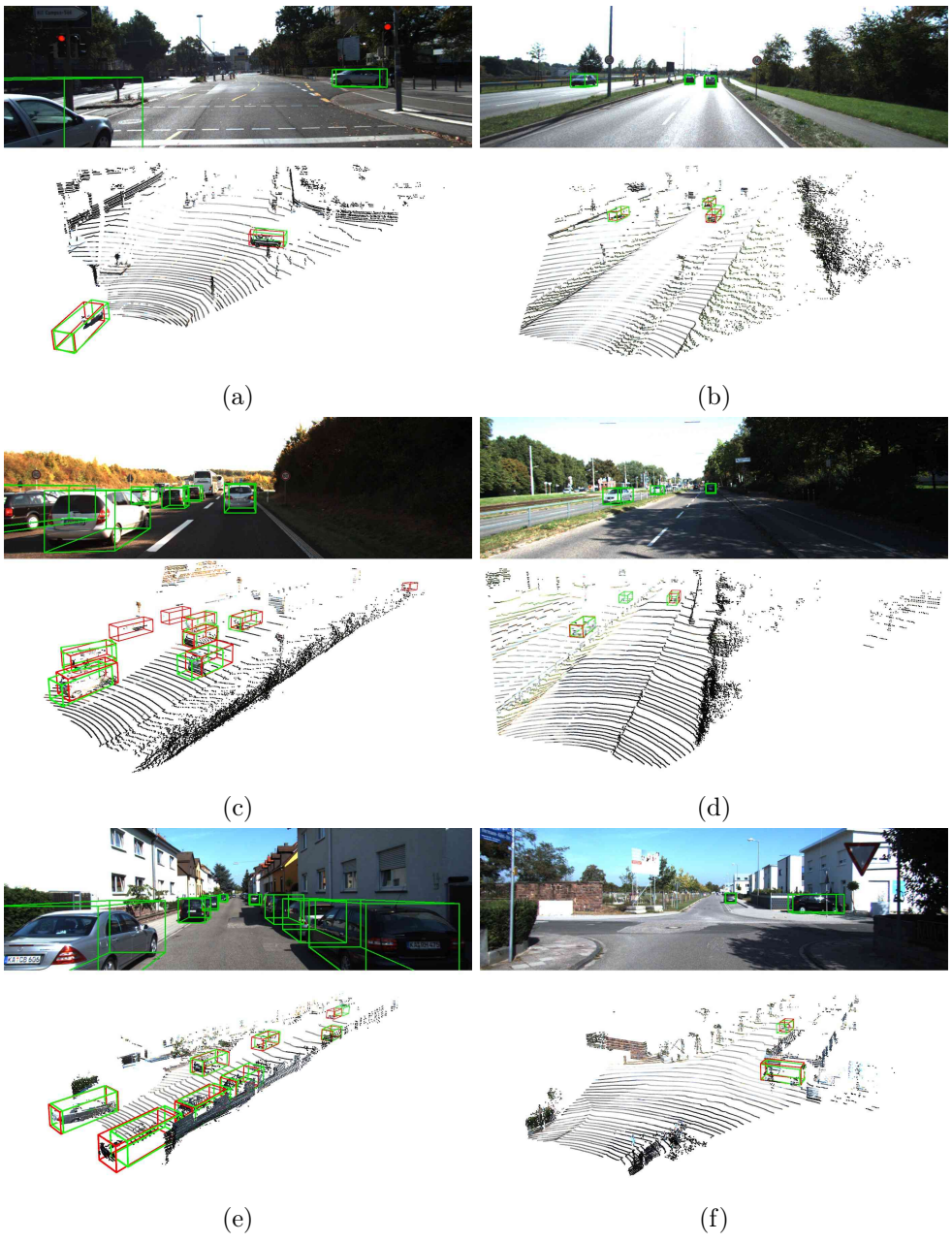


Figure 3.4: Results of the proposed method on the validation set. Red bounding boxes correspond to the ground truth, while green boxes represent the detections. The LiDAR point clouds are used exclusively for visualization. Best viewed in color.

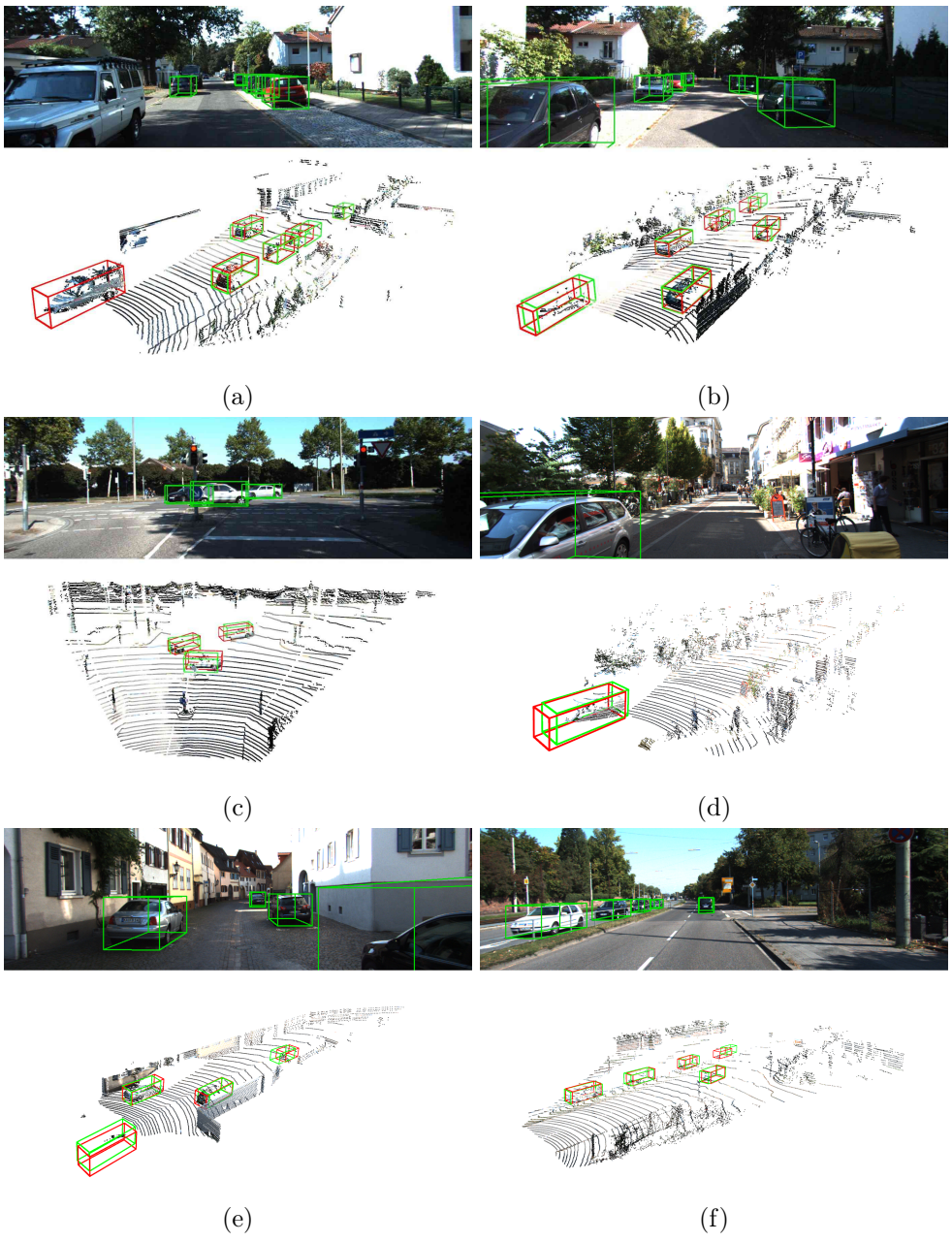


Figure 3.5: (**cont.**) Results of our method on the validation set. The red bounding boxes correspond to the ground truth, while the green boxes are our detections. The LiDAR point clouds are used exclusively for visualization. Best viewed in color.

Method	Data	AP_{BEV} @ 0.5 IoU			AP_{BEV} @ 0.7 IoU		
		Easy	Moderate	Hard	Easy	Moderate	Hard
TLNet (S) [76]	Stereo	62.46	45.99	41.92	29.22	21.88	18.83
Mono3D [15]	Mono	30.50	22.39	19.16	5.22	5.19	4.13
OFNet [78]	Mono	-	-	-	11.06	8.79	8.91
MultiFusion [80]	Mono	55.02	36.73	31.27	22.03	13.63	11.60
TLNet (M) [76]	Mono	52.72	37.22	32.16	21.91	15.72	14.32
MonoGRNet [79]	Mono	54.21	39.69	33.06	24.97	19.44	16.30
MonoPSR [16]	Mono	56.97	43.39	36.00	20.63	18.67	14.45
MonoDIS [102]	Mono	-	-	-	24.26	18.43	16.95
Ours	Mono	**60.17**	**43.45**	**36.53**	**29.70**	**21.86**	**18.13**

Table 3.1: Results for **localization** on the KITTI validation set for the *Car* class.

Method	Data	AP_{3D} @ 0.5 IoU			AP_{3D} @ 0.7 IoU		
		Easy	Moderate	Hard	Easy	Moderate	Hard
TLNet (S) [76]	Stereo	59.51	43.71	37.99	18.15	14.26	13.72
Mono3D [15]	Mono	25.19	18.20	15.22	2.53	2.31	2.31
OFNet [78]	Mono	-	-	-	4.07	3.27	3.29
MultiFusion [80]	Mono	47.88	29.48	26.44	10.53	5.69	5.39
TLNet (M) [76]	Mono	48.34	33.98	28.67	13.77	9.72	9.29
MonoGRNet [79]	Mono	50.51	36.97	30.82	13.88	10.19	7.62
MonoPSR [16]	Mono	49.65	**41.71**	29.95	12.75	11.48	8.59
MonoDIS [102]	Mono	-	-	-	18.05	14.98	13.42
Ours	Mono	**55.70**	40.34	**34.40**	**22.48**	**16.67**	**15.08**

Table 3.2: Results for **detection** on the KITTI validation set for the *Car* class.

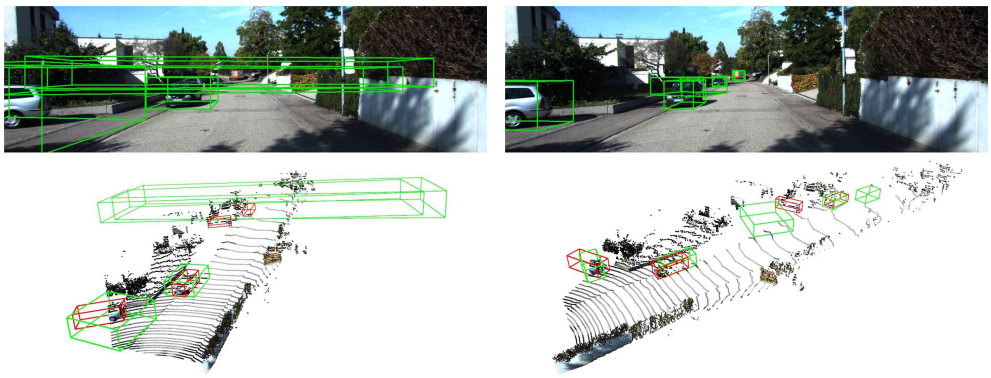
approach TLNet [76]. Likewise, at the 0.5 IoU threshold, the presented approach outperforms all other methods on both tasks except for the *moderate* samples on the detection task, where the results are slightly lower than those of MonoPSR [16]. The relatively larger improvement in performance for the 0.7 IoU case compared to the 0.5 IoU one suggests that the proposed approach returns, on average, detections having higher localization accuracy.

On a NVIDIA Tesla V100 GPU, the inference time for the 3D detection module is approximately 5 ms, with slight variations depending on the number of 2D detections that must be processed by the 3D module. The total inference time of the entire pipeline is about 50ms on full resolution KITTI images.

3.5.3 Comparison with other Loss Formulations

To validate the choice of utilizing an approximation of the GIoU loss formulation, I conducted several experiments in which different loss formulations are used.

Is learning each dimension disjointly important? As first experiment, I investigated whether utilizing a separate loss component for each degree of freedom of position and dimensions is beneficial to the performance of the resulting system. In particular, I carried out an experiment where all six parameters are optimized together in a single loss function. What I noticed was a tendency of the system to reduce the GIoU loss value in case of non-overlapping boxes by increasing their size rather than by trying to match their positions in space. This behavior led to a significant number of spurious detections early in the training, which in turn caused a plateau of the loss function around the value of 1, ultimately slowed down the learning process and leading to worse accuracy. A visualization of this phenomenon on validation images is visible in Fig. 3.6: as it can be seen, anomalous detections resulting from this behavior are very prominent at the early stages of optimization, and still persist, although less extremely, once the system is fully trained.



(a) (b)

Figure 3.6: Anomalous behavior of the 3D detection module when trained with the joint GIoU. Fig. 3.6a shows a sample result early in the training, while 3.6b shows the same sample at the end of optimization results for the trained system.

I speculate that this phenomenon is induced by how the Generalized Intersection over-Union is formulated: two very similar boxes, close together but disjoint, always leads to a higher loss value than two *very* different boxes such that one is completely inside the other. The reason for this behavior is illustrated in Fig. 3.7. This also holds true for very different boxes that share a low enough overlap such that $E >$ IoU (see Eq. 3.8). As a result, the tendency of the system, especially during the first half of the training procedure where most predicted 3D boxes tend not to overlap their associated ground truth, is to reduce the loss value by increasing the size, such that E is driven to zero. This behavior was probably not noticed when the GIoU was applied to 2D detection, as most 2D detectors optimize the refinement branch of the detection head only on anchors or proposals that share a significant overlap with the ground truth boxes to begin with. As a result, since the final predictions are obtained as a correction to these prior boxes, it is unlikely that a large portion of them are disjoint from their corresponding ground truth. Conversely, in the

proposed approach, as no 3D prior boxes are used, the estimations could be anywhere in the space, especially during the early stages of optimization, which leads to a significant number of non-overlapping samples. Allowing only one dimension to be optimized at a time avoids this problem altogether: when a center coordinate is optimized the boxes might be disjoint but the dimensions cannot be varied. When a dimension is optimized, the boxes must overlap as they share the same center.

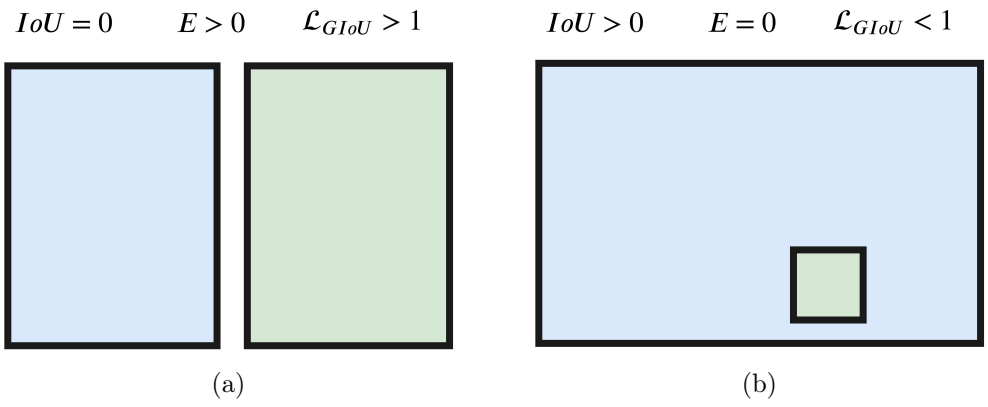


Figure 3.7: Example of the anomalous behavior of the Generalized Intersection-over-Union. Two identical non-overlapping boxes (a) correspond to a loss value that is always greater than two very different boxes where one contains the other (b).

Comparison with other optimization strategies To further test the effectiveness of the proposed optimization strategy, following the same training routine introduced in Sec. 3.4.2, I trained the 3D module considering alternative loss functions. In particular, I compared against two different formulations:

- the **direct regression**, which is the approach commonly adopted in the literature, where the parametrization introduced in Sec. 3.3 is optimized directly against the ground truth using the smooth-L1 distance as cost function;

- the **corner loss**, first introduced in [23] as a regularization term to improve box localization accuracy. This loss allows for the joint optimization of all degrees of freedom of the box via the minimization of the distance of its eight corners with respect to the corners of its corresponding ground truth box. Formally, given the predicted box $\mathbf{B}$ and its associated ground truth $\hat{\mathbf{B}}$, I compute their corresponding sets of ordered corners $\hat{\mathbf{p}} \in \mathbb{R}^{8\times 3}$ and $\mathbf{p} \in \mathbb{R}^{8\times 3}$ to which I apply the smooth-L1 cost:

$$\mathcal{L}_{\text{crn}} = \text{smooth}_{L1}\left(\hat{\mathbf{p}} - \mathbf{p}\right). \tag{3.18}$$

  For this formulation, I also tested the disjoint version, where each of the seven degrees of freedom is optimized by a separate subterm, similarly to what was done for the GIoU loss. Note that in this case, unlike the GIoU, the angle can also be optimized using the corner loss, as a differentiable close form solution always exists.

The quantitative results of these studies are displayed in Tab. 3.3. The proposed formulation convincingly outperforms all other options, and the corner loss also performs surprisingly well. As expected, optimizing each degree of freedom separately leads to considerable improvements for the GIoU-based loss, as it side-steps the anomalous behavior illustrated in the previous paragraph. The disjoint formulation also improves, albeit by much smaller margins, the overall performance of the corner loss, as it simplifies the optimization processes. On the other hand, the direct regression falls considerably behind in terms of performance. This is indicative of the fact that optimizing the model by reasoning in terms of boxes as a whole, either by maximizing an approximation of their GIoU or by minimizing the distance between their corners is more effective than optimizing the parametrization directly. I speculate that the GIoU performs better than the corner loss as it is a closer surrogate to the evaluation metric, which is an IoU-based Average Precision. Moreover, the GIoU loss is overall more stable and easier to integrate into the baseline 2D detection model, requiring no hyperparameter tuning for successful training. Conversely, both the direct and corner losses require careful balancing in order

Loss	AP_{3D} @ 0.7/0.5 IoU		
	Easy	Moderate	Hard
Direct	13.26 / 34.40	11.21 / 27.12	10.99 / 23.18
Crn	17.96 / 46.59	14.20 / 32.97	12.14 / 30.79
Crn (D)	18.93 / 48.66	14.35 / 36.76	12.09 / 31.23
GIoU	18.19 / 48.59	14.87 / 36.17	13.84 / 31.71
GIoU (D)	**22.48 / 55.70**	**16.67 / 40.34**	**15.08 / 34.40**

Table 3.3: Results of the ablation study on the loss function for 3D detection. **Direct** represents the direct loss, **Crn** the corner loss and **GIoU** the proposed formulation of the Generalized Intersection-over-Union loss. **(D)** is used to indicate the disjoint optimization of each parameter.

to avoid spikes early in the training due to outlying detections which would lead to optimization instability.

3.5.4 Qualitative results

Some qualitative results of the proposed system are shown in Fig. 3.4. In particular, I display the detection both on the input image and in 3D using the LiDAR point cloud as reference. The system is capable of detecting most 3D objects with high accuracy, especially at short to medium distances. It is also capable of locating instances that are not annotated, such as in Fig. 3.4d, for which the correctness of the prediction can be inferred by the LiDAR point cloud. The most common causes for inaccurate localization are occlusions (3.4c), which might cause the underlying 2D detection model to fail, leading to a missed detection, truncation (3.5a) and high distances (3.4e, 3.5b, 3.5e, 3.5f). Uncommon orientations also prove to be challenging for the model (3.4a, 3.5c), likely due to the limited amount of training data available for these cases.

In Fig. 3.8 I also show some detections of the trained system for harder classes such as pedestrians and cyclists. The system exhibits some promising

results on these categories, as it is able to detect them fairly confidently and determine their position with acceptable accuracy. The lack of training data for these cases, coupled with the fact that pedestrians and cyclists are inherently more complex than vehicles due to not being rigid objects, leads to 3D predictions that are overall noisy, and hinders an adequate evaluation of the system.

Figure 3.8: Qualitative results for pedestrians and cyclists.

Algorithm 1: Computation of the Generalized Intersection-over-Union for equioriented 3D cuboids.

input: Predicted $\mathbf{B}_0$ and ground truth $\hat{\mathbf{B}}_0$ equioriented boxes:

$$\mathbf{B}_0 = (x, y, z, h, w, l, 0),$$
$$\hat{\mathbf{B}}_0 = (\hat{x}, \hat{y}, \hat{z}, \hat{h}, \hat{w}, \hat{l}, 0)$$

output: $GIoU$

Compute the opposing corners for each box:

$$x_{1,2} = x \pm l/2 \qquad \hat{x}_{1,2} = \hat{x} \pm \hat{l}/2$$
$$y_{1,2} = y \pm h/2 \qquad \hat{y}_{1,2} = \hat{y} \pm \hat{h}/2$$
$$z_{1,2} = z \pm w/2 \qquad \hat{z}_{1,2} = \hat{z} \pm \hat{w}/2.$$

Compute the predicted box area of $\mathbf{B}_0$:

$$\mathcal{A} = h \cdot w \cdot l$$

Compute the ground truth box area of $\hat{\mathbf{B}}_0$:

$$\hat{\mathcal{A}} = \hat{h} \cdot \hat{w} \cdot \hat{l}$$

Compute the intersection area $\mathcal{I}$:

$$x_1^{\mathcal{I}} = \min(x_1, \hat{x}_1) \qquad x_2^{\mathcal{I}} = \max(x_2, \hat{x}_2)$$
$$y_1^{\mathcal{I}} = \min(y_1, \hat{y}_1) \qquad y_2^{\mathcal{I}} = \max(y_2, \hat{y}_2)$$
$$z_1^{\mathcal{I}} = \min(z_1, \hat{z}_1) \qquad z_2^{\mathcal{I}} = \max(z_2, \hat{z}_2)$$
$$l^{\mathcal{I}} = x_1^{\mathcal{I}} - x_2^{\mathcal{I}} \quad h^{\mathcal{I}} = y_1^{\mathcal{I}} - y_2^{\mathcal{I}} \quad w^{\mathcal{I}} = z_1^{\mathcal{I}} - z_2^{\mathcal{I}}$$
$$\mathcal{I} = \begin{cases} l^{\mathcal{I}} \cdot h^{\mathcal{I}} \cdot w^{\mathcal{I}} & \text{if } l^{\mathcal{I}} > 0, h^{\mathcal{I}} > 0, w^{\mathcal{I}} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Compute the union area $\mathcal{U}$: $\mathcal{U} = A + \hat{A} - \mathcal{I}$

Compute minimum enclosing box area A^c:

$$x_1^c = \max(x_1, \hat{x}_1) \qquad x_2^c = \min(x_2, \hat{x}_2)$$
$$y_1^c = \max(y_1, \hat{y}_1) \qquad y_2^c = \min(y_2, \hat{y}_2)$$
$$z_1^c = \max(z_1, \hat{z}_1) \qquad z_2^c = \min(z_2, \hat{z}_2)$$
$$\mathcal{A}^c = (x_1^c - x_2^c) \cdot (y_1^c - y_2^c) \cdot (z_1^c - z_2^c)$$

Compute the GIoU:

$$GIoU = \frac{\mathcal{I}}{\mathcal{U}} - \frac{A^c - \mathcal{U}}{A^c}$$

3.6 A case study: 3D GIoU applied to Frustum-PointNets

To further validate the effectiveness of the proposed loss function to the task of 3D Object Detection, I conduct a study in which I apply it to the optimization of an entirely different 3D detector: Frustum-PointNets [23]. Frustum-PointNets is a fusion-based 3D detection pipeline that uses both the RGB image and the LiDAR point cloud as inputs. First, the image is processed by Faster R-CNN to extract the set of 2D objects. The 2D detections are then cast to viewing frustums, which are used to determine the portion of point cloud relative to each object. Each of these point clouds is finally processed by the 3D detection network, responsible for locating the object within the cloud and estimating its 3D box. More specifically, the 3D detector is comprised of three modules, applied in sequence:

- a PointNet for segmentation, responsible for removing the background points;
- a T-Net (i.e. a small PointNet) that processes the remaining set of foreground points, translated such that their centroid is in the origin, and estimates the local object center coordinates;
- a second PointNet, which accepts the foreground point cloud traslated in the center estimated by the T-Net as its input and returns the 3D box estimation.

Due to requiring existing 2D detections as data for its training, this pipeline is not trained jointly. Generally, the model for 2D detection and the model for 3D detection are trained separately from one another, each using its own optimization routine. In this study, in particular, I modify only the training loss for the 3D detection module, without affecting in any way the underlying 2D detector. Moreover, for fair comparison, I adopt the exact same 2D detection model, with the same optimized set of weights released by the authors.

3.6.1 3D detector: standard optimization

All three modules involved in the estimation of the 3D box from the sampled point cloud are optimized jointly, using a multi-task loss function:

$$\mathcal{L} = \mathcal{L}_{\text{seg}} + \lambda \left(\mathcal{L}_{\text{c1-reg}} + \mathcal{L}_{\text{c2-reg}} + \mathcal{L}_{\text{h-cls}} + \mathcal{L}_{\text{h-reg}} + \mathcal{L}_{\text{s-cls}} + \mathcal{L}_{\text{s-reg}} + \gamma \mathcal{L}_{\text{corner}} \right). \tag{3.19}$$

The **segmentation** submodule returns a scalar value for each input point, encoding the probability that said point belongs to the object of interest. As such, the loss $\mathcal{L}_{\text{seg}}$ used to train this submodule takes the form of a standard binary cross-entropy loss (see Eq. 2.8), averaged over all input points.

The T-Net for **center estimation** is trained to directly estimate the position of the object center given the set of foreground points shifted around their centroid. It is trained via the loss $\mathcal{L}_{\text{c1-reg}}$, which takes the form of a smooth-L1 distance (Eq. 2.4) between the estimated and the ground truth center.

The second PointNet for **3D box estimation** is tasked to return the 3D box given the set of foreground points shifted at the estimated center coordinates. More specifically, this submodule returns a total of $3 + 4 \times \text{NS} + 2 \times \text{NH}$ outputs, where:

- the first 3 outputs represents the estimated position of the object center with respect to the input foreground point cloud. These are trained via the loss $\mathcal{L}_{\text{c2-reg}}$ which is a smooth-L1 distance between the predicted and the ground truth center;
- the subsequent $4 \times \text{NS}$ outputs encode the dimensions estimation, which is performed relative to a set of NS predetermined dimension templates. More specifically, the first NS outputs represent a discrete probability distribution that determines which template is used, and the following $3 \times \text{NS}$ values encode height, width and length corrections with respect to each template. The classification is trained via $\mathcal{L}_{\text{s-cls}}$, which is a standard multi-class cross-entropy loss (see Sec. 2.2.1). The correction is trained

via $\mathcal{L}_{\text{s-reg}}$, which is a smooth-L1 distance between the estimated correction and the ground truth one: note that in this case the loss function is only computed for the right template, determined by ground truth;

- the last $2 \times \text{NH}$ outputs encode the orientation estimation which, like for the dimensions, is split into a classification component and a correction component. In particular, the 360° are divided into NH bins: the first NH outputs represent a discrete probability distribution that the orientation angle lies in that bin and the remaining NH values encode corrections with respect to each bin center. Training follows the one used for the dimensions: $\mathcal{L}_{\text{h-cls}}$ is a multi-class cross-entropy loss for bin classification and $\mathcal{L}_{\text{h-reg}}$ is a smooth-L1 distance computed on the corrected bin, which is determined by ground truth.

Additionally, to regularize the estimation of the 3D box and improve the accuracy, an additional loss term, the corner loss $\mathcal{L}_{\text{corner}}$ is used, which consists of minimizing the smooth-L1 distance between the estimated and the ground truth box corners. In order to compute this loss term, the 3D box originated using the estimated corrections on the ground truth bins is considered.

3.6.2 3D detector: proposed optimization

I propose to modify the multi-task loss (Eq. 3.19) such that the estimation of the 3D box center and dimensions is optimized through the proposed disjoint 3D GIoU loss. More specifically, I remove the losses previously tasked to estimate center ($\mathcal{L}_{\text{c2-reg}}$) and dimensions ($\mathcal{L}_{\text{s-reg}}$) as well as the corner loss ($\mathcal{L}_{\text{corner}}$), and I subtitute them with the formulation introduced in Eq. 3.17:

$$\mathcal{L} = \mathcal{L}_{\text{seg}} + \lambda \left(\mathcal{L}_{\text{c1-reg}} + \mathcal{L}_{\text{h-cls}} + \mathcal{L}_{\text{h-reg}} + \mathcal{L}_{\text{s-cls}} + \gamma \mathcal{L}_{\text{3IoU}} \right) \tag{3.20}$$

As originally done for the corner loss computation, in order to compute the GIoU loss value I consider the box obtained from the predicted correction on the ground truth dimension template. No other modification is done to

Car	AP_{BEV} / AP_{3D} @ 0.7 IoU		
	Easy	Moderate	Hard
Original v1	87.82 / 83.26	82.44 / 69.28	74.77 / 62.56
Baseline v1	**87.64** / 82.97	83.09 / 71.11	75.75 / 63.43
GIoU v1	87.59 / **85.36**	**83.89** / **73.47**	**76.16** / **65.29**
Original v2	88.16 / 83.76	84.02 / 70.92	76.44 / 63.65
Baseline v2	88.19 / 83.33	85.38 / 71.74	77.15 / 64.01
GIoU v2	**88.41** / **86.08**	**85.99** / **74.47**	**77.64** / **66.14**

Table 3.4: Results of the study on applying the GIoU as cost function for optimizing Frustum-PointNets. The Car class is considered for evaluation. **Original** and **Original v2** indicate the results reported by the original paper [23], using the PointNet and PointNet++ based architectures respectively. **Baseline** and **Baseline v2** represent the results obtained by my reimplementation of the system. **GIoU** indicates that the reimplemented system makes use of the GIoU loss formulation in Eq. 3.17 to optimize the estimation of center and dimensions.

the system: the 3D box orientation and dimensions are still estimated using a hybrid classification-correction approach, and the model architecture is exactly the same. Also, I adopt the exact same training routine as the original work.

3.6.3 Experimental Results

The results of the experiments for the Car class are displayed in Table 3.4. More specifically, I report both the original results shown in the paper [23], as well as those obtained by my reimplementation of the system in PyTorch. I experimented woth both versions of Frustum-PointNets:

- **version 1** adopts simple PointNet networks for all three modules;
- **version 2** utilizes PointNet++ models both for the segmentation module

and the box estimation module, resulting in a more powerful and context-aware, albeit quite slower, network.

In order to avoid ambiguities and isolate the analysis to the different training strategy used to optimize the 3D detection network, at evaluation time I adopt the exact same set of 2D detections used to evaluate the original system, which are made publicly available by the authors: this way, no performance change can be attributed to a difference in behavior of the underlying 2D object detector.

The models trained with the GIoU loss formulation strongly outperform their corresponding original versions, especially for the 3D task which is the one that the loss explicitly aims to optimize. Particularly notable is the fact that the v1 model, when trained using the proposed loss, distinctly outperforms the v2 version optimized with the original loss function, despite being a considerably simpler model. This is indicative of the fact that an appropriate optimization strategy might be more important for the final result than the model architecture.

To further ablate the proposed formulation, in Table 3.5 I also show the results of the trained models for the Cyclist class. Even in this case, the v1 system trained with the GIoU loss vastly outperform its v2 counterpart with the standard training procedure. Most surprising, however, is the fact that the GIoU-optimized v2 model, while still outperforming the original model, performs worse than the GIoU-optimized v1 on the 3D metric. This behavior might be the result of overfitting: as already stated in Sec. 3.5.1, available Cyclist data is very limited compared to Car data. As a result, the risk of overfitting such data is higher, especially when adopting more complex models like the version 2 of Frustum-PointNets. Conversely, the much simpler architecture of version 1 leads to simpler features being learned, which in turn improves generalization on unseen samples. On the BEV localization task, on the other hand, version 2 firmly outperforms version 1: this is possibly due to the fact that the task is simpler, and therefore the noisier predictions induced by overfitting do not affect performance as much.

Cyclist	AP_{BEV} / AP_{3D} @ 0.7 IoU		
	Easy	Moderate	Hard
Original v2	81.82 / 77.15	60.03 / 56.49	56.32 / 53.37
GIoU v1	83.44 / **81.04**	62.09 / **59.88**	58.60 / **55.76**
GIoU v2	**84.77** / 78.53	**63.75** / 57.42	**59.44** / 53.96

Table 3.5: Results of the proposed system on the Cyclist class, compared against the original results.

3.7 Discussion

In this chapter, I proposed an extention to the 2D object detector Faster R-CNN consisting of a simple module responsible for monocular 3D detection which is trained using a disjoint formulation of the Generalized Intersection-over-Union (GIoU) loss function. To ensure the existence of an analytical solution, I disentangled the estimation of the orientation from that of position and dimensions, rotating the boxes to a canonical angle before computing their GIoU. Moreover, to avoid an anomalous behavior of the GIoU early during training, I opted for optimizing each degree of freedom separately by adopting a dedicated loss function for each.

The resulting system exhibited remarkable performance, surpassing more complex and model-driven pipelines on the autonomous driving KITTI dataset. The approach is also simple, as the 3D detection module only consists in a handful of fully-connected layers, and thus could be incorporated straightforwardly into other existing 2D detection methods.

Most of the performance gain achieved by the proposed system is to be attributed to the way that the 3D detection module is optimized, as shown by the study conducted by utilizing more traditional optimization functions. To further validate this claim, I adopted the GIoU loss function for optimizing the LiDAR-based 3D detector Frustum-PointNets, and showed that the proposed formulation can lead to significant improvements even for entirely different

neural architectures.

Chapter 4

3D Object Detection on LiDAR scans via Voting and Self-Attention Mechanisms

4.1 Prior Art and Motivation

Camera-based perception solutions, whilst being currently considerably cheaper than LiDAR-based ones, suffer from inferior performance, mostly due to the fact that images do not encode depth information directly. Stereo setups can be used to estimate depth through disparity calculation; such depth, however, tends to be considerably noisier, especially at high distances, and tends to fail in presence of specific patterns or lack of textures. Similarly, networks for 3D detection trained on monocular images require an enormous amount of data in order to generalize properly to unobserved scenes and, even then, they still strongly underperform stereo and LiDAR-based pipelines.

LiDAR sensors, on the other hand, provide extremely accurate, albeit sparser, depth information in the form of a point cloud of the surrounding environment. While currently being considerably more expensive than cameras, advancements in sensor technology are consistently reducing the costs of

LiDAR solutions, and prices are quickly approaching those of stereo-camera setups. This makes LiDAR based perception techniques worth investigating, as consumer-friendly LiDAR options are likely to emerge in the near future.

As already mentioned in Sec. 1.6.1, state-of-the-art neural networks operating on point cloud data are generally divided into two macro categories: voxel-based methods and point-based methods. Voxel-based methods [70, 71, 13, 23] first transform the point cloud into a regular grid of voxels, and then process the transformed data using three-dimensional convolutional operators. Point-based methods [14, 72, 26, 73], on the other hand, operate directly on the raw point cloud by leveraging PointNet-like [24, 25] models. Voxel-based methods tend to be faster than point-based approaches, mostly due to the fact that a regular grid of voxels is easier to process compared to points and there exist libraries [68, 69] that allow for efficient convolution computation by ignoring empty voxels. Conversely, raw point clouds are more challenging to process directly, as they are sets with no intrinsic internal structure, which makes point-based methods usually less efficient. However, processing points directly avoids the quantization introduced by the voxelization process, allowing for more accurate localization.

In this chapter I investigate self-attantion [27] as a way of strengthening intermediate feature representations of point-based methods, which all rely on PointNet-like structures to extract features. More specifically, I build upon the detection pipeline Votenet, introduced in [26] for performing 3D object detection on dense point clouds of controlled scenarios, such as room scenes [103, 104]. As this method does not adapt well to noisier scenarios, such as autonomous driving scenes obtained from LiDAR scans, I introduce some simple modifications in order to boost performance. Then, I introduce self-attention as an integral part of the *Set Abstraction* (SA) layers, with the aim of making points aware of each other when computing features, which should lead to stronger representations and, ultimately, better detection performance.

4.2 Votenet for 3D Object Detection on Driving Scenarios

Before delving into the details of the model, I briefly review PointNet models, as they represent a central component in point processing pipelines.

4.2.1 PointNets for point cloud processing

Differently from images, point clouds are sets, that is they are not characterized by any explicit internal structure. As sets, one of the properties they have is that they are invariant to permutations of their elements, meaning that shuffling the order of the points does not affect the cloud itself. Due to this property, particular care must be taken when processing this kind of data using neural networks, as the resulting model must approximate a function that is symmetric by construction, meaning that its output must be unaffected by the order of the input elements.

Currently, the dominant approach in literature for dealing with set inputs is represented by PointNets [24]. The idea behind this class of models is very simple: in order for the model to approximate a symmetric function, first each element of the set is processed individually in order to extract features, and then a global descriptor of the set is obtained by aggregating the information about each individual element via a symmetric function:

$$f\left(\{x_1, \ldots, x_n\}\right) = g\left(h\left(x_1\right), \ldots, h\left(x_n\right)\right). \tag{4.1}$$

Here, f represents the model, h can be any function and g is a symmetric function. Commonly, in PointNets the function h is approximated by a stack of fully-connected layers, also called in literature as a Multi-Layer Perceptron (MLP), while g by max pooling, as the maximum operation is symmetric. Therefore, the function f as a whole is symmetric, and returns as output a single element, often called *signature*, that encodes a global representation of the input set. Such signature can then be used for further processing: for

classification, for instance, it can be passed as input to an additional classifier model. Analogously, for point segmentation, the signature can be fused with the features of each individual point (i.e. $h(x_i)$) in order to enrich them with global information before classifying each one.

While being simple and efficient, this formulation suffers from an important limitation: by computing a global signature directly via a single symmetric function g, it is unable to reason hierarchically and extract local structures and patterns within the input. Therefore, while it works well for simple tasks such as classification or object part segmentation, it falls short for more complex scenarios, where multiple objects are involved, or harder tasks such as detection. The follow-up work PointNet++ [25] aims to resolve this weakness by extending PointNet such that hierarchical representations are extracted by looking at progressively increasing regions within the input. The core component that allows to generate such hierarchies is the *Set Abstraction* (SA) layer, which operates on a set of N input points and performs the following operations:

- first, a subset of S centroids is sampled from the input. To ensure an even distribution of the samples, sampling is performed using the Farthest Point Sampling (FPS) algorithm, which returns the S elements withing the input that are farthest apart from each other according to some metric. When working with points, the most commonly used metric is the euclidean distance between the points;

- second, S point groups are formed by assigning the original N input points to the appropriate centroids, according to some method. The most common one is *Ball Query Sampling* (BQS) where, given a radius value r and a maximum number of points k, each centroid is assigned k points that lie within a distance r from the centroid. In case not enough points are present to reach k samples, the group is padded by repeating some of the points. Often, to create stronger hierarchies, multiple radii are adopted per centroid, such that features can be computed at different scales.

Another viable option for point grouping is *k-Nearest Neighbors* (kNN), in which each centroid is assigned the k points closest to it. The upside of kNN over BQS is that no padding is ever required, as there is no limitation imposed by radius and therefore k points can always be found. Also, kNN ensures that the selected points for each centroid are always the k points closest to it, which is not guaranteed when using BQS. On the other hand, however, it is less efficient overall and it does not allow to control the spatial extent of each group: regions that are less dense often lead to bigger groups, as the search must proceed further in space to find enough points;

- third, a shared PointNet is applied to each group individually in order to compute its local signature. In case more than one radius is used to create groups, a different PointNet is used for each scale, and the signature is obtained by fusing the outputs of each PointNet.

  Note that each input point encodes two pieces of information: its current feature representation, either computed by preceding SA layers, provided by the sensor or empty, and its position with respect to its group centroid. The use of a local systems of reference for each group allows to better capture the relationships between the points contained in them.

As a result, the output of the SA layer consists of a new set containing S points, located at the centroid positions and encoding information about their local neighborhoods. By stacking multiple SA layers in a cascaded way (with increasing radii in case BQS is used) it is possible to model hierarchies of representations, similarly to how CNNs do for images. A schematic representation of a SA layer is displayed in Fig. 4.1.

Each SA layer induces a subsampling of its input set of points. In order to restore the original resolution (which might be required for tasks such as semantic segmentation), *Feature Propagation* (FP) layers are introduced. FP layers can be interpreted as the inverse operation to the SA layers, and their purpose is to propagate representations downwards along the hierarchy: sup-

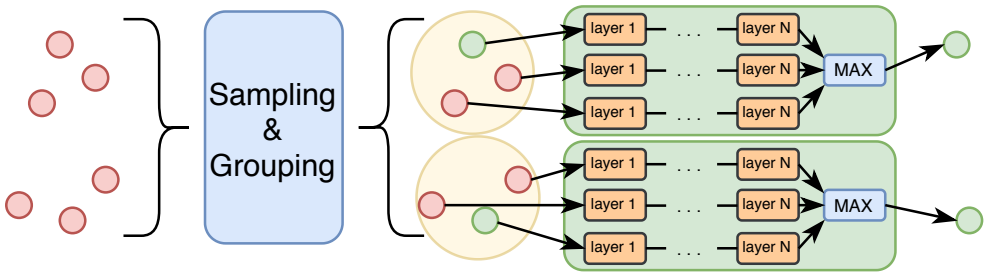


Figure 4.1: Schematic representation of the Set Abstraction layer used in PointNet++ for hierarchical feature extraction. First, the input points (here displayed in red) are sampled. Then, given the samples, point groups are formed, highlighted using yellow spheres. Finally, each point in each group is processed by a *shared* set of layers (commonly implemented as FC-BN-ReLU) followed by max pooling for feature aggregation, yielding the output features for the group. Note that the processing and aggregation network is also *shared* among different groups.

pose that the i-th SA layer takes as input the set of points $\mathbf{x}_{i-1}$ containing S_{i-1} elements, and returns the set of points $\mathbf{x}_i$ containing S_i elements, where $S_i < S_{i-1}$. The FP layer operates the following way:

- given each input point in $\mathbf{x}_{i-1}$, it identifies the 3 nearest points in $\mathbf{x}_i$;
- the features of the 3 selected points are interpolated by performing a weighted average based on their inverse distance from the input point;
- the interpolated features are fused with the original features of each input point using an MLP.

As a result, the output of the FP layer is the original set $\mathbf{x}_{i-1}$ of points, updated with the extra local information extracted by the subsequent SA layers. By applying FP layers in cascade it is possible to progressively upsample the point cloud returned by the SA layers back to its original resolution while retaining local information.

4.2.2 Baseline Votenet Model for 3D Object Detection

First introduced in [26], Votenet represents a simple yet effective approach for performing 3D object detection directly from a raw point cloud. It builds upon PointNet++ and consists of three different modules, each being a different neural network, applied sequentially to the input:

- **backbone**: this network is responsible for extracting features from the input point cloud, originating a subsampled version of it in which each point encodes information about its neighborhood;
- **voter**: given the subsampled point cloud returned by the backbone, this module is supervised to push each point close to the center of the object it belongs to, leading to the formation of clusters in correspondence to objects;
- **detector**: this module is responsible for analyzing each individual cluster and determine the state of the object it represents (if any).

Backbone As first step, the raw input cloud is processed by a backbone network in order to extract feature representations and, at the same time, determine a set of locations in space to use as starting point for detection. The backbone is implemented as a PointNet++ model having n SA layers followed by m FP layers, where $m < n$. The output of this module is therefore a subsampled version of the input point cloud, with each point encoding information about a local region of the input point cloud. These points are called *seeds*.

Voter Given the set of seeds returned by the backbone, the voter is responsible for pushing each one close to the center of the object it belongs to (if any). More specifically, given each seed point $\mathrm{s}_i \in \mathbb{R}^{3+C}$, where the first three dimensions represent its position $\mathrm{x}_i = (x_i, y_i, z_i)$ in space and the remanining dimensions encode its features, the voter outputs offsets $\Delta\mathrm{s}_i \in \mathbb{R}^{3+C}$ to apply to both position and features. The idea behind this module is to aggregate seeds belonging to the same object together in space to form clusters. Intuitively, each of these points encodes information about different regions of the scene, and therefore different object parts: as a result, each cluster can be interpreted as an entity that sums up the object as a whole by containing all of its parts, and can thus be utilized to determine the state of said object. This module is implemented as a MLP applied in parallel to each seed.

Following the original work, I train this module by explicitly supervising each point belonging to an object to be close to the object center after the shift. More specifically, if $\Delta\mathrm{x}_i$ represents the positional shift returned by the voter for the i-th seed and $\Delta\mathrm{x}_i^*$ represents the ground truth shift that would prefectly align it with its corresponding object center, the loss for the voter is given by:

$$\mathcal{L}_{\text{vote}} = \frac{1}{N_{pos}} \sum_i \|\Delta\mathrm{x}_i - \Delta\mathrm{x}_i^*\| \cdot [\mathrm{s}_i \text{ in object}] \tag{4.2}$$

The Iverson bracket function $[\mathrm{s}_i$ in object$]$ indicates that the loss function is computed only for those points who belong to an object, as background points do not have an associated center, and N_{pos} represents the number of said points.

To determine whether each point is background or belongs to an object, it is checked against each ground truth 3D box: if it is contained in one of them, it is considered positive and associated to that object. Note that no explicit supervision is performed on the feature-level shifts.

The shifted points generated by this module are called *votes*.

Detector Given the set of votes, the detection module is responsible for clustering them together and processing each cluster in order to estimate the potential object represented by it. The clustering process is achieved naturally by applying a SA layer to the votes: FPS selects the cluster centers by randomly sampling a subset of the votes; given the centers, BQS with a small radius value is used to determine the clusters; finally, the shared PointNet is applied to each cluster in order to aggregate the information and compute its signature.

Once the information about the clusters is obtained, a further submodule, represented by a MLP, is applied to each cluster in parallel to predict the boxes. The original box encoding mostly follows Frustum-PointNets [23]. More specifically, the prediction submodule estimates a total of $5 + 4 \times \mathrm{NS} + 2 \times \mathrm{NH} + \mathrm{NC}$ values per cluster, where:

- the first 2 values represent a discrete probability distribution indicating whether the cluster is background or belongs to an object;
- the following 3 values encode the correction from the cluster center to the object center, much like how the voter returns the correction from each seed to its corresponding object center;
- the subsequent $4 \times \mathrm{NS} + 2 \times \mathrm{NH}$ values encode dimension and orientation estimations respectively, encoded as classification followed by bin correction identically to [23] (see Sec. 3.6.1 for more details);
- the final NC values encode a discrete probability distribution over the possible object classes.

For simplicity and efficiency, I modify this formulation slightly, outputting $3 + 4 \times \text{NS} + 2 \times \text{NH}$ values instead:

- first, I notice that in most situations there exists exactly one size template per class, meaning that NS = NC. When this is the case, the predictor for the size template and the predictor for the object class carry out exactly the same task, rendering one of the two redundant. As a result, I opt for removing the size template predictor, and at test time I choose the right template according to the predicted object class;
- second, instead of predicting the cluster objectness (2) as well as a discrete probability distribution over the classes (NS), I predict NS individual per-class objectness values instead.

This alternative formulation performs on-par with the original, whilst being more compact and requiring $2 + \text{NS}$ less outputs.

Optimization of box centers, dimensions and orientations follows closely that of Frustum-PointNets, with the difference that in this case these losses are computed for each positive cluster (i.e. clusters whose center is contained in a ground truth box), and averaged over their number. To train the objectness predictor, NS individual binary cross-entropy losses (Eq. 2.8) are applied to each cluster, one for each class, where the ground truth probability is equal to 1 if the cluster center is inside an object of that class, 0 otherwise. Again, the final classification loss is obtained by averaging all loss values for all clusters.

At inference time, duplicate detections are handled by using Non-Maximum Suppression, favoring those having higher objectness score in case of high overlap with other detections.

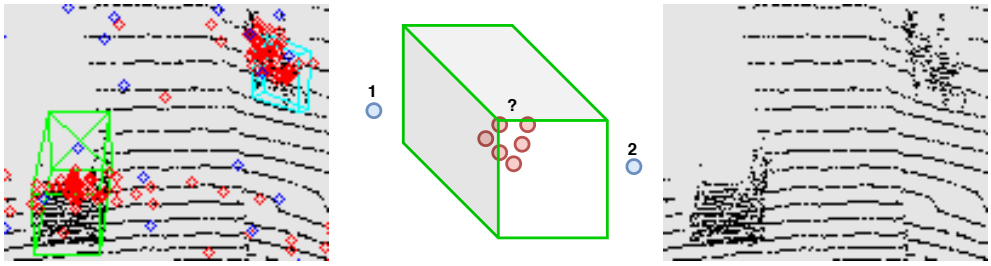
4.2.3 Modifications to the baseline for Autonomous Driving Scenarios

The Votenet baseline presented above was originally thought to perform 3D object detection on controlled scenarios, such as the indoor scenes depicted in

the ScanNet [104] and SUN RGB-D [103] datasets. Here, point clouds tend to be quite dense, and most points usually belong to objects, with a very small percentage of them that are background (i.e. points on the room floor or walls). Driving scenarios captured by LiDAR sensors, on the other hand, are substantially different: objects of interest (e.g. cars, pedestrians, cyclists etc...) are very few and far between, which leaves most of the point cloud to be background. Moreover, LiDAR clouds are generally noisier and sparses. For these reasons, the original formulation of Votenet does not adapt well to this new domain, losing performance and often missing detections.

I identify the root cause of this performance loss in the way that cluster center selection is performed. To recall from the previous section, cluster creation is accomplished through the use of a SA layer, which first determines a set of centers via the FPS algorithm and then aggregates information near each center. In case of controlled scenes, where most points belong to objects, it is likely that the clusters formed by the votes are far apart from each other, with relatively few isolated points in between. In driving scenarios, on the other hand, most points are likely to be background, which leads the clusters that form in correspondence of objects to be surrounded by numerous isolated votes. Since FPS determines the centers by starting from a random point and iteratively selecting the next ones such that they the farthest apart from the already sampled set, if some noise points near a cluster happen to be selected by FPS before any of the points belonging to the cluster, then all points of the cluster might be ignored. This behavior is displayed in Fig. 4.2. When this happens, no features are extracted by the SA layer for that cluster, which is subsequently ignored by the detection module. This has two major implications: due to some objects being missed, less positive samples are propagated to the detector during training, reducing the amount of feedback and slowing down optimization. Moreover, the same phenomenon might happen at test time, leading to false negatives that are not caused by misclassifications, but rather by objects being skipped when sampling. Both of these aspects severely undermine the overall performance of the model. To limit the performance

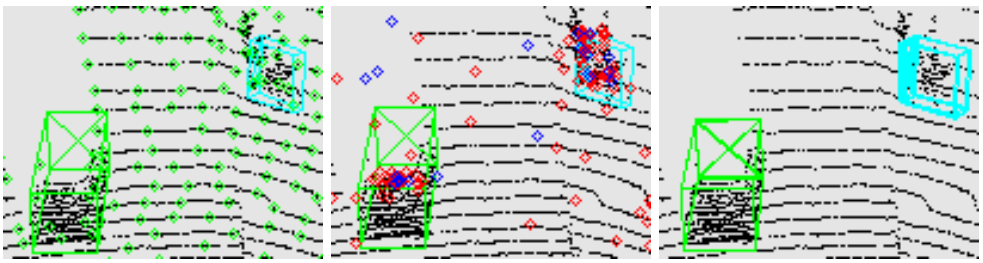
loss induced by missed clusters, I test two different solutions to cluster center sampling: seed-based sampling and feature distance-based sampling [73].



(a) (b) (c)

Figure 4.2: Examples of false negatives originated by the sampling strategy. (a) Vote points, displayed in red, sampled cluster centers, shown in blue and **ground truth** boxes for a car and a cyclist. (b) Schematic representation of what is happening: if noise points around the object are sampled first, the entire cluster might be skipped due to vicinity. (c) Resulting predicted boxes: no object is detected due to no cluster centers being sampled.

Seed-based sampling Mentioned in the original work [26], the idea behind seed-based sampling is rather simple: when determining the set of cluster centers from the votes using FPS, instead of using the coordinates of the votes, the coordinates of their corresponding seeds are used instead. Since seeds are evenly spread out in space, as they are the result of applying FPS multiple times on the original point cloud, it is unlikely that all seeds that correspond to a cluster of votes are skipped when sampling the cluster centers. Whilst contributing to almost no performance gain when the system is applied on the ScanNet and SUN RGB-D datasets, this alternative sampling strategy accounts for most of the performance gain when operating on the noisier KITTI driving scenarios. An example of result is displayed in Fig. 4.3.

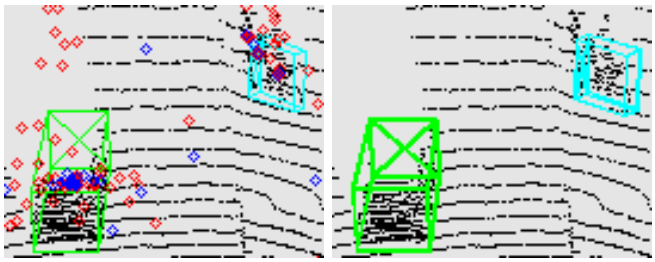


(a) (b) (c)

Figure 4.3: Detection results from the seed-based sampling strategy. (a) Locations of the seed points used for sampling, displayed in green along with the **ground truth** boxes. (b) Vote points, in red, and sampled cluster centers, in blue, along with the **ground truth** boxes. It can be observed that multiple centers per cluster are sampled. (c) Resulting predicted boxes.

Feature distance-based sampling In the recent work 3DSSD [73], the authors propose F-FPS, a variant of the FPS algorithm in which the metric is given by the sum of euclidean and feature distance between the set elements, as opposed to the traditional formulation which only considers euclidean distance and is therefore labelled as D-FPS. They show that using a combination of F-FPS and D-FPS, which they call FS (Fusion Sampling), inside the SA layers of the backbone leads to more object points being kept, which results in higher recall and overall better detection performance. This improvement is caused by the fact that F-FPS allows for points that are close to each other to be sampled, provided that these points encode different entities in space, such as different object parts. Conversely, background points such as points on the road are less likely to be chosen, as they tend to have similar feature embeddings. As alternative solution to seed sampling, I propose to use F-FPS instead of D-FPS in the SA layer of the detection module: this avoids missed clusters, as points belonging to said clusters encode different information from the surrounding background points, and thus are likely to be selected even if the latter are

chosen first. An example of result is displayed in Fig. 4.4.



(a) (b)

Figure 4.4: Detection results from the feature distance-based sampling strategy. (a) Vote points, in red, and sampled cluster centers, in blue, along with the **ground truth** boxes. This method, on average, leads to more objects points to be selected when compared to seed-based sampling. (b) Resulting predicted boxes.

Given the increase in recall shown in 3DSSD, I also opt for adopting FS in the backbone module, following the original implementation. Moreover, instead of predicting objectness like in the original system, I choose to predict centerness instead, due to its synergy with the sampling strategies above.

Centerness estimation Whilst sampling cluster centers via D-FPS results in at most one center per cluster, when using either seed or F-FPS based sampling it is likely that multiple centers per cluster are sampled. As a result, each object is likely to be detected multiple times by the detection module. While duplicate detections are handled by NMS, where the estimated confidence/objectness is the determining factor in choosing which of the multiple detections is kept, there is no real correlation between said confidence and the true quality of the predicted box. Cluster centers that are closer to their corresponding object centers, however, often result in higher quality predictions. As a result, following [73], instead of predicting a per-class objectness for each

cluster center like in Sec. 4.2.2, I predict its per-class centerness values instead. More specifically, if a cluster center is inside a ground truth box of a specific class, it is trained to predict the following value

$$p_{\text{ctr}} = \sqrt[3]{\frac{\min(f,b)}{\max(f,b)} \cdot \frac{\min(l,r)}{\max(l,r)} \cdot \frac{\min(t,d)}{\max(t,d)}} \tag{4.3}$$

and 0 otherwise. In the above equation, $\{f, b, l, r, t, d\}$ stand to indicate the distances of the cluster center from the front, back, left, right, top and bottom faces of the assigned ground truth box respectively. By predicting centerness instead of objectness, the subsequent NMS prioritizes boxes originated from cluster centers that are closer to the object center, and thus having on average higher quality, which leads to overall better performance of the system. To train for centerness prediction, I adopt the standard binary cross-entropy loss function using as target, instead of a binary 0/1 value like for the objectness, the ground truth centerness value.

4.3 Enhancing SA layers via Self-Attention

While PointNet++ represents a strong model for extracting features from point clouds that are local and hierarchical, its structure suffers from a limitation. In SA layers, the feature aggregation of each group is performed through the use of a shared PointNet. To achieve permutation invariance, PointNet processes each point in each group *independently* using a MLP and then aggregates their information via a max pooling operation. As a result, the points in each group are *unaware of each other* while being processed by the MLP, which likely leads to suboptimal feature representations.

A possible solution for circumventing this problem has been proposed in PointWeb [105], where a feature adjustment module is introduced before the MLP to recalibrate the features of each point according to its relationship with all other points. Instead, I propose to exploit the attention mechanism to explicitly model inter-point relationships within each group.

4.3.1 The attention mechanism

Attention can be defined as a function mapping three sets of elements, called *queries*, *keys* and *values* into a set of output elements. More specifically, each *query* element corresponds to an output, which is given by a weighted sum of the *values* where the weight associated to each *value* depends on the compatibility of the *query* with the *key* corresponding to that *value*. Formally, let $Q \in \mathbb{R}^{n_q \times d_k}$ be the set of queries, $K \in \mathbb{R}^{n_{kv} \times d_k}$ the set of keys and $V \in \mathbb{R}^{n_{kv} \times d_v}$ the set of values, where n_q indicates the number of query elements, n_{kv} indicates the number of key/value pairs, d_k indicates the dimensionality of queries and keys and d_v the dimensionality of the values. Then, the most commonly adopted version of attention [27] is as follows:

$$\mathrm{Att}\left(Q, K, V\right) = \Omega\left(Q \cdot K^{\mathrm{T}}\right) \cdot V. \tag{4.4}$$

Here, the pairwise dot product $Q \cdot K^T \in \mathbb{R}^{n_q \times n_{kv}}$ measures how compatible each *query* is with each *key*, and $\Omega\left(\cdot\right)$ is a function mapping the compatibility values into weights. Usually, $\Omega\left(\cdot\right)$ takes the form of a Softmax function, applied to each row of $Q \cdot K^T$ independently.

The sets of *queries*, *keys* and *values* can be used to model any quantity, so long as they can be encoded in the form of vectors. Commonly, in deep learning these quantities are obtained by projecting feature representations extracted by one or more neural networks:

$$Q = P_Q(\mathbf{x}_q), \qquad K = P_K(\mathbf{x}_k), \qquad V = P_V(\mathbf{x}_v). \tag{4.5}$$

Here, $\mathbf{x}_q$, $\mathbf{x}_k$, $\mathbf{x}_v$ could be, for instance, feature maps returned by CNNs or points extracted by a PointNet++ backbone. In the former case, each pixel represents an element of the set; in the latter, each point represent an element of the set. $P_Q\left(\cdot\right)$, $P_K\left(\cdot\right)$, $P_V\left(\cdot\right)$ represent the projection functions used to map each of these representations into the sets of *queries*, *keys* and *values*. Such functions are usually optimized together with the rest of the models, and are commonly implemented using an MLP applied concurrently to each element

or, in the simplest case, as a matrix multiplication, where the parameters of the matrices are trainable. In most cases, *keys* and *values* are obtained from the same set of features, that is $\mathbf{x}_{kv} = \mathbf{x}_k = \mathbf{x}_v$. Also, by imposing the dimentionality of the *values* to be the same as that of the *keys* and *queries* (i.e. $d = d_k = d_v$), the outputs returned by the attention function are often used to update the original set of query features:

$$\mathbf{y} = \text{Att}(P_Q(\mathbf{x}_q), P_K(\mathbf{x}_{kv}), P_V(\mathbf{x}_{kv})) + \mathbf{x}_q. \tag{4.6}$$

The resulting set of features $\mathbf{y}$ can therefore be interpreted as an updated version of the original set of features $\mathbf{x}_q$, in which the update depends on the relationship between each element in the set $\mathbf{x}_q$ and all the elements in the set $\mathbf{x}_{kv}$. Note that this operation is invariant to the permutation of the elements in the set $\mathbf{x}_{kv}$ and equivariant to the permutation of the elements in the set $\mathbf{x}_q$, meaning that permuting the elements in $\mathbf{x}_q$ induces the same permutation on the set $\mathbf{y}$, but the individual values do not change. **Self-attention** is a variant of attention in which *queries*, *keys* and *values* are all derived from the *same set of features*, that is $\mathbf{x} = \mathbf{x}_q = \mathbf{x}_k = \mathbf{x}_v$. Self-attention can therefore be interpreted as an operation that updates each element in $\mathbf{x}$ depending on its relationship with the rest of the elements in same set.

Arguably the first successful application of self-attention is represented by the seminal work in [27], in which the authors propose an entirely novel neural architecture to perform the task of machine translation revolving entirely around the attention mechanism. More specifically, they introduce the Transformer, an Encoder-Decoder structure: in the Encoder, self-attention is used to model the relationship between the different words in the input sentence. In the Decoder, self-attention is first used to extract information from the translated sentence generated so far; then, by using these features as queries, and the features generated by the Encoder as keys/values, additional attention blocks are used to generate the set of features used for next word prediction. Perhaps one of the most significant innovations introduced in the Transformer model is, however, Multi-Headed Attention. The idea behind this concept is as follows:

instead of projecting the feature sets $\mathbf{x}_q$ and $\mathbf{x}_{kv}$ into a single set of *queries*, *keys* and *values*, they are projected into h of these sets instead:

$$\begin{aligned} P_Q\left(\mathbf{x}_q\right) &= Q = \left[Q^{(1)}, \ldots, Q^{(h)}\right] \in \mathbb{R}^{n_q \times (h \cdot d_h)}, \\ P_K\left(\mathbf{x}_{kv}\right) &= K = \left[K^{(1)}, \ldots, K^{(h)}\right] \in \mathbb{R}^{n_k \times (h \cdot d_h)}, \\ P_V\left(\mathbf{x}_{kv}\right) &= V = \left[V^{(1)}, \ldots, V^{(h)}\right] \in \mathbb{R}^{n_k \times (h \cdot d_h)}. \end{aligned} \tag{4.7}$$

Each of these sets is then processed independently and in parallel using attention, and their outputs are fused together via an additional projection function $P_O\left(\cdot\right)$, yielding the output of the Multi-Headed Attention:

$$\mathrm{MHA}\left(Q, K, V\right) = P_O\left(\left[\mathrm{Att}_1, \ldots, \mathrm{Att}_h\right]\right) \in \mathbb{R}^{n_q \times d}. \tag{4.8}$$

Here, $\left[\mathrm{Att}_1, \ldots, \mathrm{Att}_h\right] \in \mathbb{R}^{n_q \times (h \cdot d_h)}$ and $\mathrm{Att}_i = \mathrm{Att}\left(Q^{(i)}, K^{(i)}, V^{(i)}\right)$. Again, the projection function $P_O\left(\cdot\right)$ is implemented either as an MLP or, more commonly, as a matrix multiplication with learnable matrix parameters. Like before, the result of the Multi-Headed Attention is finally used to update the input set of query features:

$$\mathbf{y} = \mathrm{MHA}(P_Q\left(\mathbf{x}_q\right), P_K\left(\mathbf{x}_{kv}\right), P_V\left(\mathbf{x}_{kv}\right)) + \mathbf{x}_q. \tag{4.9}$$

The authors argue that using multiple, independent attention heads allows the model to concentrate on different aspects of the input at different positions, improving the overall performance of the system. Note that by choosing $d_h = d/h$ and parallelizing the computation of the several attention components, Multi-Headed Attention introduces no computational overhead compared to the traditional formulation. To better fuse the input features with those computed by the Multi-Headed Attention and improve training, each Multi-Headed Attention Block (MAB) in the Transformer encoder adopts Layer Normalization [106] for feature normalization, as well as an additional MLP for performing additional feature fusion, leading to the following final formulation:

$$\mathrm{MAB}\left(\mathbf{x}_q, \mathbf{x}_{kv}\right) = \mathrm{LN}\left(\widetilde{\mathbf{y}} + \mathrm{MLP}\left(\widetilde{\mathbf{y}}\right)\right), \tag{4.10}$$

where $\widetilde{\mathbf{y}} = \mathrm{LN}(\mathbf{y})$, $\mathbf{y}$ is obtained using Eq. 4.9, MLP indicates the MLP used for feature fusion and LN represent the Layer Normalization operator. A schematic representation of the MAB block is depicted in Fig. 4.5.

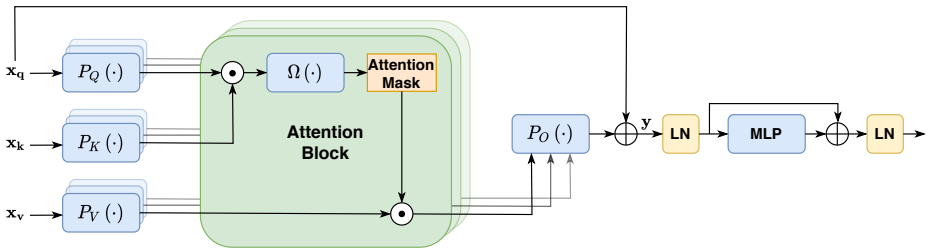


Figure 4.5: Schematic representation of a MAB block.

4.3.2 Self-attention applied to SA layers

I propose to replace both the MLP for feature extraction and the max-pooling for feature aggregation inside SA layers with attention-based processing, in order to enhance the extracted feature representations by explicitly modeling the relationships between the points within each group. In order to be effective, such attention-based mechanisms should preserve the property of invariance to permutations of the original formulation. More specifically, I introduce two different formulations to replace the MLP.

Full Attention Here, I propose to replace each layer of the SA MLP $h(\mathbf{x})$ (see Eq. 4.1) with a self-attentive MAB block $\mathrm{MAB}(\mathbf{x}, \mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^{K \times C}$ represents any single group of points returned by the sampling and grouping stages of the SA layer, K is the number of points in the group and C is the dimensionality of each point. This can be done directly, as $\mathrm{MAB}(\mathbf{x}, \mathbf{x})$ is permutation equivariant. Commonly, most models progressively increase the dimensionality C of the features they extract as their depth increases. To allow for the same flexibility when using attention, I introduce an additional

projection operation $P_C(\cdot)$ inside the MAB block

$$\mathrm{MAB}(\mathbf{x}, \mathbf{x}) = \mathrm{LN}\left(P_C(\widetilde{\mathbf{y}}) + \mathrm{MLP}(\widetilde{\mathbf{y}})\right), \tag{4.11}$$

which is tasked to project $\widetilde{\mathbf{y}}$ in a space such that its dimensionality matches that of $\mathrm{MLP}(\widetilde{\mathbf{y}})$, much like how projection shortcuts are used in ResNets [3] to match input and output dimensionalities in residual blocks. This enables the MLP to extract features having different dimensionality than those of the input, and therefore allows each MAB block to modulate the number of channels just like FC layers do in the original implementation. Like all other projections, $P_C(\cdot)$ is implemented as a learnable linear transformation, and it is set to the identity whenever the dimensionality returned by $\mathrm{MLP}(\widetilde{\mathbf{y}})$ is the same as that of $\widetilde{\mathbf{y}}$. A schematic representation of the proposed full attention is displayed in Fig. 4.6.

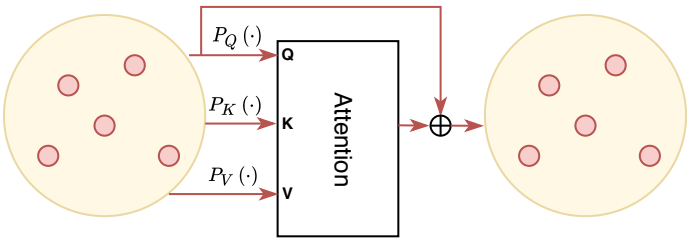


Figure 4.6: Schematic representation of the full attention block. For simplicity, LN operations, projections and additional MLPs for feature fusion are omitted.

Induced Attention Whilst being general and relatively simple, full self-attention is expensive: the space and time cost for computing and storing the compatibility matrix $Q \cdot K^T$ is quadratic in the number of elements in each group, that is $\mathcal{O}\left(K^2\right)$. If S is the number of groups extracted by the SA layer, this would bring the total cost up to $\mathcal{O}\left(S \cdot K^2\right)$, which might be intractable if the number of groups S or the size of each group K is big.

In [107], the authors introduced a variant to the attention mechanism in which self-attention on points is not performed directly, but rather through an intermediate set of elements: the *inducing points*. First, a MAB block is used to update the set of inducing points $\mathbf{z} \in \mathbb{R}^{I \times C}$ with the input points information: to do so, the inducing points are used as *queries* and the input points $\mathbf{x}$ are used as *keys/values*

$$\mathbf{z}' = \text{MAB}\left(\mathbf{z}, \mathbf{x}\right), \tag{4.12}$$

where $\mathbf{z}' \in \mathbb{R}^{I \times C}$ is used to represent the set of updated inducing points. Then, the update on the input set of points is performed by attending them using the updated set of inducing points:

$$\mathbf{o} = \text{MAB}\left(\mathbf{x}, \mathbf{z}'\right), \tag{4.13}$$

where $\mathbf{o} \in \mathbb{R}^{K \times C'}$ represents the updated set of input points.

In the original work, the set of inducing points $\mathbf{z}$ is part of the layer and optimized during training. In the shown experiments, however, induced attention is always applied to the *entire* point cloud, and the datasets used are mostly toy examples. As a result, such a solution is likely to work poorly when applied to the parallel processing of the many heterogeneous groups of points within SA layers. These groups are likely to contain very different structures and patterns, which renders learning a single set of inducing points to be used for all of them challenging and suboptimal. For this reason, I propose to learn a linear transformation that maps the *signature* of each group into the set of inducing points instead:

$$\mathbf{z} = \text{LN}\left(\max\left(\mathbf{x}\right) \cdot W_z^T\right), \tag{4.14}$$

where $\max\left(\mathbf{x}\right) \in \mathbb{R}^{1 \times C}$ represents the signature for the group, computed as usual via max-pooling, $W_z \in \mathbb{R}^{I \cdot C \times C}$ is the matrix of learnable weights mapping the signature into the set of inducing points, and LN represents the Layer Normalization, performed over the C channels. More specifically, the output of the linear transformation $\max\left(\mathbf{x}\right) \cdot W_z^T \in \mathbb{R}^{1 \times I \cdot C}$ is reshaped to be in $\mathbb{R}^{I \times C}$

before applying LN and obtaining the inducing points. This formulation still allows the inducing points to be learned from data, but does not force the same set of inducing points to be used for every group, as they are now a function of the group signature.

In both cases, the resulting induced attention has complexity $\mathcal{O}(S \cdot I \cdot K)$, which now grows linearly with the number of group elements instead of quadratically, leading to reduced memory and time costs especially for groups of large size ($K \gg 1$). Moreover, the proposed formulation is still equivariant to permutation, given the permutation invariant nature of the inducing points calculation. A schematic representation of the proposed induced attention is displayed in Fig. 4.7.

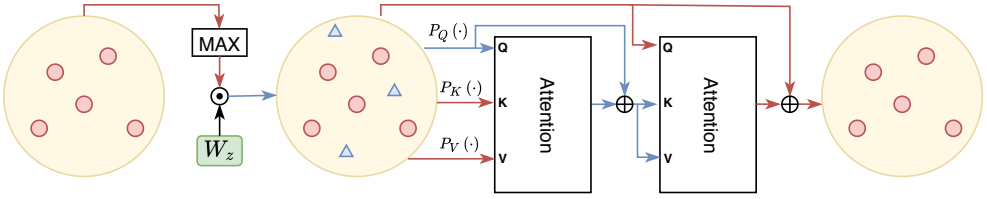


Figure 4.7: Schematic representation of the induced attention block. Group points are represented by red dots, whereas inducing points are depicted by blue triangles. Red arrows are used to represent information flow pertaining to the group points, while blue arrows represent information flows regarding inducing points. For simplicity, LN operations and additional MLPs for feature fusion are omitted.

Besides enhancing feature extraction by using self-attention, following [107] I also strengthen the aggregation phase $g(\cdot)$ (Eq. 4.1). Instead of using a simple max-pooling layer, I opt for using the first half of an induced attention layer (i.e. Eq. 4.12) with a single inducing point to perform aggregation. Like before, the inducing point is computed using Eq. 4.14. This formulation allows the model to weigh the contribution of each element, extending the set of functions that

the SA layer can approximate. Like max-pooling, it is permutation invariant.

4.4 Training Setup

4.4.1 KITTI LiDAR Dataset

Like for the monocular 3D detector presented in the previous chapter, the proposed system is trained and evaluated on the KITTI [22] autonomous driving dataset, already introduced in Sec. 3.5.1. Besides providing 7481 training images with the respective annotated 3D bounding boxes, this dataset also supplies the corresponding, calibrated, 360° Velodyne scans. Since 3D annotations only encompass the field of view of the camera, these scans are filtered before providing them to the model, keeping only the points within the camera viewing frustum. This leaves each scan to be comprised of approximately 17.000 points: in order to ensure a consistent number of input points among different samples, I randomly select 16384 points from each scan before forwarding them to the model. Besides the position of each point, I also provide its reflectance value returned by the sensor as additional input feature. Again, following previous work [67, 17, 76], the annotated data is split into training and validation, comprised of 3712 and 3769 samples respectively.

4.4.2 Model architecture

The proposed model architecture adopted for the experiments, unless stated otherwise, is depicted in Fig. 4.8.

The **backbone** is a standard PointNet++ model, comprised of three SA layers followed by a FP layer, used for feature extraction and seed point creation. More specifically, the SA layers downsample the input cloud down to 4096, 1024 and 512 points respectively, and the FP layer upsamples it once, yielding a total of 1024 seed points. For *point sampling*, I adopt D-FPS in the first SA layer (as input points do not carry feature representations) and FS in the following two SA layers, meaning that half the points are sampled

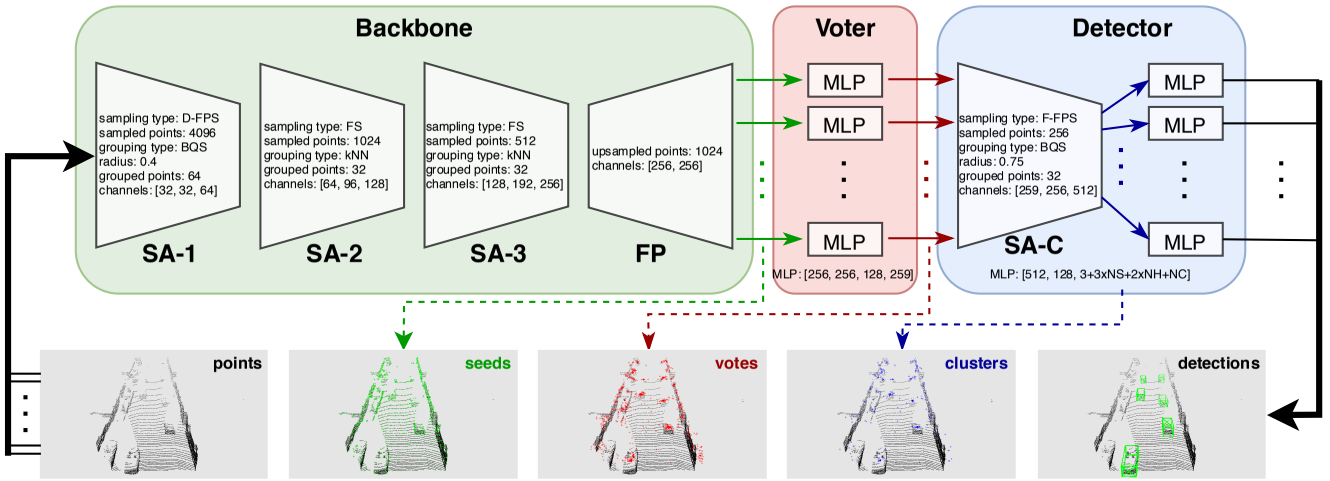


Figure 4.8: Schematic representation of the proposed pipeline for 3D object detection from point cloud. Each individual SA layer specifies the algorithm adopted for point sampling (sampling type), the number of sampled points (sampled points), the algorithm adopted for point grouping (grouping type), the number of points for each group (grouped points) and the number of output channels of each layer used for point processing (channels). In case BQS is used for grouping, the radius value of the grouping sphere is also displayed (radius). The FP layer specifies the number of points resulting from the upsampling (upsampled points) and the output channels of the FC layers used for feature fusion. Voter and Detector blocks also specify the output channels of the MLPs used for vote prediction and box estimation.

via D-FPS and the remaining half using F-FPS. For *point grouping* I adopt a hybrid approach: in the first SA layer I use BQS, as it allows to control the receptive field of each center and scales well for a large number of input points; for the subsequent SA layers I adopt kNN instead. While being less efficient than BQS, kNN still performs acceptably well on a reduced set of points; it also avoids point repetition, which might interfere with the attention mechanism. The drawback of kNN is that the receptive field is not fixed, but rather it depends on the density of the points in the region: by using attention, however, the model should be able to detect outlying points, ignoring them. I embed and test both full-attention and induced-attention in the layers SA2 and SA3. The integration is rather straightforward: each FC layer of the original SA layer MLP is swapped with either a full-attention block or an induced-attention block, maintaining the exact same channel scaling as the baseline model. I also substitute feature aggregation by max-pooling with its attention based counterpart, as explained in Sec. 4.3.2. I avoid using attention mechanisms in SA1 both due to the absence of proper feature representations associated to the points and due to the high number of groups (4096) and large size of each group (64), which render attention quite costly to compute.

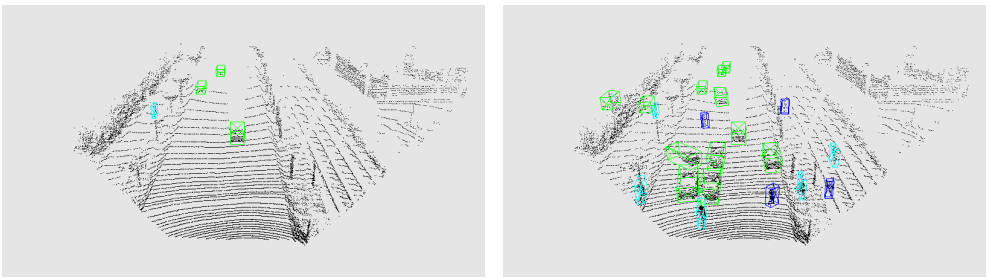
The **voter** is implemented as a standard MLP, composed by two fully-connected layers with batch normalization and ReLU activation followed by a third fully-connected layer for position and features shift prediction.

The **detector** is comprised of a SA layer, which identifies 256 clusters among the votes and processes them, followed by an MLP predictor, applied to each cluster for bounding box prediction. I adopt F-FPS as default choice for cluster center sampling, as it avoids the problem of missed clusters. However, I also test the original setup, which adopts D-FPS, as well as seed-based sampling in the experiments section. For point aggregation I choose BQS with small radius, as it exploits the fact that votes should form clusters in space and it performs better than kNN. The MLP predictor has the same structure as that of the voter, and the predictions are encoded as presented in Sec. 4.2.2.

4.4.3 Training Routine

All tested models are trained on the KITTI training split for a total of 80700 iterations, using ADAM as optimizer and batch size 4. The learning rate is initially set to $5 \cdot 10^{-4}$ and reduced by a factor of 10 after 64560 iterations.

Given the relatively limited dataset, I employ heavy data augmentation in order to favor generalization and improve performance. First of all, I adopt the so-called *mixup augmentation* [71], an augmentation technique often adopted in recent literature when training point-based and voxel-based detectors which consists in inserting into the current scene objects from other scenes in order to provide richer training examples to the model. Practically, a database containing all objects in the training split is generated. Then, at training time, every time a new training sample is loaded, a certain amount of randomly sampled objects from this database is pasted into it, taking care that no collisions between objects are generated in the process. An example of a mixup-augmented image is shown in Fig. 4.9.



(a) (b)

Figure 4.9: Example of mixup augmentation. (a) original sample with the associated ground truth boxes. (b) Sample after mixup augmentation.

After mixup augmentation is performed, additional data augmentation is carried out to further increase the variability of the training data:

- first, the point cloud is flipped with respect to the camera xz-plane with

probability 0.5;

- second, the point cloud is scaled by a factor randomly sampled within the interval $[0.9, 1.1]$;
- third, the point cloud is rotated about the camera y-axis by an angle randomly sampled within the interval $[-\pi/4, \pi/4]$;
- fourth, each object, and its associated points, is independently rotated about its y-axis by an angle randomly sampled within the interval $[-\pi/3, \pi/3]$;
- fifth, each object, and its associated points, is shifted along the camera x and z directions by two quantities randomly sampled within the interval $[-1, 1]$.

4.5 Experimental Results

In this section, I present the results obtained from the experiments performed on the proposed system. First, I perform an ablation study on the architectural design choices presented in sections 4.2.3 and 4.3.2, validating them. Then, I carry out a more in-depth study on attention, comparing different formulations against the vanilla model with no attention mechanisms. Finally, I perform a comparison between the proposed system and other state-of-the-art LiDAR-based 3D detection models.

I note that, despite the fact that the system is trained on all three main KITTI classes (i.e. car, pedestrian and cyclist), most comparisons are performed considering only the *car* class since, given the limited size of the training and evaluation splits (3712 and 3769 samples respectively), it is the only class that provides numerous enough ground truth samples to enable reliable analyses.

4.5.1 Ablation study on the design choice

The improvements in performance brought about by the modifications to the original Votenet architecture (Sec. 4.2.3) and by the integration of the attention mechanism (Sec. 4.3.2) are displayed in Tab. 4.1.

It can be seen that, indeed, the main cause for Votenet poor performance on noisier autonomous driving scenarios is its inadequate cluster center sampling strategy: just switching to seed-based center sampling leads to a relative improvement of about $\sim 14\%$. Note that this performance is not the result of a different neural architecture, but rather is due to the fact that seed-based sampling mostly avoids missing clusters, increasing test-time recall. Moreover, this sampling strategy often leads to the same cluster being processed multiple times by the box predictior, as it is likely that multiple points inside it end up being sampled; as a result, the network benefits from increased feedback during training, which leads to faster and better convergence and therefore higher quality detections.

Mixup augmentation contributes to a further boost in performance, despite it being quite limited for the car class and mostly isolated to the hard examples. This is likely due to the fact that cars are by far the most represented class in the dataset, while also being the most spread out among different samples. Conversely, pedestrians and cyclists are far less in number (about $1/5$ and $1/10$ compared to cars, respectively) and tend to concentrate on a few select samples [22]. Therefore, forcing each input sample to contain multiple instances of those classes stabilizes training, which is no longer dominated by cars. This leads to a considerable gain in performance for pedestrian and cyclist detection, as displayed in Tab. 4.2.

Estimating centerness over objectness further increases detection accuracy, since in this case the score assigned by the model to each predicted box better correlates with the quality of the box itself. This has two implications: on the one hand, noisy detections and outright false positives are more likely to have low scores, which causes the Precision-Recall curve to have greater area, improving the AP metric. On the other hand, it is less likely that NMS discards

	Seed Sampling	Feature Sampling	Mixup Augmentation	Centerness	Attention	Car AP 3D		
						Easy	Moderate	Hard
Votenet [26]						76.85	65.99	64.69
	✓					87.81	76.91	72.81
	✓		✓			87,37	76.96	74,73
	✓		✓	✓		87.86	78.25	77.14
Proposed		✓	✓	✓		88.49	78.55	77.38
Proposed		✓	✓	✓	Full	89.20	79.23	78.35
Proposed		✓	✓	✓	Induced (8)	**89.37**	**79.40**	**78.51**

Table 4.1: Ablation study on different variants of the proposed system over the KITTI evaluation set for the Car class.

better boxes in favor of more inaccurate ones. Unsurprisingly, centerness estimation leads to better improvements for harder samples: since these samples mostly consist of truncated or far away objects, they tend to be represented by very few points and therefore are more likely to result in noisier cluster center predictions.

Experimentally, using F-FPS on votes to sample cluster centers performs better than seed-based sampling, and thus it constitutes the default choice for the proposed vanilla model. Extending such model with attention-based blocks leads to further boosts in performance. A more detailed study related to attention follows in the next section.

	AP_{3D})			
Model	class	Easy	Moderate	Hard
Vanilla	Ped	36.89	35.03	32.03
SS	Ped	53.28	49.96	45.72
SS+MIX	Ped	61.84	56.31	51.46
Vanilla	Cyc	39.29	27.50	26.79
SS	Cyc	67.13	52.52	50.52
SS+MIX	Cyc	83.33	64.63	60.50

Table 4.2: Comparison between the vanilla system, the system with seed-based sampling (SS), and the system with seed-based sampling and mixup augmentation (SS+MIX) on the KITTI evaluation set for the Pedestrian (Ped) and Cyclist (Cyc) classes.

4.5.2 Ablation study on attention type

Point-based 3D detection methods (i.e. models based off of PointNet++) are, by construction, non-deterministic: the result of each FPS operator, in fact, depends on the first selected point in the cloud, which is chosen randomly. While this phenomenon was shown to have a negligible effect in terms of test

results stability [25], when coupled with the heavy data augmentation involved during training and the fact that the training set is relatively limited, it might lead to non-trivial differences during optimization. Consequently, in order to perform a more accurate comparison between the vanilla model and the various attention-based formulations and to reduce the impact of such randomness on the results, I train 5 identical models of each type, showing the best model out of the 5 as well as their mean performance and their standard deviation.

The results of this study are shown in Tab. 4.3. As can be observed, all attention-based models perform, on average, better than their vanilla counterpart, despite being characterized by higher training instability as indicated by their higher standard deviation values.

Surprisingly, the induced-attention based models outperform their full-attention counterparts. The reason for this might be twofold: on the one hand, induced-attention results in a higher number of trainable parameters, and therefore in a model having higher representational capacity, compared to full-attention, as each induced-attention layer is comprised of two multi-headed attention blocks instead of one (see Fig. 4.7). On the other hand, the use of inducing points might allow the model to better handle potential outlying elements within each group, reducing their contribution to the output.

Performing feature aggregation via attention instead of max-pooling also leads to relevant improvements, as it allows the model to specifically modulate the contribution to the output of each component of the group, allowing the system to represent a much wider family of functions compared to simply performing a channel-wise max-pooling. This stronger representational power, however, also leads to further training instability, as displayed by the increase in the standard deviation value of the validation AP.

I also test both full-attention and induced-attention applied to the SA layer of the detection module (SA-C), which is responsible for cluster creation and grouping. In both cases, despite the higher representational capacity, the results are inferior to their counterparts using a standard MLP followed by max-pooling. While induced-attention performs only slightly worse, full-attention

leads to a more pronounced drop in performance, resulting to a solution that on average performs worse than the vanilla model. This solution is also considerably more unstable during training, as shown by the standard deviation of its performance, which is almost four times that of the model with no attention in SA-C. This degradation in performance is likely to be attributed to the fact that SA-C utilizes BQS with low radius as grouping algorithm, which leads to many clusters having repeated points due to padding. This is especially true for votes that do not belong to objects, which are likely to be isolated in space. Repeated points lead to instabilities when used in conjuction with attention, as each one actively contributes to the end result. The obvious solution for avoiding this problem would be to replace BQS with kNN; this, however, leads to additional noise in the clusters, as each one is far more likely to include background noise points or even points from external objects. Considering that the resulting system performs worse than its BQS-based counterpart (see Tab. 4.4) while being less efficient, I opt against this kind of solution.

4.5.3 Comparison with State-of-the-Art Systems

Tab. 4.5 shows a comparison between the proposed solution and other LiDAR and fusion-based state-of-the-art 3D detection systems on the KITTI validation set.

The vanilla system performs considerably better than all other approaches barring PointRCNN, which slightly outperforms it at all difficulties. PointRCNN, however, is overall slower, exhibiting an inference time of about 100ms against the 70ms of the vanilla model.

Integrating attention in the backbone network considerably increases performance, allowing the resulting models to convincingly surpass PointRCNN while only requiring an extra 15ms for full attention and 19ms for induced-attention. This improvement is particularly relevant for harder objects, which advocates for the effectiveness of attention mechanisms to deal with harder and noisier data.

Model	Car AP_{3D}: mean ± st.d. (max)		
	Easy	Moderate	Hard
Vanilla	88.45 ± 0.17 (88.49)	78.43 ± 0.17 (78.55)	77.29 ± 0.11 (77.38)
F-23-MP	88.35 ± 0.23 (88.65)	78.59 ± 0.22 (78.90)	77.67 ± 0.19 (77.99)
F-23-AP	88.83 ± 0.24 (89.20)	78.93 ± 0.23 (79.23)	78.03 ± 0.25 (78.35)
I(8)-23-MP	88.61 ± 0.23 (88.72)	78.77 ± 0.15 (78.93)	77.79 ± 0.20 (77.89)
I(8)-23-AP	89.01 ± 0.32 (89.37)	79.00 ± 0.26 (79.40)	78.10 ± 0.24 (78.51)
F-23C-AP	88.29 ± 0.72 (89.07)	78.37 ± 0.82 (79.09)	77.34 ± 0.98 (78.15)
I(8)-23C-AP	88.92 ± 0.15 (89.15)	78.93 ± 0.20 (79.25)	78.03 ± 0.26 (78.43)

Table 4.3: Ablation study on the attention mechanism. Attention-based models are shown in the format **A-B-C**: **A** represents the attention type, where F indicates full-attention and I(n) indicates induced attention with n inducing points; **B** represents the SA layers the attention is applied to, where 2, 3 and C indicates the layers SA-2, SA-3 and SA-C repsectively (see Fig. 4.8); **C** represents the technique adopted for performing feature aggregation: MP is the standard channel-wise max-pooling, AP is the attention-based aggregation. The results are shown in the format **mean ± st.d. (max)**, over a total of 5 experiments.

SA-C	Car AP_{3D}: mean ± st.d. (max)		
	Easy	Moderate	Hard
BQS	88.45 ± 0.17 (88.49)	78.43 ± 0.17 (78.55)	77.29 ± 0.11 (77.38)
kNN	88.19 ± 0.23 (88.48)	78.11 ± 0.19 (78.33)	76.82 ± 0.30 (77.19)

Table 4.4: Performance comparison between using BQS and kNN in the SA-C layer. Both models are without attention.

Method	AP_{BEV} @ 0.7 IoU			AP_{3D} @ 0.7 IoU		
	Easy	Moderate	Hard	Easy	Moderate	Hard
MV3D [67]	86.55	78.10	76.67	71.29	62.68	56.56
F-PointNet [23]	88.16	84.02	76.44	83.76	70.92	63.65
AVOD [12]	-	-	-	84.41	74.44	68.65
VoxelNet [70]	89.60	84.81	78.57	81.97	65.46	62.85
SECOND [71]	89.96	87.07	79.66	87.43	76.48	69.10
PointPillars [13]	-	-	-	-	77.98	-
PointRCNN [13]	-	-	-	88.88	78.63	77.38
Vanilla (mean)	90.08	87.95	84.73	88.45	78.43	77.29
I(8)-23-AP (mean)	90.26	88.34	87.36	89.01	79.00	78.10
F-23-AP (mean)	90.22	88.26	87.30	88.83	78.93	78.03
Vanilla (max)	90.17	88.18	86.03	88.49	78.55	77.38
I(8)-23-AP (max)	**90.46**	**88.58**	**87.53**	**89.37**	**79.40**	**78.51**
F-23-AP (max)	90.36	88.56	87.52	89.20	79.23	78.35

Table 4.5: Performance comparison between the proposed models and state-of-the-art 3D detection systems on the KITTI validation set for the *Car* class. I highlight the best performing models, considering both mean performance and **best performance**.

4.5.4 Qualitative results

In Figs. 4.10 and 4.11 I show the results of the best performing induced attention-based model on some KITTI validation samples, displaying all three main classes: Cars, Pedestrians and Cyclists.

As can be observed, the model is capable of detecting objects with high positional accuracy, while being robust to occlusions and uncommon orientations (Figs. 4.10a, 4.10b, 4.10c, 4.11a). A common cause for detection inaccuracies is heavy object truncation, such as in Fig. 4.10d, in which case the predicted orientation might be noisy. High distances might also be problematic, as they could lead to false negatives (Fig. 4.10e) especially for the smaller classes, as the number of points might be insufficient for successful object identification.

Of all the classes, pedestrians are easily the most challenging for a LiDAR-based system due to the fact that they are small, non-rigid and have no standard structure like cars or bicycles. As a result, they are difficult to estimate, especially in terms of orientation (Fig. 4.11a), and might give rise to false positives as they are easily confused with other entities, such as traffic sign, poles or small trees. An example of this phenomenon is displayed if Fig. 4.10f, where a traffic sign is mistakenly interpreted as a pedestrian. Another challenging scenario for this kind of model is represented by situations in which many small objects are close together in space, such as groups of pedestrians (Fig. 4.11d). In these situations, vote points might erroneously gather around objects that are different from the ones they belong to: when this happens, certain objects might be left with very small clusters whose features are very similar to those of neighboring clusters, and therefore run the risk of being ignored during sampling, originating a false negative.

The examples shown in Fig. 4.11, in particular, highlight cases in which the model correctly detects objects that are clearly visible, but not annotated:

- the pedestrian on the right in Fig. 4.11a, distinctly visible in both the image and the LiDAR scan;
- the pedestrian and the cyclist on the right in Fig. 4.11b;

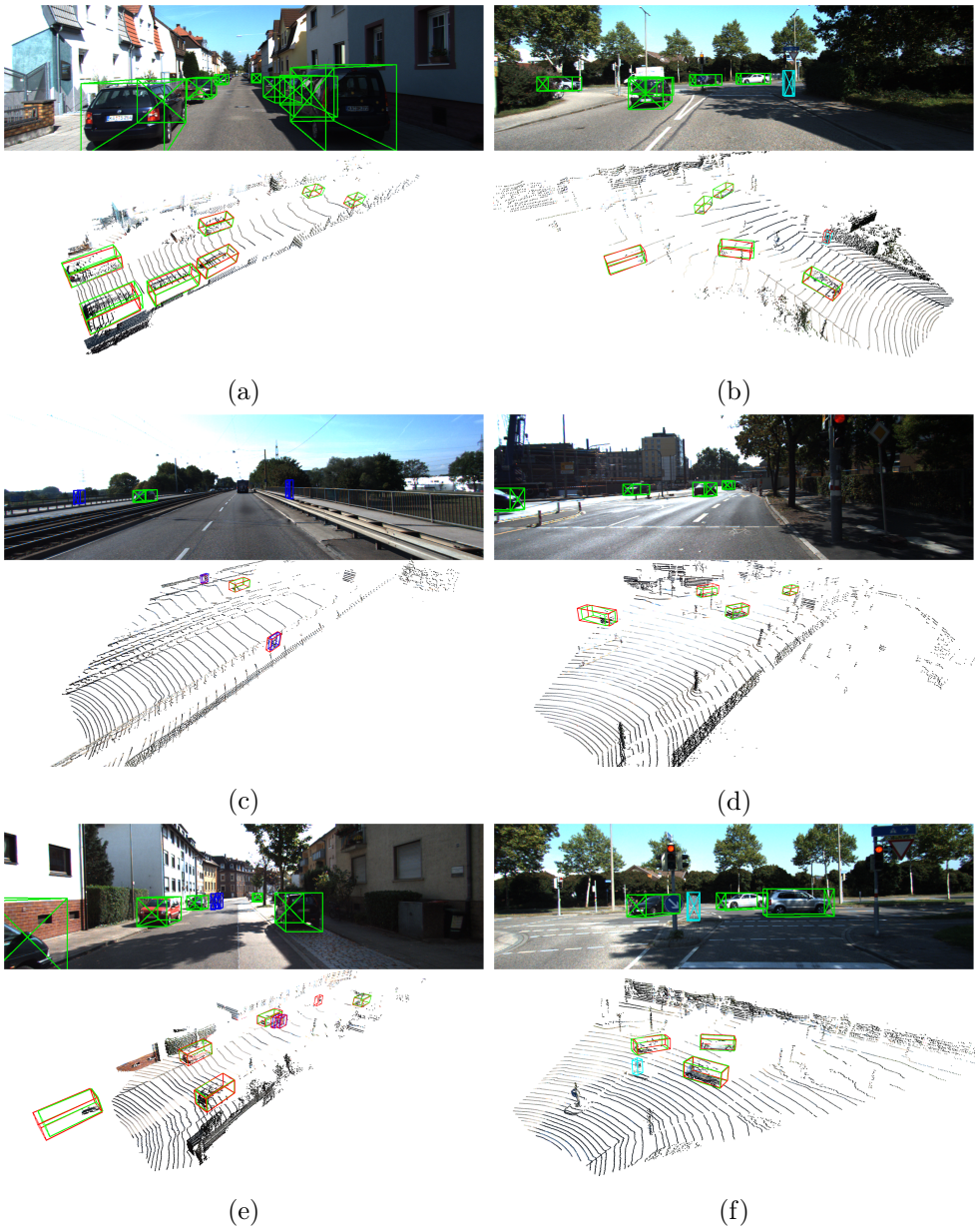


Figure 4.10: Results of the proposed method on the validation set. Red bounding boxes correspond to the ground truth, while green, cyan and blue boxes are car, pedestrian and cyclist detections respectively. Images are used exclusively for visualization purposes. Best viewed in color.

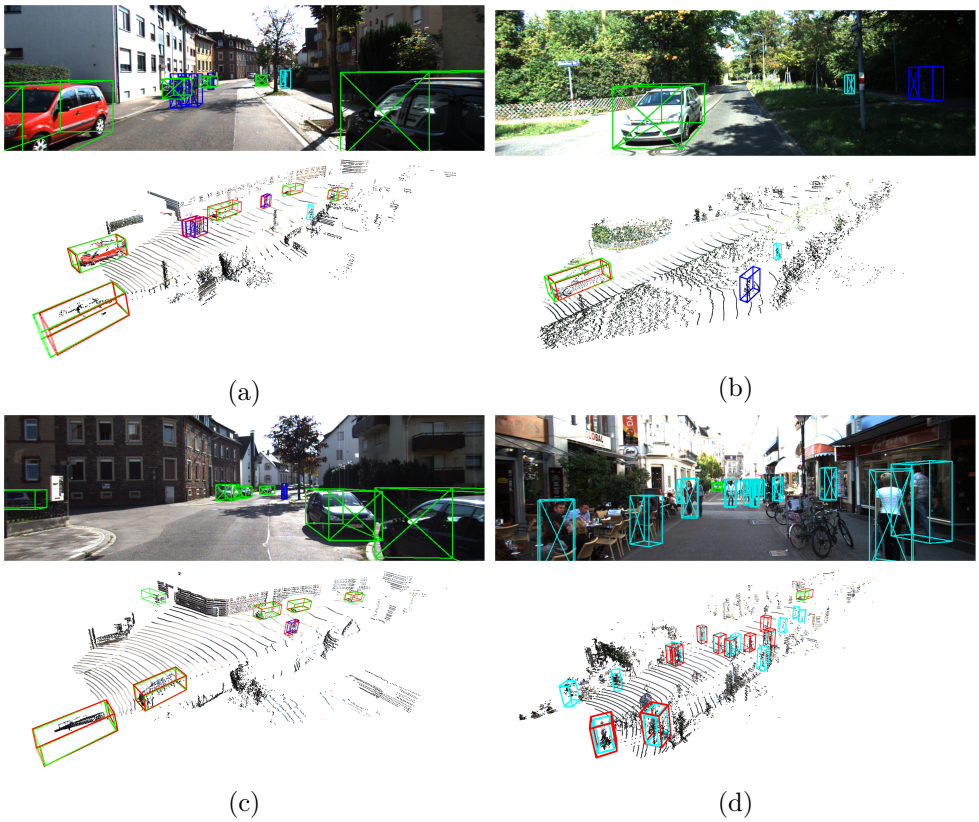


Figure 4.11: Additional results on the validation set. Examples where the model correctly detects objects that are clearly visible but not annotated, resulting in misleading performance degradation.

- the partially truncated car on the left in Fig. 4.11c;

- many pedestrians in the back occluded in the image by the pedestrians in the front but visible in the point cloud in Fig. 4.11d. Also, sitting people are labelled as a different class compared to standing pedestrians.

All these labelling inaccuracies result in misleading performance degradation, as correct detections are erroneously interpreted as false positives by the metric. This is particularly problematic for the pedestrian and cyclist classes, as they appear to be more affected by missing labels while being less numerous, which renders the evaluation on these classes less reliable for gauging the actual performance of the system. On the other hand, however, it also means that the performance obtained by the proposed system is closer to the obtainable limit on the KITTI dataset than it seems (at least in case of the car class), as perfect Average Precision cannot be attained due to labelling mistakes.

4.6 Discussion

In this chapter I proposed to modify the point-based 3D object detector Votenet to make it suitable for the noisier and sparser point clouds returned by LiDAR sensors, which are commonly used in autonomous driving contexts. More specifically, I altered the original technique used for point sampling, as it caused considerable information loss and led to numerous false negatives, and I adjusted the classification logic such that it better correlates with effective box quality. I also redesigned the Set Abstraction layers, substituting the traditional MLP used for feature extraction and the max pooling used for feature aggregation with attention-based mechanisms.

The overall system displayed convincing performance on the autonomous driving KITTI dataset. Attention, in particular, showed promising results, contributing to a non-trivial improvement to the overall system performance while requiring minor extra computational resources.

The limited nature of the KITTI dataset, however, allows to perform only an initial and superficial evaluation of the proposed system. In order to properly asses the importance of attention and measure its true potential and usefulness, further studies on large emergent datasets are warranted.

Conclusions

In this thesis, I researched deep learning-based object detection, which I applied to solve three distinct problems in autonomous driving contexts.

In chapter 2 I proposed a deep learning pipeline to perform parking slot detection from surround-view images. To achieve this, I modified the existing 2D image-based detector Faster R-CNN to allow for the prediction of generic quadrilaterals instead of axis-aligned bounding boxes. To optimize the model, I collected and annotated a small dataset comprised of diverse road scenes and parking lots, containing varied parking slot types and recorded under different observation conditions. The resulting system showed satisfactory results, as it was capable to withstand significant noise and accurately estimate the location of unobserved parking slots despite the extremely limited training data it was trained on, while being able to process over 13 surround-view scenes per second. It also displayed remarkable capabilities to adapt to an entirely different domain, as proven by its ability to perform acceptably well when trained and tested on the original spherical images used to generate the surround-view, which advocates for the robustness and effectiveness of the method. Further studies on model sparsification showed that comparable performance can be obtained by using just around 7% of the total trainable parameters, which leaves wide margins for execution time improvements.

In chapter 3 I extended Faster R-CNN to perform the task of monocular 3D car detection in driving contexts. This was accomplished by introducing an extra module that is responsible for estimating the 3D bounding box correspond-

ing to each image-level detection returned by the original model. To train this module, I proposed a novel loss function based off the Generalized Intersection-over-Union, a variant of the Intersection-over-Union originally introduced to improve the performance of 2D detectors that guarantees a gradient value even in case of disjoint boxes. To adapt it to the more challenging 3D case, I opted for disentangling the estimation of orientation from that of position and dimensions, computing the GIoU on canonical, equioriented boxes. Moreover, to speed-up optimization and avoid an anomalous behavior of the proposed formulation, I further split the optimization of position and dimensions into six separate loss components, each responsible for one degree of freedom. Tests conducted on the autonomous driving KITTI dataset show that the proposed approach is an effective solution to the problem of monocular 3D detection, as it surpassed many existing and more complex state-of-the-art systems while retaining real-time speed on existing hardware. The module responsible for 3D detection, moreover, is very simple and could thus be integrated into other 2D detection pipelines quite straightforwardly. Several studies conducted on the proposed system, as well as on an entirely different LiDAR-based 3D detector, showed that the proposed GIoU-based formulation is effective, and could therefore be utilized to potentially boost the performance of other existing 3D pipelines.

Finally, in chapter 4 I investigated 3D object detection on a different type of input data: LiDAR point clouds. More specifically, I focused on the point-based method Votenet, as it is conceptually simple and showed promising results on indoor scenes collected from RGB-D sensors. First, I introduced some modifications to the cluster sampling strategy and to the classifier of the original pipeline in order to improve performance on the noisier LiDAR scans utilized in autonomous driving. Then, I studied the effectiveness of attention-based mechanisms inside the Set Abstraction layers used for point feature extraction and aggregation, as they should allow to better model inter-point relationships within each point group, resulting in stronger feature representations. The overall method displayed convincing performance on the KITTI dataset,

performing on par and above existing state-of-the-art LiDAR-based pipelines. The proposed attention formulations, in particular, led to promising improvements in the results while requiring minor additional computational resources compared to the vanilla model, showing their potential effectiveness as a strong alternative symmetric function for point cloud feature extraction. The rather limited scope of the KITTI dataset, however, restricts the extent in which the efficacy of the attention mechanism can be evaluated: further experimentation of different model formulations on larger and more heterogeneous datasets is required in order to properly determine its importance and assess its benefits.

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR. arXiv*, 12 2014.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11 2013. doi:10.1109/CVPR.2014.81.

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Con-*

ference on Computer Vision and Pattern Recognition (CVPR), pages 779–788, 06 2016. doi:10.1109/CVPR.2016.91.

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander Berg. Ssd: Single shot multibox detector. volume 9905, pages 21–37, 10 2016. doi:10.1007/978-3-319-46448-0_2.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. volume 9351, pages 234–241, 10 2015. doi:10.1007/978-3-319-24574-4_28.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 06 2015. doi:10.1109/TPAMI.2016.2577031.

[10] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. 12 2016.

[11] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. 12 2016.

[12] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation. 12 2017.

[13] Alex Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. pages 12689–12697, 06 2019. doi:10.1109/CVPR.2019.01298.

[14] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. pages 770–779, 06 2019. doi:10.1109/CVPR.2019.00086.

[15] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. pages 2147–2156, 06 2016. doi:10.1109/CVPR.2016.236.

[16] Jason Ku, Alex Pon, and Steven Waslander. Monocular 3d object detection leveraging accurate proposals and shape reconstruction. pages 11859–11868, 06 2019. doi:10.1109/CVPR.2019.01214.

[17] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. Stereo r-cnn based 3d object detection for autonomous driving. pages 7636–7644, 06 2019. doi:10.1109/CVPR.2019.00783.

[18] Zhishuai Zhang, Jiyang Gao, Junhua Mao, Yukai Liu, Dragomir Anguelov, and Congcong Li. Stinet: Spatio-temporal-interactive network for pedestrian detection and trajectory prediction. 05 2020.

[19] A. Zinelli, L. Musto, and F. Pizzati. A deep-learning approach for parking slot detection on surround-view images. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 683–688, 2019.

[20] Seyed Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian D. Reid, and Silvio Savarese. Generalized intersection over union: A metric and A loss for bounding box regression. *CoRR*, abs/1902.09630, 2019. URL: http://arxiv.org/abs/1902.09630, arXiv:1902.09630.

[21] Andrea Zinelli and Luigi Musto. Monocular 3d object detection using disjoint generalized intersection-over-union loss. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8, 09 2020. doi:10.1109/ITSC45102.2020.9294398.

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

[23] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas Guibas. Frustum pointnets for 3d object detection from rgb-d data. pages 918–927, 06 2018. doi:10.1109/CVPR.2018.00102.

[24] R. Charles, Hao Su, Mo Kaichun, and Leonidas Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. pages 77–85, 07 2017. doi:10.1109/CVPR.2017.16.

[25] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. 06 2017.

[26] Charles Qi, Or Litany, Kaiming He, and Leonidas Guibas. Deep hough voting for 3d object detection in point clouds. pages 9276–9285, 10 2019. doi:10.1109/ICCV.2019.00937.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[28] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[29] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[30] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[31] Y. Bengio and Yann Lecun. Scaling learning algorithms towards ai. 01 2007.

[32] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15, 01 2010.

[33] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network, November 2015. URL: http://arxiv.org/abs/1505.00853, arXiv:1505.00853.

[34] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 971–980. Curran Associates, Inc., 2017. URL: http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf.

[35] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. URL: http://dx.doi.org/10.1038/323533a0.

[36] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999. URL: http://www.sciencedirect.com/science/article/pii/S0893608098001166, doi:https://doi.org/10.1016/S0893-6080(98)00116-6.

[37] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[38] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. URL: http://arxiv.org/abs/1412.6980.

[39] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679 – 698, 12 1986. doi:10.1109/TPAMI.1986.4767851.

[40] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. 06 2015.

[41] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120, 12 2015. doi:10.1007/s11263-016-0911-8.

[42] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[43] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[44] Yann Lecun, Patrick Haffner, and Y. Bengio. Object recognition with gradient-based learning. 08 2000.

[45] Shane Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[47] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of*

Computer Vision (IJCV), 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y.

[48] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL: http://arxiv.org/abs/1409.4842.

[49] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 02 2015.

[50] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Weinberger. Densely connected convolutional networks. 07 2017. doi:10.1109/CVPR.2017.243.

[51] Andrew Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 04 2017.

[52] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. 03 2016.

[53] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. pages 3213–3223, 06 2016. doi:10.1109/CVPR.2016.350.

[54] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[55] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Zitnick. Microsoft coco: Common objects in context. volume 8693, 04 2014. doi:10.1007/978-3-319-10602-1_48.

[56] Evan Shelhamer, Jonathon Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1–1, 05 2016. doi:10.1109/TPAMI.2016.2572683.

[57] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP, 06 2016. doi:10.1109/TPAMI.2017.2699184.

[58] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. 06 2017.

[59] Jasper Uijlings, K. Sande, T. Gevers, and Arnold Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 09 2013. doi:10.1007/s11263-013-0620-5.

[60] Ross Girshick. Fast r-cnn. 04 2015. doi:10.1109/ICCV.2015.169.

[61] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. 04 2018.

[62] Abhinav Shrivastava, Harikrishna Mulam, and Ross Girshick. Training region-based object detectors with online hard example mining. pages 761–769, 06 2016. doi:10.1109/CVPR.2016.89.

[63] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern*

Analysis and Machine Intelligence, PP:1–1, 07 2018. doi:10.1109/TPAMI.2018.2858826.

[64] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. 03 2017.

[65] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixá. Tracking without bells and whistles. pages 941–951, 10 2019. doi:10.1109/ICCV.2019.00103.

[66] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. 06 2016. doi:10.15607/RSS.2016.XII.042.

[67] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. pages 6526–6534, 07 2017. doi:10.1109/CVPR.2017.691.

[68] Benjamin Graham and Laurens Maaten. Submanifold sparse convolutional networks. 06 2017.

[69] Benjamin Graham, Martin Engelcke, and Laurens Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. 11 2017.

[70] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. pages 4490–4499, 06 2018. doi:10.1109/CVPR.2018.00472.

[71] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18:3337, 10 2018. doi:10.3390/s18103337.

[72] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. pages 1951–1960, 10 2019. doi:10.1109/ICCV.2019.00204.

[73] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3dssd: Point-based 3d single stage object detector. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[74] Chenhang He, Hui Zeng, Jianqiang Huang, Xian-Sheng Hua, and Lei Zhang. Structure aware single-stage 3d object detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[75] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[76] Zengyi Qin, Jinglu Wang, and Yan Lu. Triangulation learning network: From monocular to stereo 3d object detection. pages 7607–7615, 06 2019. doi:10.1109/CVPR.2019.00780.

[77] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. pages 8437–8445, 06 2019. doi:10.1109/CVPR.2019.00864.

[78] Thomas Roddick, Alex Kendall, and Roberto Cipolla. Orthographic feature transform for monocular 3d object detection. 2019.

[79] Zengyi Qin, Jinglu Wang, and Yan Lu. Monogrnet: A geometric reasoning network for monocular 3d object localization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:8851–8858, 07 2019. doi:10.1609/aaai.v33i01.33018851.

[80] Bin Xu and Zhenzhong Chen. Multi-level fusion based 3d object detection from monocular images. pages 2345–2353, 06 2018. doi:10.1109/CVPR.2018.00249.

[81] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Celine Teuliere, and Thierry Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. pages 1827–1836, 07 2017. doi:10.1109/CVPR.2017.198.

[82] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81, 02 2009. doi:10.1007/s11263-008-0152-6.

[83] Nicholas True. Vacant parking space detection in static images. 2007.

[84] Marc Tschentscher and Marcel Neuhausen. Video-based parking-space detection. pages 159–166, 01 2012.

[85] Keiichi Yamada and Morimichi Mizuno. A vehicle parking detection method using image segmentation. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 84(10):25–34, 2001. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ecjc.1039, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/ecjc.1039, doi:10.1002/ecjc.1039.

[86] Jin Xu, Guang Chen, and Ming Xie. Vision-guided automatic parking for smart car. pages 725 – 730, 02 2000. doi:10.1109/IVS.2000.898435.

[87] Ho Jung, Dong Kim, Pal Yoon, and Jaihie Kim. Structure analysis based parking slot marking recognition for semi-automatic parking system. pages 384–393, 08 2006. doi:10.1007/11815921_42.

[88] Yusnita Rahayu, Norbaya Fariza, and Basharuddin Norazwinawati. Intelligent parking space detection system based on image processing. 2012.

[89] Linshen Li, Lin Zhang, Xiyuan Li, Xiao Liu, Ying Shen, and Lu Xiong. Vision-based parking-slot detection: A benchmark and a learning-based

approach. pages 649–654, 07 2017. doi:10.1109/ICME.2017.8019419.

[90] Yoav Freund and Robert E. Schapire. A short introduction to boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann, 1999.

[91] Kazukuni Hamada, Zhencheng Hu, Mengyang Fan, and Hui Chen. Surround view based parking lot detection and tracking. pages 1106–1111, 06 2015. doi:10.1109/IVS.2015.7225832.

[92] Yuliang Liu and Lianwen Jin. Deep matching prior network: Toward tighter multi-oriented text detection. pages 3454–3461, 07 2017. doi:10.1109/CVPR.2017.368.

[93] Zhuoyao Zhong, Lei Sun, and Qiang Huo. An anchor-free region proposal network for faster r-cnn based text detection approaches. *International Journal on Document Analysis and Recognition (IJDAR)*, 04 2018. doi:10.1007/s10032-019-00335-y.

[94] Petr Hurtik, Vojtech Molek, Jan Hula, Marek Vajgl, Pavel Vlasanek, and Tomas Nejezchleba. Poly-yolo: higher speed, more precise detection and instance segmentation for yolov3. *arXiv preprint arXiv:2005.13243*, 2020.

[95] Mark Everingham, Luc Van Gool, Christopher Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 06 2010. doi:10.1007/s11263-009-0275-4.

[96] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL: http://arxiv.org/abs/1902.09574, arXiv:1902.09574.

[97] Heiko Hirschmüller. Stereo processing by semi-global matching and mutual information. *in IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:328–341, 02 2008.

[98] Koichiro Yamaguchi, David Mcallester, and Raquel Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. pages 756–771, 09 2014. doi:10.1007/978-3-319-10602-1_49.

[99] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. pages 5410–5418, 06 2018. doi:10.1109/CVPR.2018.00567.

[100] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals using stereo imagery for accurate object class detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP, 08 2016. doi:10.1109/TPAMI.2017.2706685.

[101] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Košecká. 3d bounding box estimation using deep learning and geometry. pages 5632–5640, 07 2017. doi:10.1109/CVPR.2017.597.

[102] Andrea Simonelli, Samuel Rota Buló, Lorenzo Porzi, Manuel Lopez Antequera, and Peter Kontschieder. Disentangling monocular 3d object detection. pages 1991–1999, 10 2019. doi:10.1109/ICCV.2019.00208.

[103] Shuran Song, Samuel Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. pages 567–576, 06 2015. doi:10.1109/CVPR.2015.7298655.

[104] Angela Dai, Angel Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias NieBner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. pages 2432–2443, 07 2017. doi:10.1109/CVPR.2017.261.

[105] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. pages 5560–5568, 06 2019. doi:10.1109/CVPR.2019.00571.

[106] Jimmy Ba, Jamie Kiros, and Geoffrey Hinton. Layer normalization. 07 2016.

[107] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL: http://proceedings.mlr.press/v97/lee19d.html.