A Deep Neural Network for Object Detection and Tracking with 3D LiDAR

Leonardo Cabral Ferreira Cardoso

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Dr. Pedro Daniel dos Santos Miraldo

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Dr. Pedro Daniel dos Santos Miraldo
Member of the Committee: Prof. Jacinto Carlos Marques Peixoto do Nascimento

September, 2021

Declaration:

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the *Universidade de Lisboa*.

Declaração:

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Instituto Superior Técnico

A Deep Neural Network for Object Detection and Tracking with 3D LiDAR

Leonardo Cabral Ferreira Cardoso

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervision team:
Dr. Pedro Daniel dos Santos Miraldo
Msc. Gonçalo José Dias Pais
Msc. André Gonçalves Mateus

September, 2021

Abstract

Autonomous vehicles scan their environment with a range of sensors (*e.g.* camera, LiDAR) to take the safest action on the road. Therefore, locating and predicting the motion of other road agents has motivated plenty of research on the computer vision tasks of object detection and object tracking. This thesis takes an existing deep learning pipeline for 3D object detection [30] and modifies it to make location and tracking predictions from 3D semantic point clouds only. These point clouds are obtained in a pre-processing step which exploits the fusion of dense 2D semantic segmentation results with 3D point clouds that naturally offer depth information. The tracking method from [40] is embedded within the network to predict objects' displacements between two consecutive frames, thus, predicting their velocity. An extra input channel for a heatmap containing the objects' location in the previous frame was also tested. Several ablation studies were conducted to test the model's performance using different types of heatmaps, and not using heatmaps in any way. Results showed that a heatmap absent model yielded overall better results, because our model could not predict the heatmaps correctly. Our deep learning approach allows end-to-end learning for detection and tracking, and runs 38% faster (18 FPS) than the baseline model (13 FPS).

Keywords: LiDAR, detection, tracking, deep learning, computer vision, displacement, heatmap, ablation study.

Resumo

Os veículos autónomos analisam o seu ambiente com diversos sensores (p.e. câmeras, *LiDAR*) de forma a tomar a ação mais segura na estrada. Consequentemente, localizar e prever a trajetória de outros agentes rodoviários motivou a investigação de métodos de deteção e *tracking* de objetos, no domínio de Visão por Computador. Esta tese modifica uma rede *deep learning* usada para a deteção de objetos 3D de forma a fazer simultaneamente deteção e *tracking* de objectos a partir de *point clouds* semânticas 3D apenas. Estas point clouds são obtidas num pré-processamento que tira vantagem da fusão de resultados densos de segmentação semântica em 2D com *point clouds* 3D que oferecem naturalmente informação de profundidade. O método de *tracking* é incluído na rede de forma a que sejam feitas predições de vetores de deslocamento de objectos entre duas frames consecutivas, podendo assim prever a sua velocidade. Adicionalmente, foi testado um canal extra de *input* para um *heatmap* que contém a posição dos objetos na frame anterior. Foram realizados diversos estudos de ablação para testar a *performance* do modelo proposto ao usar diferentes estilos de *heatmaps*, e ao não usar o *input* extra de *heatmap* de todo. Os resultados demonstram que um modelo sem *heatmaps* oferece os melhores resultados, porque o modelo não consegue estimar os *heatmaps* corretamente. A nossa rede *deep learning* consiste num único modelo *end-to-end* para a deteção e tracking, e corre 38% mais rápido (18 FPS) que o modelo basal (13 FPS).

Palavras-chave: *LiDAR*, deteção, *tracking*, deep *learning*, visão por computador, deslocamento, *heatmap*, estudo de ablação.

Acknowledgements

I want to thank my supervisor Dr. Pedro Miraldo and his two Ph.D. students, Gonçalo Pais and André Mateus, who have been of invaluable help during this sixteen-month project. The weekly, sometimes biweekly, meetings and the prompt answers to my questions have truly propelled my work.

To my colleagues who turned into friends throughout this five-year journey. The restless nights spent finishing projects or studying for exams were avenged with restless nights of celebration, which allowed us to truly enjoy the academic lifestyle. To my college dorm friends for all the hilarious and sharing moments we had in those countless evenings.

To my family who supported me throughout these five long years away from home and to my outstanding girlfriend who celebrated my small victories with me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this work, a deep learning architecture is proposed to perform 3D object detection and tracking, while taking advantage of the fusion of road environment information collected by different sensors, namely, camera and LiDAR (Light Detection And Ranging) sensors. The dense representation captured by the camera and the depth information captured by the LiDAR complement each other, thus the network takes advantage of a rich environment representation. The idea is based on a combination of the object detection pipeline from Complexer-YOLO [29] and the online tracking method present in CenterTrack [40].

Initially, 2D pictures obtained by a camera are fed into an off-the-shelf, pre-trained model [11] that performs semantic segmentation. The model outputs RGB images where each pixel color represents a class label. Then, the 2D semantic segmentation results are projected onto the 3D point cloud obtained by a LiDAR sensor. This is achieved by a mapping between pixels in the image and points in the point cloud, given by the camera projection matrix. Then, the semantic labels are copied from the 2D domain to the 3D domain. The result of this first module is a 3D semantic point cloud where each point, within the camera FoV (Field of View), has a class label associated with it. The 3D semantic point clouds are obtained in this pre-processing step, i.e., they are obtained in an offline manner. The object detection module consists of a single-shot object detector, Complex-YOLO [30], which was altered to accept as input the 3D semantic point clouds. With the object detection task achieved, the tracking method present in CenterTrack [40] is embedded within the network. Thus, accomplishing the proposed deep learning architecture that takes a sequence of road data scenes as input and outputs oriented 3D

objects' locations, velocities, and tracking IDs for the respective bounding boxes. The tracking method is online, i.e., the outputs are a consequence of the current and previous frames only, there's no peeking into the future.

1.1 Motivation

In recent years, autonomous driving has gathered increasing attention from the industry. Modern vehicles already perform some automated tasks such as adaptive cruise control and assisted parking. Besides improving comfort, vehicle automation can also improve safety on the road. A good perception of the fast-changing surroundings is required to take the safest actions within the road environment. Consequently, today's vehicles pushing the envelope of automated driving are equipped with a wide range of sensors, such as cameras and 3D LiDAR sensors, to scan their surroundings.

My thesis focuses on developing a deep learning 3D object detection and tracking pipeline for the road environment. In a pre-processing step, the method exploits the fusion of semantic information from RGB images with LiDAR point clouds that naturally provide object location to achieve a 3D semantic point cloud. The goal is to make detection and tracking predictions based on this 3D input only.

1.2 Contributions and Outline of the Thesis

The contributions of this thesis are three-fold:

- Modify Complex-YOLO pipeline to accept 3D semantic input;
- Embed tracking prediction on the Complex-YOLO architecture;
- Conduct an ablation study for the proposed 3D Object Detection and Tracking module (see appendix Fig. A.1).

In Chapter 2, a literature review is conducted on the tasks of semantic segmentation, object detection, and object tracking applied to the 3D point cloud. Then, in Chapter 3, the methods on which this thesis work is based are described in further detail. In Chapter 4, the proposed pipeline is described in a sequential approach. Initially, the pipeline input data from the KITTI dataset is presented. Then, the pipeline's three major modules are

described: the semantic segmentation module (Sec. 4.2), the 3D Object Detection and Tracking module (Sec. 4.3), and the 3D Object Detection and Tracking module (Sec. 4.4). With the pipeline detailed, Chapter 5 presents the experimental results obtained in the testing stage of the proposed pipeline. Lastly, in Chapter 6, the conclusion discusses the achieved work in retrospect.

Chapter 2

Related Work

In this chapter, a literature review is performed on existing methods that explore the 3D LiDAR point cloud to achieve semantic segmentation, object detection, and object tracking, in the three-dimensional space. The point cloud data format is introduced as well as the techniques developed to exploit its advantages for the mentioned computer vision tasks.

2.1 The Point Cloud

As the name indicates, a point cloud is a set of data points in space, represented in a three dimensional coordinate system, as shown in Fig. 2.1. They are usually obtained by 3D scanners, such as 3D LiDAR, or an RGB-D sensor. While LiDAR only provides depth information, RGB-D sensors get the color of the scene, but has a lower range as the LiDAR one. On the other hand, LiDAR point clouds are much sparser than RGB-D ones.

In contrast to 2D images, which are grid-like structures, point clouds are unstructured, i.e., they are unordered and (generally) a very sparse set of points. These characteristics prove to be the main challenges of handling point clouds. However, rich underlying 3D information resides in this data format, such as the inherent depth information and location of objects in space. Therefore, recently, several methods [23, 24, 10, 16, 34, 12, 28, 37, 13, 29] have partially but successfully exploited some of these characteristics and improved significantly the current State-of-the-Art (SOTA) in 3D semantic segmentation, object detection, and object tracking. However, applying

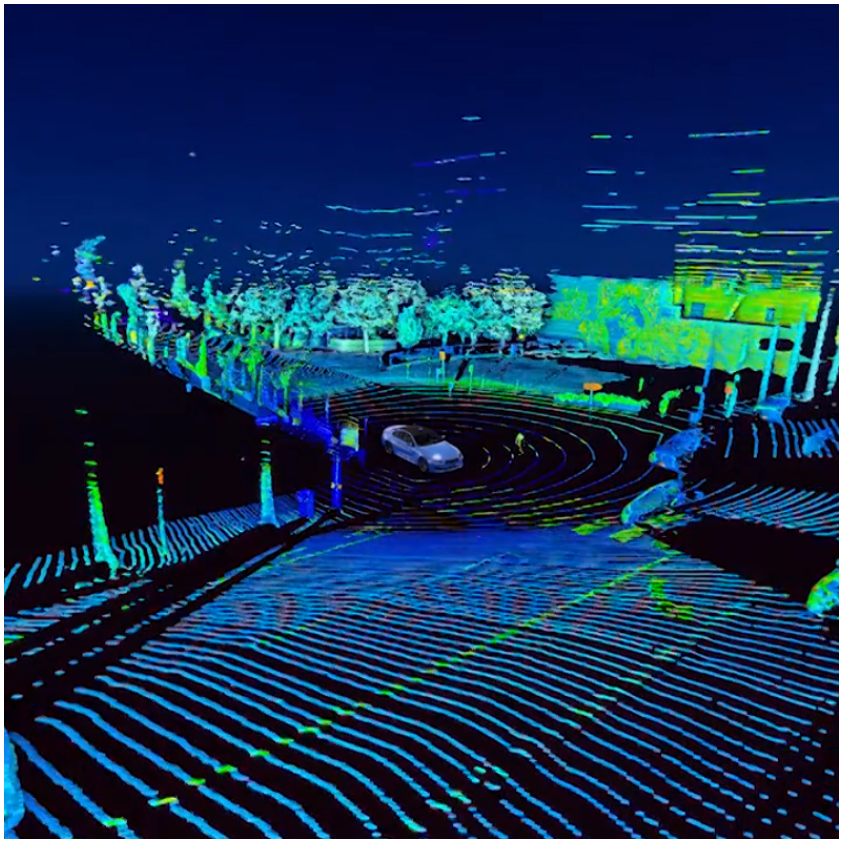


Figure 2.1: Example of a LIDAR point cloud mounted on a vehicle and scanning a road scenario. Figure obtained from [35].

deep learning techniques to point clouds is still an active problem in the computer vision community with many open problems that motivate plenty of research.

2.2 3D Semantic Segmentation

Segmentation in point clouds consists of the assignment of a class label to each 3D point. For a better visualization of semantic segmentation on point clouds, different classes are associated with different point colors, as shown in Fig. 2.2. Both global geometric structure and small details of each point cloud are essential to successfully segment the data. The goal of semantic segmentation is the division of the input point cloud into subsets of points that share the same semantic meaning. When it comes to dealing with point clouds as input, two main paradigms exist, projection-based methods

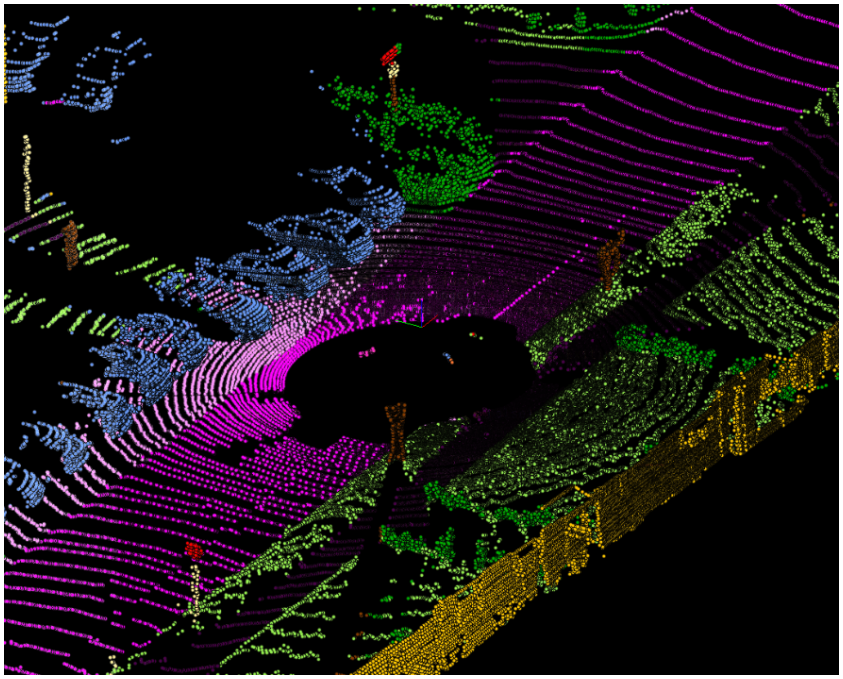


Figure 2.2: Semantic segmentation results on a 3D point cloud. The colors represent each different semantic score. Here, the color pink means the points belong to the road, blue means the point represent a car, yellow is for the buildings' points and green is for any type vegetation. Figure obtained from [20].

and point-based methods, reviewed in Secs. 2.2.1 and 2.2.2, respectively.

2.2.1 Projection-based Methods

Due to its unstructured format, various approaches have been proposed to handle point clouds. These consist on the projection of the input point cloud onto multiple-views representation, spherical representation, or volumetric representation.

Multi-view representation

A semantic segmentation pipeline for 3D point clouds projected on multi-view representation was proposed in [15]. First, it renders 2D images from the input point cloud. Then it feeds the synthetically obtained images to a Convolutional Neural Network (CNN). The network performs 2D semantic segmentation using a well established and SOTA method, namely [19]. The pipeline is agnostic to which method is used for

2D segmentation. It retrieves three rendered images (RGB, depth, normals) from the original point cloud. The 2D segmentation is performed and the three scores are added, making the final segmentation prediction of the 3D point cloud.

Instead of obtaining 2D renders of arbitrary views of the point cloud, [32] propose a virtual orthogonal camera for each point belonging to the point cloud. The view is obtained by generating a fixed size tangent plane to every point. Neighboring points are projected onto the plane, obtaining a discrete signal through signal interpolation to a 2D plane. Each of these images is then fed to a CNN that performs typical planar convolution. Results showed great scalability for large point clouds with millions of points.

Spherical representation

RangeNet++ [20] propose a fast, real-time, method for semantic segmentation of 3D point clouds. It takes the raw point cloud as input; projects it onto a spherical representation, i.e, a 2D grid representation; and applies a 2D CNN as the backbone structure that performs the semantic segmentation, thus obtaining semantic labels for each pixel of the image. Afterwards, the point cloud is reconstructed and a post-processing step is proposed. This step consists of a GPU-enabled k-Nearest Neighbors (kNN) based algorithm, which tackles the discretization errors and blurry outputs from the 2D CNN.

Volumetric representaiton

The SEGCloud [33] framework applies a 3D Fully Convolutional Neural Network (FCNN) on the voxelized 3D point cloud. This leads to coarse segmentation since all points inside a voxel are assigned the same label, making the accuracy results dependent on voxel size. To refine the segmentation results, a post-processing step using interpolation and a Conditional Random Field (CRF) is used. The interpolation step re-projects the coarse voxel predictions back to the point cloud to reduce information loss caused by coarse voxelization. The CRF is used to fine-graine segmentation results and it is modeled by an RNN that is trained alongside the FCNN.

In [7], the authors focus on the inherently sparse nature of 3D point clouds and how inefficient, concerning the memory and computation costs, the standard dense implementations of CNNs are when applied directly on point clouds. Prior sparse implementations

of CNN dilated the sparse data at each layer by applying full convolutions, they propose a simple solution that restricts the output of the convolution only to non-empty voxels. This novel operator designated submanifold sparse convolution is the building block of the proposed network called Submanifold Sparse Convolutional Network (SSCN). Results showed that this method is suitable for efficient processing of high dimensional and spatially-sparse data, including point clouds.

2.2.2 Point-based Methods

In order to take full advantage of the 3D point cloud data format and fully exploit its intrinsic geometric and structural information, several methods work on learning from point clouds directly without any transformation. Existing point-based methods are split into three categories, point-wise MLP, point convolution, and graph-based.

Point-wise MLP

These type of methods usually implement shared MLP as the basic building block of their networks due to its efficiency. This simplification comes at a cost, point features extracted by shared MLP do not capture the local geometry nor the relationship among neighbouring points, which is essential for the segmentation task. Therefore, these methods employ different strategies in their networks in order to gather wider contextual information for each point, thus learning richer local structures.

In [23], Qi proposes the PointNet, a pioneering work in learning directly from raw point clouds. It is a deep learning architecture that takes point clouds directly as input to avoid the increase of data volume and the introduction of quantization artifacts, when transformed into 3D voxel grids. PointNet learns point-wise features with stacked MLP layers, and extracts a global descriptor with a max-pooling layer. PointNet comprises two networks: a classification network which extracts global features from the input and a segmentation network that concatenates the global features with the point features, in order to predict the local geometry and global semantics. The output consists of class labels and the semantic labels for each point. Geometric transformations were considered in order for the semantic labeling to work independently of such transformations. The PointNet architecture is shown in Fig. 2.3.

Motivated by the success and advantages of methods that directly operate on raw

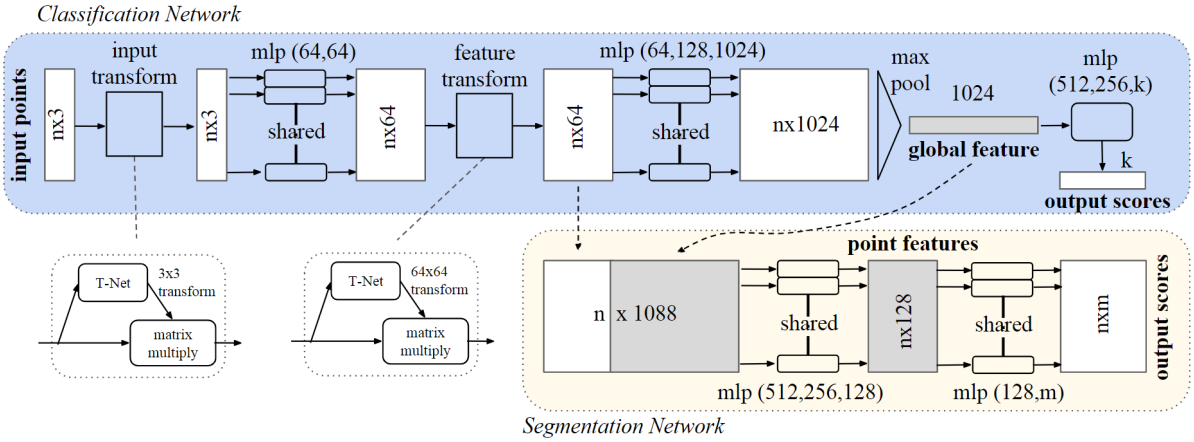


Figure 2.3: PointNet architecture. The classification network takes n input points from the point cloud, then input and feature transformations are applied and point-wise features are aggregated by the max pooling operation. The output for this network consists of classification scores for k classes. The segmentation network starts by concatenating global and local features and outputs point-wise labels. m denotes the size of the learned point-wise features. [23].

point clouds, the work in [3] aims to improve upon the way the point neighborhood region is addressed to aggregate features. In contrast to PointNet [23] that performed aggregation globally, the authors in [3] propose to define the point neighborhoods in a dynamic manner that is adaptive and sensitive to local geometry by applying k-means clustering on the point cloud features in the input space, and then define dynamic neighborhoods in the learned feature space by applying kNN. Dedicated loss functions such as pairwise distance loss and centroid loss are applied to further structure the learned point feature space.

Point Convolution

The success of CNNs relies on the convolution operator that detects the existent high correlation in local groups of values, represented in a regular data format such as an array. Given the nature of point clouds, an unstructured and unordered data format, the convolution operation cannot be easily transferred and applied to them. To avoid information loss, associated with projections made to the point cloud, several methods have been proposed to effectively apply convolution operations directly on point clouds.

Motivated by the universal property of the spatially-local correlation, which is inde-

pendent of data representation, and the success of the convolution operator in exploiting said correlation in regular data representations, authors at [16] address the issue of applying convolution directly on the unordered and irregular data format of point clouds and propose the PointCNN. Direct convolution application leads to arbitrary ordering and loss of shape information. The authors proposed a learnable X-transformation matrix dependent on input point cloud size, using an MLP. The goal is to weight and permute input features at the same time. Then, apply convolution on the transformed features. This makes the X-Conv block, the basic building block of the developed PointCNN. Results show that the learned X-transformations are not ideal. However, their use is far better than the direct application of convolution and it is competitive with the SOTA.

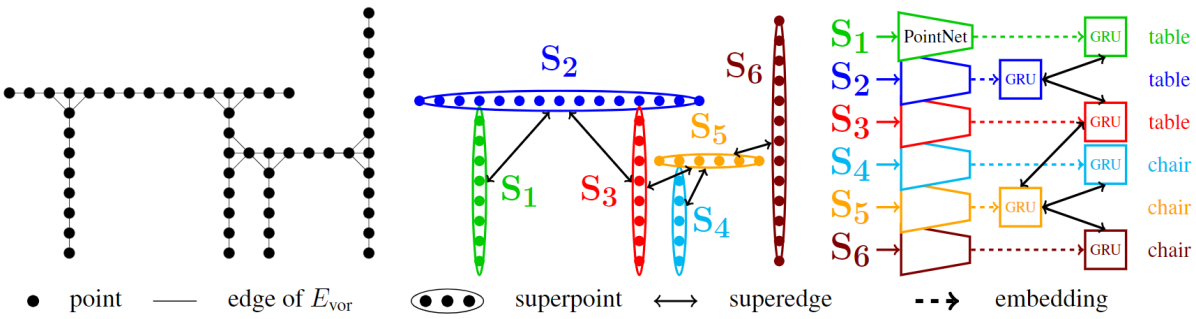
Graph-based

Some methods have used graph representations of point clouds and experimented with graph networks to capture intrinsic geometric shapes and structures embedded in 3D point clouds.

A novel graph-based structure SuperPoint Graph (SPG) is proposed at [12], for an efficient representation of large-scale point clouds with millions of points, as shown in Fig. 2.4(b). The first step of the proposed method takes the whole input point cloud and efficiently partitions it into simple geometrical shapes, called superpoints, in an unsupervised manner. From the superpoints (primitive geometric shapes), the attributed directed SPG is built. Each node of the SPG corresponds to a small and semantically homogeneous part of the point cloud. These smaller point clouds, the superpoints, are fed into a PointNet [23], that learns the spatial distribution of each superpoint. The final step consists of contextual segmentation where each superpoint is classified based on its PointNet embedding results and its local surroundings within the SPG. This is done by applying a graph CNN that naturally exploits the contextual relationship between object parts, given the proposed SPG representation. Experimental results show that the initial unsupervised partition step of the point cloud could be improved.

2.3 3D Object Detection

The 3D object detection task consists of accurately detecting the presence and location of all objects of interest in a given scene. The object detection task predicts a



(a) Input point cloud (b) Superpoint graph (c) Network architecture

Figure 2.4: Superpoint graph representation of a toy point cloud (a), represented by a superpoint graph (b). Each superpoint is fed into a PointNet whose output is refined, by considering SPG's edges (superedges) connections, to produce the final labelling (c). Figure obtained from [12].

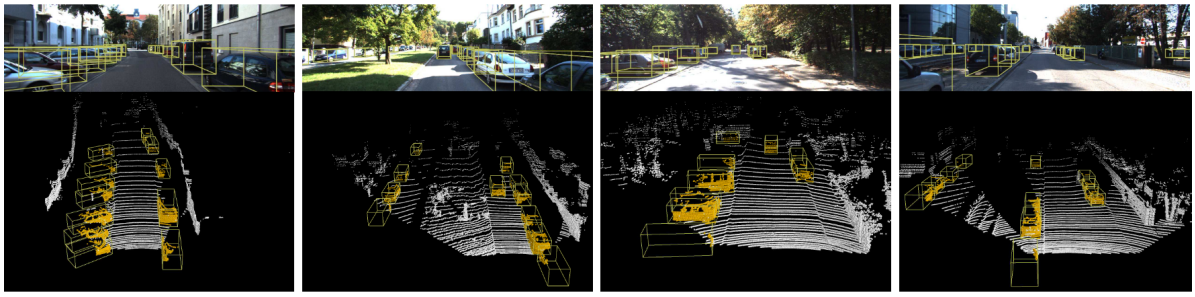


Figure 2.5: Visualization of 3D bounding boxes projected on RGB images and LIDAR point clouds, in the upper and lower row, respectively. Figure obtained from [39].

class label and estimates a 3D oriented bounding box for each object. Examples of 3D bounding box predictions are shown in Fig. 2.5. Two main paradigms exist for 3D object detection methods, region proposal-based methods and single shot methods. They are addressed in Secs. 2.3.1 and 2.3.2, respectively.

2.3.1 Region Proposal-based Methods

These methods initially propose several possible regions in space that contain objects of interest, these regions are called proposals. With the 3D search space reduced, region-wise features are extracted to obtain the class label of each proposal. Region proposals allow the generation of high quality detections at later detection stages. However, any missed object instances at the proposal generation stage can't be recovered afterward.

Therefore, achieving a high recall during the region proposal stage is critical for these methods. These methods can be further categorized into two categories according to their region proposal generation approach: multiple view and frustum-based methods.

Multi-view Methods

Multi-view methods fuse proposal-wise features obtained from multiple viewpoints, such as bird's-eye view (BEV), the LIDAR front view or even RGB images, to estimate the oriented 3D bounding boxes.

MV3D, presented in [2], is a framework designed for 3D object detection in a road scene. It takes advantage of multiple views obtained from LIDAR (BEV and front view) and from RGB images obtained from a camera. The input of the network consists of a multi-view representation of the 3D point cloud and a 2D image. The 3D object proposals are obtained from the BEV map and then projected onto the RGB image and the LIDAR front view. The reasoning for using the BEV to propose 3D bounding boxes lies in the fact that in the road scene all vehicles are on the ground level and occlusions are much limited from this perspective. With box candidates on these three different views, the region-wise features are fused to predict oriented 3D bounding boxes. The multi-view features are fused by a proposed region-based fusion network that deeply fuses the multi-view feature information to accurately predict oriented 3D bounding boxes.

Frustum-based Methods

Frustum-based methods start generating region proposals from existing 2D object detectors and then extract 3D frustum candidates for each 2D proposal region. Efficient proposals from 3D frustum point clouds is possible, however, these methods' performance is highly limited by 2D image detectors and, in the worst case scenario, an object can be entirely missed if it is not detected in 2D.

Qi et al., in [22], propose the Frustum PointNets, a pioneering work in taking raw point clouds as input to localize and classify objects in 3D space. The proposed framework for 3D object detection, which takes an image and a point cloud as input, consists of three main modules: frustum proposal, 3D instance segmentation, and 3D amodal bounding box estimation, as shown in Fig. 2.6. The first module starts by applying a

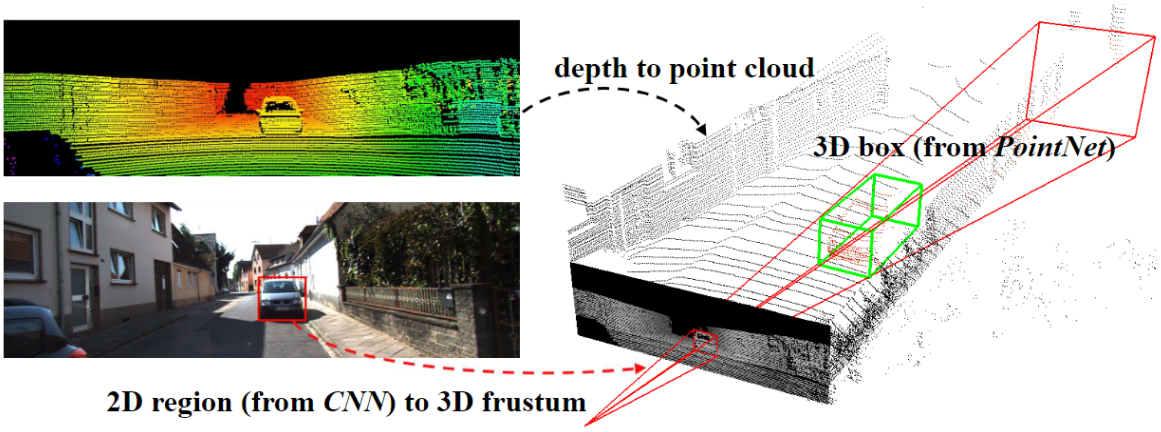


Figure 2.6: The 3D object detection method by Frustum PointNet starts by proposing a 2D region and then lifts it into a frustum region in the 3D space. Finally, PointNet is applied on the frustum to perform object classification and to estimate the amodal 3D bounding box. Figure obtained from [22].

well known 2D object detector network [17] (the whole framework is agnostic to which one is used) to obtain 2D bounding boxes from a viewpoint. Given the camera projection matrix, the 2D bounding box can be lifted to a frustum, whose near and far planes are specified by the depth information from the point cloud. This frustum defines the 3D search space for the object, thus the frustum proposal is obtained. The second module applies PointNet, [23], to learn features directly from the point cloud and to perform 3D instance segmentation. The last module estimates the object's amodal oriented 3D bounding box by applying a regression PointNet.

2.3.2 Single Shot Methods

Single shot methods do not perform the region proposal generation step and as a result, they tend to run faster and are easier to adapt to real-time applications. Since no region proposal stage is performed, these methods directly, i.e., in a single stage, predict the class probabilities of objects and regress their respective oriented 3D bounding boxes.

The PIXOR, [38], is a real-time 3D object detector that takes point clouds as input in the context of autonomous driving, it outputs the bounding boxes and respective orientations decoded from pixel-wise predictions of the 3D object estimate. To ease computation cost, the point cloud scene is represented from the BEV, this is feasible in

the context of autonomous driving since the objects are all on ground level and they don't overlap each other as it occurs in first-person view corresponding to the LIDAR front view angle. Being a single stage dense object detector and the non usage of anchors also lead to a simpler detection framework capable of meeting real-time requirements.

VoxelNet, in [41], is an end-to-end deep learning architecture for point cloud based 3D object detection which aims to learn features directly from a 3D voxel grid constructed from a LIDAR point cloud. No manual feature representations are used in order to effectively exploit the 3D information. Due to the sparsity of point clouds, voxels with less than a threshold number of points are not processed to reduce memory and computation costs. A novel Voxel Feature Encoding (VFE) layer combines point-wise features with a locally aggregated feature thus transforming a group of points inside each voxel into a unified feature representation. Stacking several VFE layers allows learning more complex features that characterize 3D shape information. The 3D convolution layers further aggregate local voxel features. The point cloud is now encoded in a tensor like data structure and it is fed into an RPN algorithm [27] that outputs the detection result. VoxelNet unified point cloud feature learning with the RPN algorithm used for 3D detection tasks. Results showed strong performance. However, the method is too slow (4.4 FPS) for real-time applications, due to the 3D convolutions and the sparsity of voxels.

Lang et al., in [13], propose a method for 3D object detection named PointPillars that performs end-to-end learning using only convolutional layers for 2D data. The framework takes point clouds as inputs and outputs predictions of oriented 3D bounding boxes for vehicles, cyclists, and pedestrians. It comprises three main stages: a feature encoder network that transforms the point cloud to a sparse pseudo-image, a backbone consisting of a 2D CNN, and a detection head responsible for detecting and regressing 3D bounding boxes. The novel encoder network applies PointNet, [23], to learn features of point clouds organized in vertical columns (pillars) and it encodes the learned features as a pseudo-image. The pseudo-image is fed into a 2D object detection pipeline that outputs 3D bounding boxes. While only using LIDAR point clouds in the training stage, results demonstrated that the PointPillars network outperforms, in terms of Average Precision (AP), most fusion approaches that use LIDAR and images. Moreover, PointPillars can run at 62 FPS on both the BEV and 3D KITTI benchmarks [6], since 2D convolution operations are extremely efficient on modern GPUs. The speed and detection perfor-

mance make PointPillars highly suitable for practical applications in real-time driving scenarios.

2.4 3D Object Tracking

The task of 3D object tracking is to predict the location of an object, given its location in the previous frames. This task is dominated by methods that perform the tracking step after the detection step. 3D object tracking is an emerging investigating topic which has been increasingly attracting attention. An example for image-based and another for LiDAR-based approaches are presented in Secs. 2.4.1 and 2.4.2, respectively.

2.4.1 Image-based Methods

CenterTrack, in [40], is a point-based framework for joint object detection and tracking. In their approach, each object is represented by a unique point located at the center of its 2D bounding box. The point is tracked through time. A point is associated with each detected object therefore, at each frame, there is a constellation of objects represented by a heatmap of their respective points. The object detector used works solely on the input from two consecutive frames as well as a heatmap of previous tracklets. The output is an array that stores the offset between the current object center point and its center point in the previous frame. The offset is learned and the detector outputs predictions for the position of the object in the following frame. It learns to jointly detect objects from the current frame and their offsets relative to the previous frame. Since only consecutive frames are considered, a trade-off is made, the ability to have long-range tracks is discarded in favor of simplicity and speed. Results showed that this trade-off was worth it on 2D datasets and even on monocular 3D object tracking ones.

2.4.2 LIDAR-based Methods

By using the rich and fine-grained geometric structure information of point clouds, the 3D object tracking task should overcome the problems of 2D image-based tracking methods which include occlusion, scale variation and illumination issues because the geometric information captured by LIDAR prevails under bad visibility circumstances.

Simon et al., in [29], propose a real-time 3D object detection and tracking framework for point clouds, in the context of autonomous driving, as shown in Fig. 2.7. The framework, named Complexer-YOLO, takes LIDAR point clouds and RGB images as input and it is comprised of the tracking pipeline that incorporates the whole detection pipeline in itself. The detection pipeline starts by performing 2D semantic segmentation of the front camera images using a pre-trained and efficient off-the-shelf 2D semantic segmentation network [21]. The input point cloud is voxelized and by fusing the semantic scores, obtained from the front view image, a voxelized semantic point cloud is obtained. For the final 3D bounding box prediction, the Complex YOLO, [30] by the same authors, is used. The tracking pipeline starts by grabbing frames from the stream of LIDAR and RGB stream, then the whole detection pipeline is executed on those frames. Lastly, temporal information is used to improve the robustness and accuracy of multi-target tracking. Although object detection is achieved via a deep neural network, the tracking method is based on a classic method. Moreover, the authors noted some issues related with the IoU metric for 3D bounding box evaluation, namely the fact that it takes too long to process the IoU metric for rotating 3D bounding boxes and the fact that a bounding box rotated by an angle of 180° has the same IoU, despite having the opposite direction. To this end, a simplified and powerful evaluation metric for comparison of object detections, called Scale-Rotation-Translation score (SRFs), is proposed which speeds up inference time up to 20%.

Our method is based on the Complexer-YOLO idea. However, we do not perform point cloud voxelization since it is computationally expensive. Instead, we project the point cloud to the ground plane and feed it to a CNN. By doing this, we increase the FPS during inference by 38%. Moreover, we propose a deep learning network to achieve end-to-end joint detection and tracking, based on 3D semantic point clouds, in a faster way.

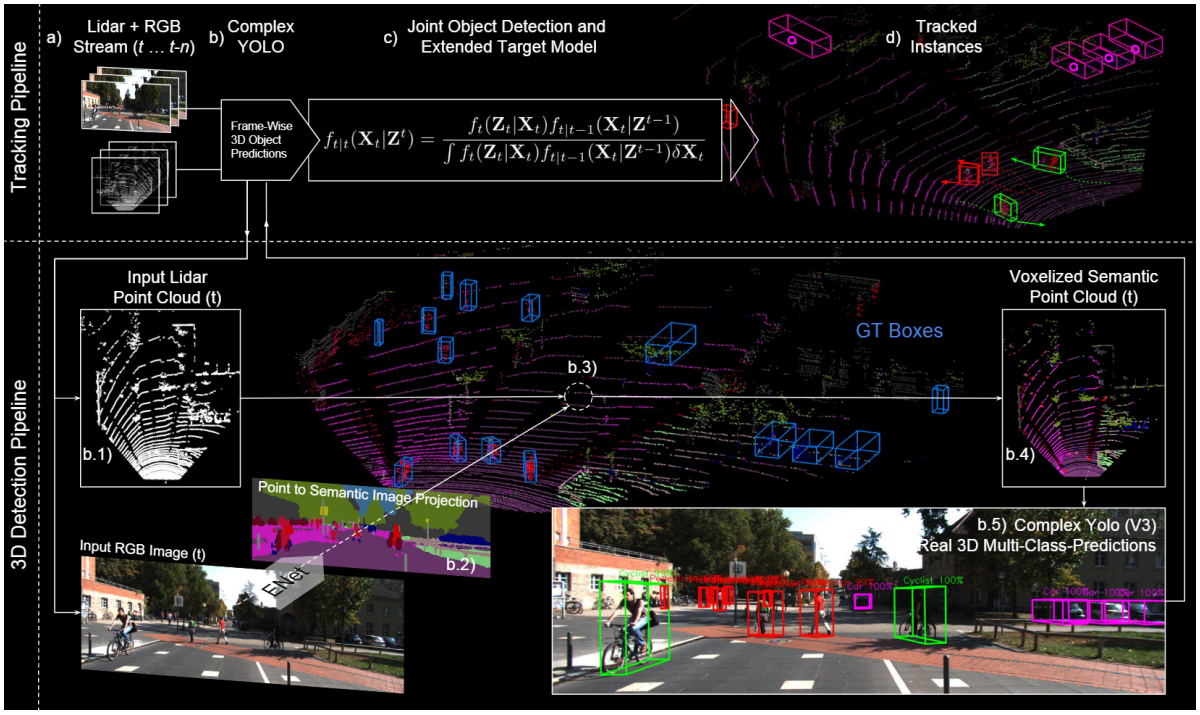


Figure 2.7: Complexer-YOLO framework. The 3D object detection pipeline is used in our proposed pipeline. Figure obtained from [29].

Chapter 3

Methods

In this chapter, a selection of the relevant methods is presented in more detail compared to the methods presented in Sec. 2, since these are the methods on which this thesis is based. Therefore, a full understanding of these methods is essential. Based on Complexer-YOLO [29], our pipeline feeds a 3D semantic point cloud to the Complex-YOLO [30] model, which is a 3D object detector. Since Complex-YOLO expands on YOLOv2 [25], the latter is detailed first in Sec. 3.1 and the former is presented in 3.2. No official code was released for Complex-YOLO. However, several implementations can be found online. Some are true to the original paper, thus, using YOLOv2 as the base for Complex-YOLO, while others, more recent versions, are based on YOLOv3 and YOLOv4, which are newer versions of the YOLO model. The chosen and later adapted source code[1] was an implementation of Complex-YOLO based on YOLOv4 [1] since it was the most recent version of YOLO at the time. The third and fourth versions of YOLO are consecutive improvements over YOLOv2. However, most aspects of the object detector remain the same. Key differences from YOLOv2 to YOLOv3 and YOLOv4 are detailed in Sec. 3.3 and Sec. 3.4, respectively.

3.1 YOLOv2

YOLOv2 [25] is a real-time 2D object detector whose input is a single RGB (Red Green Blue) image, $I \in \mathbb{R}^{H \times W \times 3}$. The network architecture comprises convolutional and pooling operations only. The final layer is a 1×1 convolutional layer with N filters.

[1] https://github.com/maudzung/Complex-YOLOv4-Pytorch

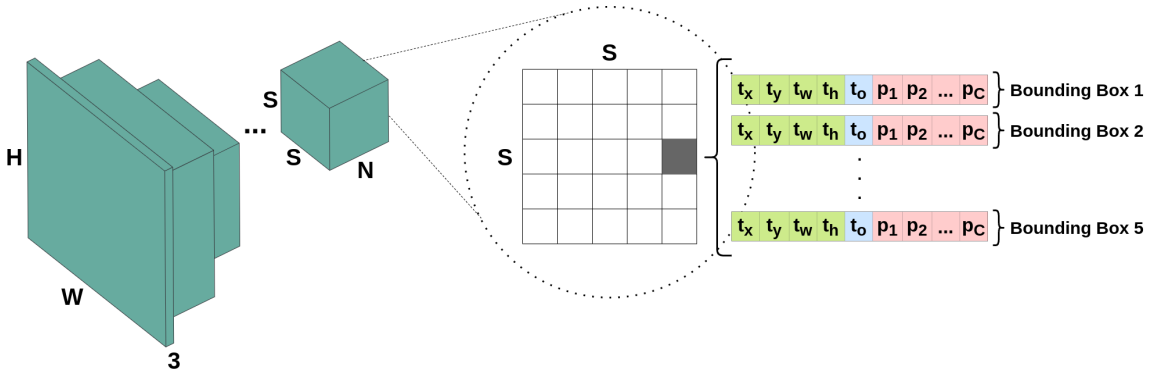


Figure 3.1: Given a square image $I \in \mathbb{R}^{H \times W \times 3}$, a single forward pass through the YOLOv2 network produces an output feature map $\hat{F} \in \mathbb{R}^{S \times S \times N}$, where $S = H/32 = W/32$. Each grid predicts $B = 5$ bounding boxes based on five different anchors. Each cell predicts $N = B \times (5 + C)$ parameters, including the boxes' spatial information, an objectness score, and C conditional class probabilities given there is an object inside the predicted box.

In the middle of the feature extraction, the input image I is downsampled throughout the network by a factor of 32, producing an output feature map $\hat{F} \in \mathbb{R}^{\frac{H}{32} \times \frac{W}{32} \times N}$. This feature map depicts a grid where each cell is responsible for predicting B bounding boxes. YOLOv2 uses anchor boxes to predict bounding boxes. These anchor boxes are selected via k-means clustering on the training set bounding boxes to generate fitting anchor boxes. The authors chose $k = 5$ as a good trade-off between high recall and model complexity. Therefore, for each cell in the output feature map, the network predicts $B = 5$ bounding boxes based on the $k = 5$ anchor boxes. For each bounding box, the network predicts its coordinates and dimensions, t_x, t_y, t_w, t_h, an objectness score t_o, and C class probabilities, one for each class of the dataset. Therefore, the number of channels in the output feature map is given by

$$N = B \times (5 + C). \tag{3.1}$$

The output feature map $\hat{F}$ consists of a division of the input image I onto a grid, whose cells represent regions that propose bounding box candidates. Therefore, YOLOv2 works like a single-shot RPN (Region Proposal Network), as illustrated in Fig. 3.1.

Each box predicts its coordinates, (t_x, t_y), in the output feature map as well as its dimensions, (t_w, t_h), based on the anchor box's dimensions. This process is detailed in Fig. 3.2. Besides the spatial information, each box predicts its objectness score, t_o, given

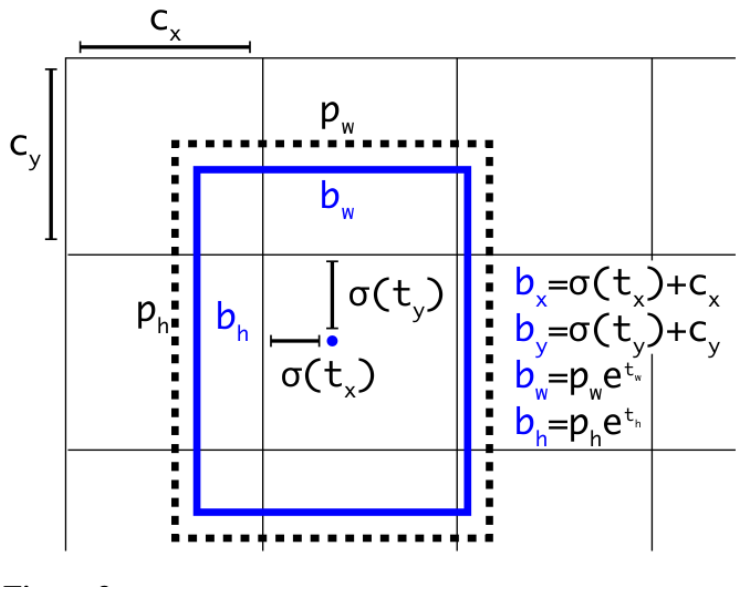


Figure 3.2: Example of YOLOv2's bounding box prediction (blue) based on an anchor box (dotted line). The coordinates predictions, t_x and t_y, pass through a sigmoid activation function to bound the values between $[0, 1]$, thus giving their relative location inside the grid cell. In the grid-like feature map, the bounding box coordinates b_x and b_y are obtained by their relative location inside the current cell, $\sigma(t_x)$ and $\sigma(t_y)$, added to the offset (cx, cy) from the top left corner of the grid, in order to obtain the location amid the whole feature map. The width and height of the bounding box, b_w and b_h, respectively, are scaled from the anchor box's dimensions. These results are then upscaled to the original input size. Figure taken from [25].

by

$$t_o = Pr(\text{Object}) \times IoU_{\text{prediction}}^{\text{ground truth}}, \tag{3.2}$$

which reflects how confident the model is that the box contains an object and how accurate the predicted box is. If no objects exist in a cell, the objectness score should be zero, otherwise, it should equal the IoU (Intersection over Union) between the ground truth and the predicted bounding box. Lastly, a class probability is predicted as a conditional probability of a detected object belonging to a class given that there is an object inside the predicted box, i.e., $P(\text{Class}_c|\text{Object})$, with $c \in C$.

3.1.1 YOLOv2 Loss Function

Each cell in the output feature map predicts $B = 5$ bounding boxes. Thus, the network predicts many irrelevant boxes that either (1) do not contain any object or (2) contain an object, but other boxes capture the object better, i.e., have higher IoU with the ground truth box. Therefore, the authors introduce the concept of a bounding box responsible for predicting an object. Should the center of an object fall into a grid cell of the output feature map, that grid cell is designated the responsible cell for detecting that object. This leads to $B = 5$ boxes within that responsible cell. At training time, only one box is chosen to be responsible for predicting an object. It is the one with the highest IoU between itself and the ground truth. YOLOv2's loss function is given by

$$
\begin{aligned}
L_{\text{YOLOv2}} = & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_{ij} - \hat{x}_{ij})^2 + (y_{ij} - \hat{y}_{ij})^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_{ij}} - \sqrt{\hat{w}_{ij}} \right)^2 + \left(\sqrt{h_{ij}} - \sqrt{\hat{h}_{ij}} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(O_{ij} - \hat{O}_{ij} \right)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(O_{ij} - \hat{O}_{ij} \right)^2 \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{Classes}} p_{ij}(c) \log \left(\hat{p}_{ij}(c) \right)
\end{aligned}
\tag{3.3}
$$

where $\mathbb{1}_{i}^{\text{obj}}$ depicts a binary variable which equals to one if an object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the j^{th} bounding box is responsible for that object prediction. Moreover, $\mathbb{1}_{ij}^{\text{noobj}}$ equals to one if (i) there is no object inside cell i or (ii) there is an object, but the j^{th} box of cell i is not responsible for that object. Otherwise, it equals zero. Therefore, the loss function only penalizes coordinate error for responsible boxes, and classification error is only penalized if an object exists in that grid cell, hence the conditional class probability mentioned above.

The loss function (3.3) consists of a multipart loss whose first and second terms represent the localization loss, the third and fourth terms represent the confidence (objectness) loss, and the last term depicts the classification loss. Localization and confidence loss

terms consist of the SSE (Sum of Squared Errors) in the model output. SSE equally weights width and height errors in large and small boxes. However, small variations of the boxes' dimensions should be penalized more in small boxes than in large ones. To remedy this, the model predicts the square root of the bounding box width and height instead of directly predicting the width and height of the bounding box. Lastly, the classification loss is only computed for a class given that there is an object in the cell i. It consists of the cross-entropy loss to measure the performance of the output probability value for each class.

In the YOLOv2 paradigm, the majority of predicted boxes are not responsible for detecting objects. This sets their objectness scores (confidence scores) to zero, thus, overwhelming the gradient from cells that contain objects. This leads to model instability, making the training diverge early on. To tackle this issue, a smaller weight is attributed to the confidence loss of non-responsible boxes, by multiplying its term with $\lambda_{\text{noobj}} = 0.5$. Similarly, the localization loss is multiplied by $\lambda_{\text{coord}} = 5$ to assign more weight to the localization error in the backward propagation of errors to assure convergence in the early stages of training.

At test time, the C conditional class probabilities are multiplied by the box's objectness score (3.2):

$$P(\text{Class}_c|\text{Object}) \times P(\text{Object}) \times \text{IoU}^{\text{ground truth}}_{\text{prediction}} = P(\text{Class}_c, \text{Object}) \times \text{IoU}^{\text{ground truth}}_{\text{prediction}}, c \in C. \quad (3.4)$$

This results in class-specific confidence (objectness) scores for each predicted box. These scores reflect the probability of a given class appearing in the box and how well the predicted box captures the object. Lastly, the iterative NMS (Non-Maximum Suppression) algorithm is applied. It filters the best fitting bounding box, for each object, out of all the predicted bounding boxes. It works by selecting the box whose objectness score is higher and then removing similar boxes with high IoU with the selected box.

3.2 Complex-YOLO

Complex-YOLO [30] is a real-time 3D object detector whose input is LiDAR-based only. It expands on YOLOv2, by predicting the orientation of the regressed bounding

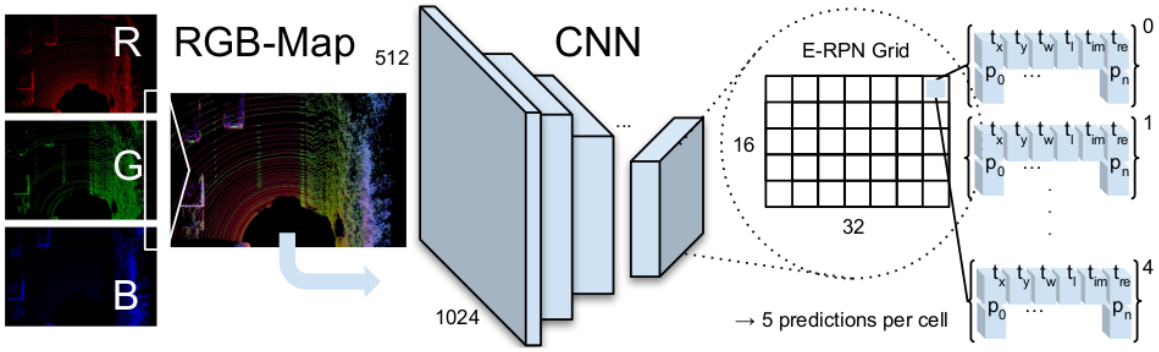


Figure 3.3: The Complex-YOLO pipeline. It takes as input the RGB-encoded handcrafted features of the point cloud, feeds it into a modified YOLOv2 network which predicts two extra box parameters: the real and imaginary part, t_{re} and t_{im}, of the heading angle of the predicted bounding box. Figure taken from [30].

boxes. The input point cloud is pre-processed into a BEV RGB image where each channel encodes handcrafted features. These channels encode the points' height, intensity, and density, respectively. The resulting BEV RGB map is fed into the Complex-YOLO network whose novel E-RPN (Euler Region Proposal Network) is responsible for estimating the heading of the object by adding a real and an imaginary fraction, t_{re} and t_{im}, to the regression network. With the oriented bounding boxes obtained on the BEV perspective, a pre-defined height for each class is attributed to the objects to output 3D bounding boxes. Complex-YOLO's pipeline is illustrated in Fig. 3.3.

Following YOLOv2, Complex-YOLO also predicts $B = 5$ bounding boxes per grid cell based on five different anchor boxes. Since boxes now comprise their orientation angle, the degrees of freedom increased, i.e., the number of possible anchors increased. However, Complex-YOLO keeps the number of predicted bounding boxes $B = 5$ for efficiency reasons. Therefore, based on the KITTI dataset box distribution, three different sizes and two angle directions were selected for anchor boxes, w.r.t BEV orientation: (1) car size (heading up), (2) car size (heading down), (3) cyclist size (heading up), (4) cyclist size (heading down), and (5) pedestrian size (heading left).

The novel E-RPN extends the Grid-RPN, used in YOLOv2, by also predicting a complex angle

$$b_\phi = arg(|z|e^{ib_\phi}) = \text{atan2}(t_{im}, t_{re}) \tag{3.5}$$

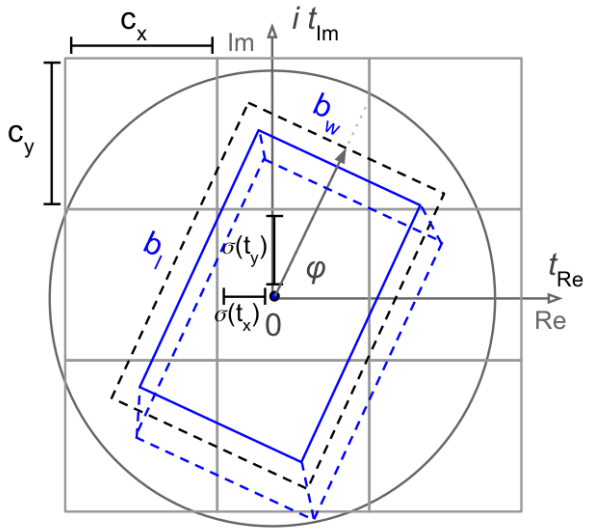


Figure 3.4: Example of a Complex-YOLO's bounding box prediction (blue solid line) based on an anchor box (black dashed line). The transition from the 2D BEV map to 3D is done by a predefined height for each class (blue dashed line). The new predicted parameters, t_{im} and t_{re}, encode the complex number $t_{re} + it_{im}$, whose phase is given by (3.5). Figure taken from [30].

encoded by an imaginary and real part, t_{im} and t_{re}, respectively, which are predicted by the output feature map. The estimated angle corresponds to the orientation of the predicted bounding box, as illustrated in Fig. 3.4. With the two added parameters to predict for each bounding box, the number of filters in the output feature map is now given by

$$N = B \times (7 + C). \tag{3.6}$$

Complex-YOLO loss function extends YOLOv2's loss function (3.3) by adding an Euler regression part:

$$L_{\text{Complex-YOLO}} = L_{\text{YOLO}} + L_{\text{Euler}}. \tag{3.7}$$

Complex-YOLO predicts a complex number $|\hat{z}|\, e^{i\hat{b}_\phi}$ encoded by t_{im} and t_{re}. However, the predicted modulus $|\hat{z}|$ value is irrelevant to the orientation of the proposed bounding box, consequently, the loss function assumes that both the predicted and ground truth complex numbers fall onto the unit circle, i.e $|\hat{z}| = 1$ and $|z| = 1$. Consequently, only phase estimation is penalized. The network aims to minimize the absolute value of the

squared error to get a real loss:

$$\begin{aligned} L_{\text{Euler}} &= \lambda_{\text{coord}} \sum_{n=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{nj}^{\text{obj}} \left| \left(e_{(nj)}^{ib_\phi} - e_{(nj)}^{\hat{ib}_\phi} \right)^2 \right| \\ &= \lambda_{\text{coord}} \sum_{n=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{nj}^{\text{obj}} \left[\left(t_{im}^{(nj)} - \hat{t}_{im}^{(nj)} \right)^2 + \left(t_{re}^{(nj)} - \hat{t}_{re}^{(nj)} \right)^2 \right] \end{aligned} \tag{3.8}$$

Similarly to (3.3), the Euler loss term is also scaled by the factor λ_{coord} to guarantee convergence in early training stages. Moreover, the prediction of t_{im} and t_{re} is only penalized for responsible bounding boxes, i.e., $\mathbb{1}_{nj}^{\text{obj}}$ equals to one only if the j^{th} bounding box predictor in cell n has the highest IoU between itself and the ground truth box.

3.3 YOLOv3

YOLOv3 [26] is the follow-up work of YOLOv2. Similarly, it takes as input a single RGB image, $I \in \mathbb{R}^{H \times W \times 3}$, and performs bounding box regression in the same way from an output feature map. However, YOLOv3 innovates by making predictions at three different scales, i.e., an output feature map, $\hat{F} \in \mathbb{R}^{\frac{H}{R} \times \frac{W}{R} \times N}$, is obtained for each scale. R denotes the downscaling factor of the feature maps and N denotes the number of channels, which, similarly to YOLOv2, is given by (3.1) with $B = 3$, i.e., it predicts $B = 3$ bounding boxes per grid cell of the output feature map.

YOLOv3's architecture (illustrated in Fig. 3.5) takes advantage of residual connections between lower-level features and higher-level features to build the output feature maps at each scale. Downscaling of the feature maps is achieved by convolutional operations to prevent the loss of low-level features attributed to pooling. The first scale outputs a feature map, $\hat{F}$, with a similar downscaling factor to YOLOv2, i.e., $R = 32$. This scale is adequate at detecting large objects but lacks in detecting small objects. Consequently, a second and third scale with downscaling factors $R = 16$ and $R = 8$, respectively, are introduced in the network. Predicting at a higher resolution feature map allows fine-grained bounding box regression, which results in better detection of small objects in the image.

[2] https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b. Accessible on July 10, 2021.

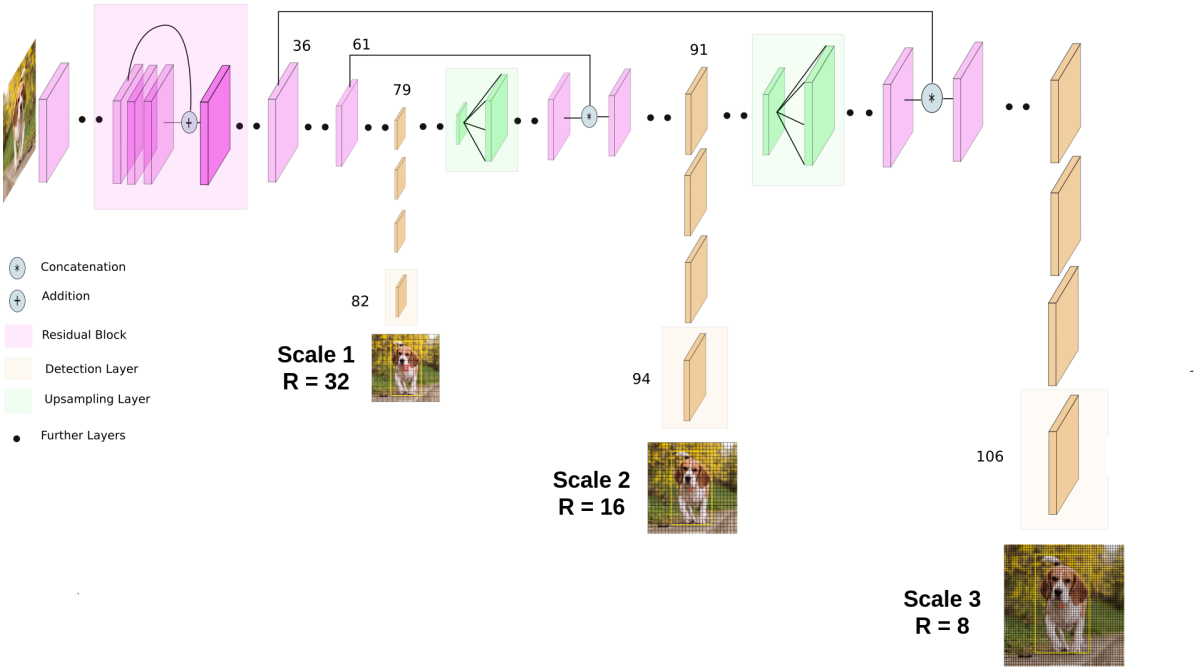


Figure 3.5: Simplified YOLOv3 architecture. From a single input image, three output feature maps are obtained, one for each scale. The network takes advantage of residual connections, i.e., shortcut connections, to concatenate low-level features with high-level features. The first scale, the shallowest of the three, detects large objects more easily but struggles with smaller objects. Deeper in the network, the second scale detects objects of medium size. The deepest of the three, the third scale, contains the higher-level features, which together with a higher resolution feature map allow better detection of small objects. Figure taken from[2].

As stated in Sec. 3.1.1, YOLOv2 computes its loss function in (3.3) concerning the output feature map. The same idea is applied in YOLOv3, thus, the following loss

function is computed for each scale of the output feature maps:

$$
\begin{aligned}
L_{\text{YOLOv3\_Scale\_s}} = \lambda_{\text{coord}} & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_{ij} - \hat{x}_{ij})^2 + (y_{ij} - \hat{y}_{ij})^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_{ij}} - \sqrt{\hat{w}_{ij}} \right)^2 + \left(\sqrt{h_{ij}} - \sqrt{\hat{h}_{ij}} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[-O_{ij} \log \left(\hat{O}_{ij} \right) - (1 - O_{ij}) \log \left(1 - \hat{O}_{ij} \right) \right] \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left[-O_{ij} \log \left(\hat{O}_{ij} \right) - (1 - O_{ij}) \log \left(1 - \hat{O}_{ij} \right) \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{Classes}} \left[-p_{ij}(c) \log \left(\hat{p}_{ij}(c) \right) - (1 - p_{ij}(c)) \log \left(1 - \hat{p}_{ij}(c) \right) \right].
\end{aligned}
\tag{3.9}
$$

It is identical to (3.3), however, the objectness (confidence) and classification terms (the last three terms) are now optimized by the Binary Cross-Entropy (BCE) loss. For classification, the BCE loss is used with logistic activation (sigmoid function) to allow for multi-label prediction, i.e., prediction of overlapping classes such as 'Car' and 'Vehicle', for the same object. YOLOv3 aims to optimize the sum of the three loss functions corresponding to each output scale. Consequently, its loss function is given by

$$
L_{\text{YOLOv3}} = \sum_{s=1}^{3} L_{\text{YOLOv3\_Scale\_s}}. \tag{3.10}
$$

3.4 YOLOv4

YOLOv4 [1] has an identical pipeline to its predecessor. However, it presented some modifications to increase the model performance. These innovations are two-fold: (i) changes that only increase the training cost and (ii) changes that slightly increase the inference cost but result in a significant accuracy improvement.

The changes which increase the training cost comprise essentially data augmentation techniques, which increase the variability of an image to increase the model's robustness and improve its generalization. Besides applying the usual techniques (crop, rotation,

hue, saturation), the authors introduced two novel types of data augmentation, mosaic augmentation and Self-Adversarial Training (SAT). Mosaic data augmentation stitches four images together, thus, letting the model learning to find smaller objects while paying less attention to neighboring scenes that are not immediately around the object. SAT works in two forward-backward stages. In the first stage, the network modifies the original image instead of its weights, to obscure the object present in the image. In the second stage, the network is trained to detect the object in the noisy image with the correct label, thus forcing the network to generalize to new features.

Despite having an identical working pipeline, YOLOv4 significantly altered the YOLOv3 network. These modifications significantly improve the accuracy of the model, with a slight increase in the inference cost. The feature extractor is redesigned to accommodate Cross-stage Partial (CSP) connections, a Spatial Pyramid Pooling (SPP) module, and a Path Aggregation Network (PANet).

CSP originated from CSPNet [36] which is based on DenseNet [9]. DenseNet was proposed to connect layers in CNN's to (i) reduce the vanishing gradient problem (i.e., the difficulty of backpropagating loss signals in very deep networks), (ii) reinforce feature propagation, (iii) promote features reuse, and (iv) reduce the number of network parameters. PANet was introduced in [18] to boost the propagation of low-level feature information to higher-level feature maps. In the original implementation of PANet, the low-level feature maps are added to the higher-level feature maps. YOLOv4 changes the addition operation to a concatenation operation. SPP was introduced in [8] as a means of eliminating the requirement of a fixed size input image to a CNN. YOLOv4 adds an SPP block after the feature extractor because it increases the receptive field, filters the most important context features obtained by the feature extractor while adding a very low cost to the network inference speed.

Despite the significant changes promoted to the architecture, YOLOv4 still predicts bounding boxes at three different scales. Applying the same equation term names from Sec. 3.1.1, YOLOv4 changed from SSE to MSE (Mean Square Error) for the localization

loss terms of each scale (the first two terms), and it is now given by:

$$
\begin{aligned}
L_{\text{YOLOv4\_Scale\_s}} = \lambda_{\text{coord}} \frac{1}{N} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_{ij} - \hat{x}_{ij})^2 + (y_{ij} - \hat{y}_{ij})^2 \right] \\
+ \lambda_{\text{coord}} \frac{1}{N} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_{ij}} - \sqrt{\hat{w}_{ij}} \right)^2 + \left(\sqrt{h_{ij}} - \sqrt{\hat{h}_{ij}} \right)^2 \right] \\
+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[-O_{ij} \log \left(\hat{O}_{ij} \right) - (1 - O_{ij}) \log \left(1 - \hat{O}_{ij} \right) \right] \\
+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left[-O_{ij} \log \left(\hat{O}_{ij} \right) - (1 - O_{ij}) \log \left(1 - \hat{O}_{ij} \right) \right] \\
+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{Classes}} \left[-p_{ij}(c) \log \left(\hat{p}_{ij}(c) \right) - (1 - p_{ij}(c)) \log \left(1 - \hat{p}_{ij}(c) \right) \right],
\end{aligned}
\tag{3.11}
$$

where N denotes the number of predicted bounding boxes. YOLOv4 loss function is given by

$$
L_{\text{YOLOv4}} = \sum_{s=1}^{3} L_{\text{YOLOv4\_Scale\_s}}. \tag{3.12}
$$

Chapter 4

Proposed Pipeline

Our pipeline performs semantic segmentation on a 2D image and fuses this information (using a camera projection matrix) with a 3D point cloud to obtain a 3D semantic point cloud. Then, it performs 3D object detection and tracking on this rich environment representation that offers both semantic and depth information. In this chapter, the developed pipeline is described, and the choices behind the design are justified. The dataset of choice, the Semantic Segmentation module, the 3D Object Detection, and the 3D Object Detection and Tracking module are described in detail in Sec. 4.1, Sec. 4.2, Sec. 4.3, and Sec. 4.4, respectively.

4.1 KITTI dataset

The KITTI dataset [6] was chosen due to its popularity and widespread use among research methods that apply deep learning techniques focused on autonomous driving scenarios. Two branches of the dataset were used, the 3D Object Detection and the 3D Multi-Object Tracking datasets. The former is a set of unordered frames and was used to train and evaluate the object detector. The latter is a set of sequences of frames and was used to train and evaluate the proposed joint object detector and tracker. A third branch of the KITTI dataset, the semantic segmentation dataset, was used only by the pre-trained semantic segmentation model. The contents of these three branches of the KITTI dataset are described in Table 4.1.

The 3D Object Detection and 3D Multi-Object Tracking datasets consist of frames from the KITTI vehicle's environment. An RGB image and a 3D point cloud for an

Table 4.1: This table summarizes the contents of each branch of the KITTI dataset, relevant to the proposed pipeline. Only the 3D Object Detection and 3D Multi-Object Tracking datasets include 3D information (point clouds).

	Labeled frames	Frame contents
Semantic Segmentation	200	RGB Image Pixel-wise Label
3D Object Detection	7481	RGB Image Velodyne Point Cloud Calibration Matrices 3D Object-wise Label
3D Multi-Object Tracking	8004	RGB Image Velodyne Point Cloud Calibration Matrices 3D Object-wise Label with Track ID

example frame are illustrated in Fig. 4.4 and 4.6, respectively. Each frame corresponds to a full spin of the LiDAR scanner, with a sampling frame rate equal to 10 Hz. As seen in Fig. 4.1, the vehicle is equipped with two grayscale cameras (indexed 0 and 1) and two color cameras (indexed 2 and 3). The monocular RGB images used are captured by Camera 2. Every time the Velodyne laser scanner (LiDAR) rotates to the vehicle's forward position, the cameras are triggered to capture one image, guaranteeing the synchronization between the camera and LiDAR sensors.

4.1.1 Class Distribution

Both 3D Object Detection and 3D Multi-object Tracking datasets label seven different classes: Car, Van, Truck, Cyclist, Pedestrian, Person Sitting, and Tram. The class distribution for these datasets is detailed in Table 4.2. The dominance of the Car class will reflect on the experimental results, as detailed in Chapter 5.

4.1.2 Calibration Matrices

To project a 3D point $\mathbf{x} = [x, y, z, 1]^T$ in the world, in homogeneous coordinates, to a 2D point $\mathbf{y} = [u, v, 1]^T$ in the i^{th} camera image plane, the equation

$$\mathbf{y} \sim \mathbf{P}_{\text{rect}}^{(i)} \mathbf{R}_{\text{rect}}^{(0)} \mathbf{T}_{\text{cam}}^{\text{velo}} \mathbf{x} \tag{4.1}$$

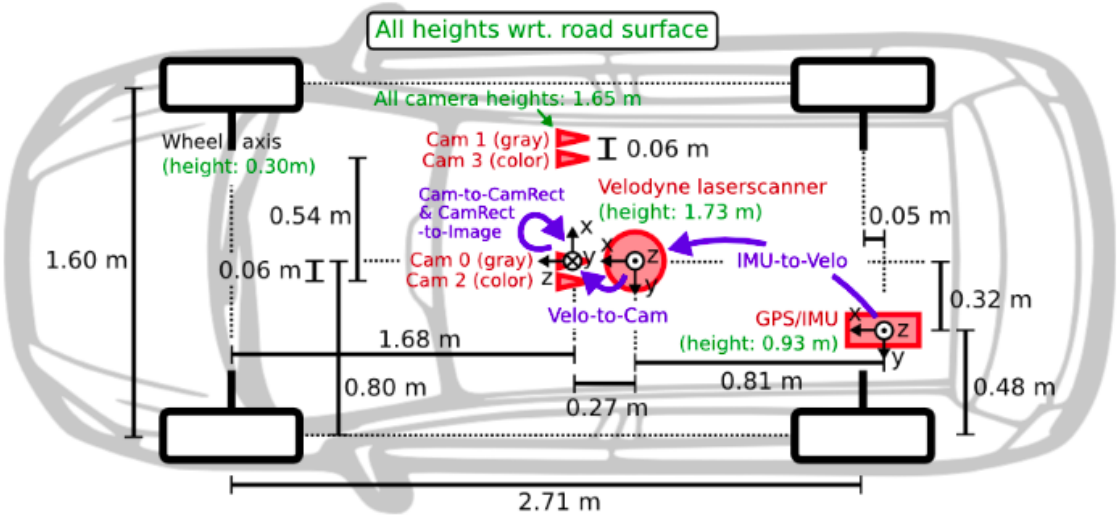


Figure 4.1: Sensor setup of the KITTI dataset: sensors illustrated in red and transformations among sensors shown in purple. All cameras lie on the same xy plane. (Figure taken from [5])

Table 4.2: Dataset class distribution (percentage) of the relevant KITTI branches. Clear domination of the Car class, on both datasets, makes them imbalanced datasets.

	Dataset Class Distribution (%)	
Class	3D Object Detection	3D Multi-object Tracking
Car	72.7	70.5
Van	7.3	8
Truck	2.8	3.8
Cyclist	4	4.1
Pedestrian	11.4	12.8
Person Sitting	0.6	0
Tram	1.2	0.8

is applied. Here, $\sim$ means the equation is defined up to a scale factor, $\mathbf{P}_{\text{rect}}^{(i)} \in \mathbb{R}^{3\times4}$ denotes the camera projection matrix, after rectification, of the i^{th} camera, and is given by

$$\mathbf{P}_{\text{rect}}^{(i)} = \begin{bmatrix} f_u^{(i)} & 0 & c_u^{(i)} & -f_u^{(i)} b_x^{(i)} \\ 0 & f_v^{(i)} & c_v^{(i)} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{4.2}$$

This matrix describes the mapping of 3D points in the world, through a pinhole camera model (as shown in Fig. 4.2), to 2D points in the i^{th} camera image plane. Following

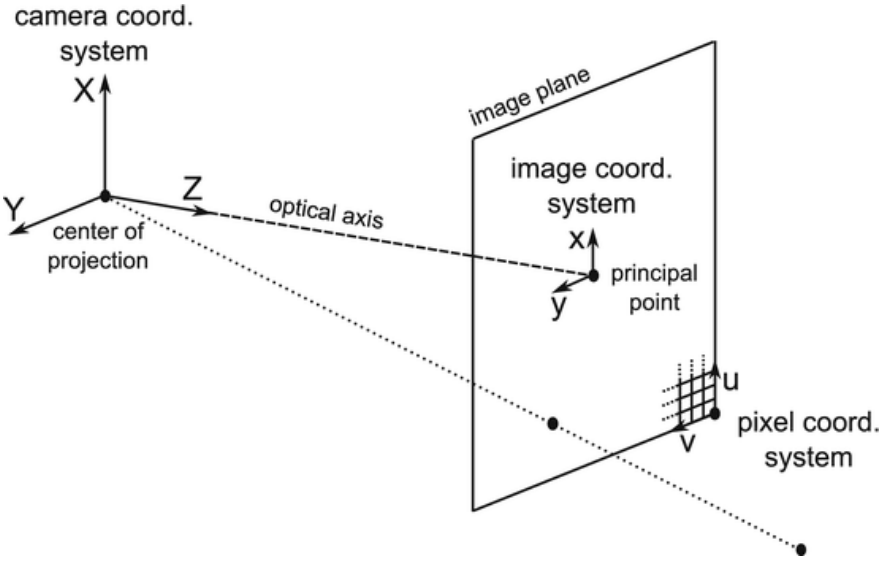


Figure 4.2: The pinhole camera model. The pinhole is located at the origin of the camera coordinate system. The focal length parameter is defined as the distance between the pinhole and the image plane. Figure taken from [31].

the pinhole camera model (Fig. 4.2), the pair (u, v) denotes the pixel coordinate system on the image plane, $f_u^{(i)}$ and $f_v^{(i)}$ denote the focal length along the u axis and v axis, respectively. Moreover, the pair $(c_u^{(i)}, c_v^{(i)})$ denotes the principal point of the camera. Lastly, $b_x^{(i)}$ denotes the baseline w.r.t the reference camera, i.e., the distance, in meters, between the i^{th} camera and camera number 0. Its value is $0.06m$ (as illustrated in Fig. 4.1) and it adds the offset relative only to the x axis, since all cameras lie on the same xy plane and their displacement is along the x axis, in the camera coordinate system.

$\mathbf{R}_{\text{rect}}^{(0)} \in \mathbb{R}^{4\times4}$ is the rectifying rotation matrix of the reference camera, i.e., camera number 0. Rectification consists of applying a rotation to make images of distinct cameras lie on the same plane [4].

Finally, $\mathbf{T}_{\text{velo}}^{\text{cam}} \in \mathbb{R}^{4\times4}$ denotes the 3D rigid-body transformation that takes points from the Velodyne coordinate system to the camera coordinate system. This transformation matrix is given by

$$\mathbf{T}_{\text{velo}}^{\text{cam}} = \begin{bmatrix} \mathbf{R}_{\text{velo}}^{\text{cam}} & \mathbf{t}_{\text{velo}}^{\text{cam}} \\ 0 & 1 \end{bmatrix} \tag{4.3}$$

where $\mathbf{R}_{\text{velo}}^{\text{cam}} \in \mathbb{R}^{3\times3}$ denotes the rotation matrix from the Velodyne coordinate system to the camera coordinate system and $\mathbf{t}_{\text{velo}}^{\text{cam}} \in \mathbb{R}^{3\times1}$ denotes the translation vector from

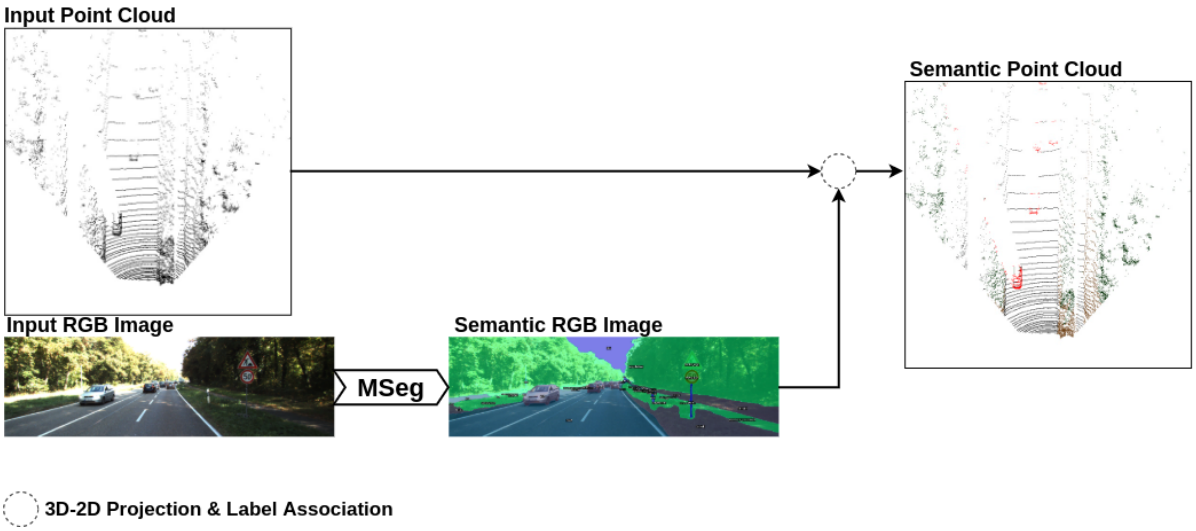


Figure 4.3: Semantic Segmentation Module

the Velodyne coordinate system to the camera coordinate system.

4.2 Semantic Segmentation Module

In this module, the goal is to obtain a 3D semantic point cloud, i.e., a point cloud whose points contain not only their location information (x, y, z) but also their class label. To take advantage of the dense environment representation, the semantic segmentation is performed, pixel-wise, on 2D images and then projected to the respective 3D points. The proposed 3D Semantic Segmentation module is depicted in Fig. 4.3.

4.2.1 MSeg model

Following the SOTA methods for semantic segmentation [15, 19, 32, 23, 3, 16, 12], which are all based on deep learning techniques, using a deep learning network to perform semantic segmentation was the obvious choice. Since the semantic segmentation module will be fed with images from the KITTI domain, the semantic segmentation network should be trained on RGB images from the Semantic Segmentation branch of the KITTI dataset. However, the KITTI Semantic Segmentation dataset is quite limited, as it only provides 200 labeled RGB images (see Table 4.1). Training a deep learning network on such a low number of samples would lead to poor results in the task at hand.

Figure 4.4: RGB image captured by Camera 2 of the KITTI sensor setup.

A solution to this problem is presented by the MSeg dataset [11]. The authors introduce a composite dataset that combines several existing semantic segmentation datasets, including the KITTI dataset, into a unified domain. Their composite dataset enables training a single semantic segmentation network, which is effective even when tested on individual datasets.

Besides solving the scarce dataset problem, the authors in [11] also offer a plethora of pre-trained models[1], ready to use with the KITTI dataset. The available semantic segmentation models, similarly named MSeg, are evaluated by the mean Intersection over Union (mIoU) metric. For the semantic segmentation task, the Intersection over Union (IoU) metric is given by

$$\text{IoU} = 100 \cdot \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalseNegatives} + \text{FalsePositives}} \tag{4.4}$$

which is calculated, for each class, at pixel-level. The mIoU is then calculated by averaging the IoU values of each class. The selected pre-trained model was the one that presented the highest mIoU value for the KITTI dataset.

Performing semantic segmentation on the RGB images, from the 3D Object Detection and 3D Multi-Object Tracking datasets, implies feeding the images to the pre-trained MSeg model. In other words, the input of the model consists of an image, such as the one shown in Fig. 4.4. The network outputs pixel-wise class labels, predicted from 193 different class labels. Assigning a unique color to each class allows for visualization of the predicted semantic segmentation results (as illustrated in Fig. 4.5).

[1] https://github.com/mseg-dataset/mseg-semantic

Figure 4.5: Semantic segmentation labels overlaid on the RGB image from Fig. 4.4

4.2.2 Projecting Semantic Labels to Point Cloud

With the semantic segmentation results obtained for the RGB image, the final step of this module consists of mapping 3D points in the world to pixels in the image plane, in order to obtain a 3D semantic point cloud (as shown in Fig. 4.6).

Initially, to simplify computation, every point in the point cloud is converted to homogeneous coordinates, i.e., a point $\mathbf{x} = [x, y, z]^T$ now has the coordinates $\mathbf{x} = [x, y, z, 1]^T$. Next, (4.1) is applied to project the whole point cloud onto the 2nd image plane (RGB is captured by camera number 2). Then, the whole point cloud is normalized by the z-coordinate to convert to pixel coordinates. Given the whole point cloud in pixel coordinates of the 2nd image plane, the points that live outside the camera FoV are removed. Finally, having every 3D point inside the camera FoV associated with a pixel from camera 2, the correspondent pixel-wise class label is copied onto the respective 3D point.

4.3 3D Object Detection Module

This module picks up where the Semantic Segmentation module left off. The 3D object detection module is based on the Complex-YOLO pipeline [30]. As stated in Sec. 3.2, Complex-YOLO is a 3D object detector that takes as input an RGB image, obtained from a point cloud's BEV snapshot, and outputs oriented bounding boxes in the BEV RGB map (as illustrated in Fig. 3.3). As stated in Sec. 1, the selected Complex-YOLO pipeline consists of an implementation based on YOLOv4. Henceforth, this implementation will be called Complex-YOLOv4. The proposed 3D Object Detection module is illustrated in Fig. 4.7.

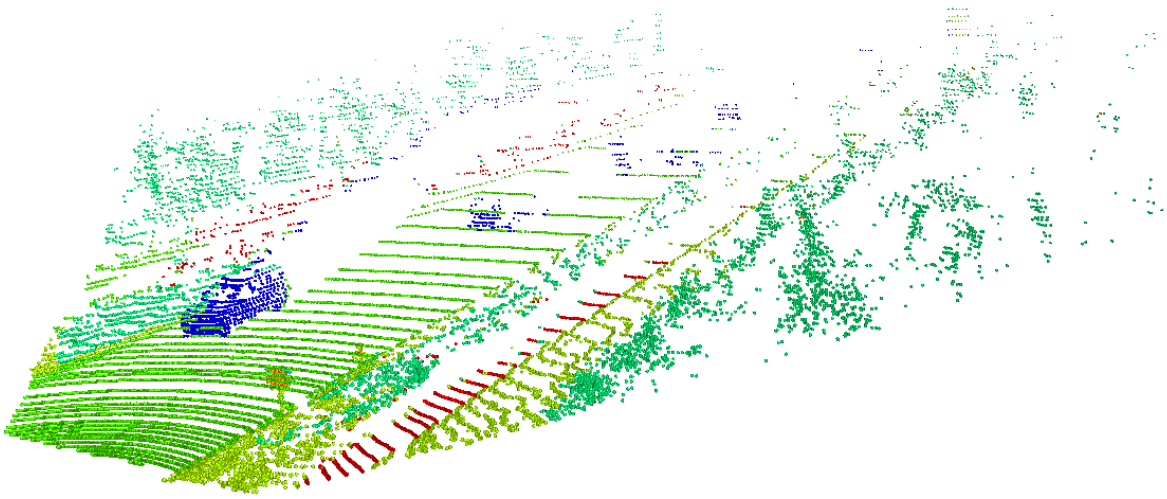


Figure 4.6: The 3D semantic point cloud was obtained by projecting the semantic segmentation results on the image plane (Fig. 4.5) to the point cloud. Points outside the camera FoV do not contain a semantic label, thus, they are filtered out.

4.3.1 Changes to the Complex-YOLOv4 implementation

The Complex-YOLOv4 pipeline was altered to accommodate the semantic information obtained in the Semantic Segmentation module (Sec. 4.2). The main difference to the original pipeline, to achieve 3D object detection only, lies in the network input. Further changes to embed the tracking task to this pipeline are later explained in Sec. 4.4. For now, only the changes made to achieve object detection on semantic point clouds are described. Originally, the network takes as input a BEV RGB map where each channel consists of handcrafted features, such as the points' height, intensity, and density, respectively. Now, the Complex-YOLOv4 network takes as input a BEV RGB map where the RGB channels encode the points' semantic class labels. A side-by-side comparison between the two RGB encoding methods is illustrated in Fig. 4.8.

Another change promoted to the Complex-YOLOv4 pipeline is related to its network architecture, in order to predict and detect an extra class, the Truck class. The KITTI dataset labels seven different classes of objects: Car, Van, Truck, Cyclist, Pedestrian, Person Sitting, and Tram. The Complex-YOLOv4 implementation follows the original paper and predicts only three classes: Car, Pedestrian, and Cyclist. It does so by grouping Car and Van into the same Car class, Pedestrian and Person Sitting into the unique Pedestrian class, and Cyclist needs no further grouping. Therefore, the

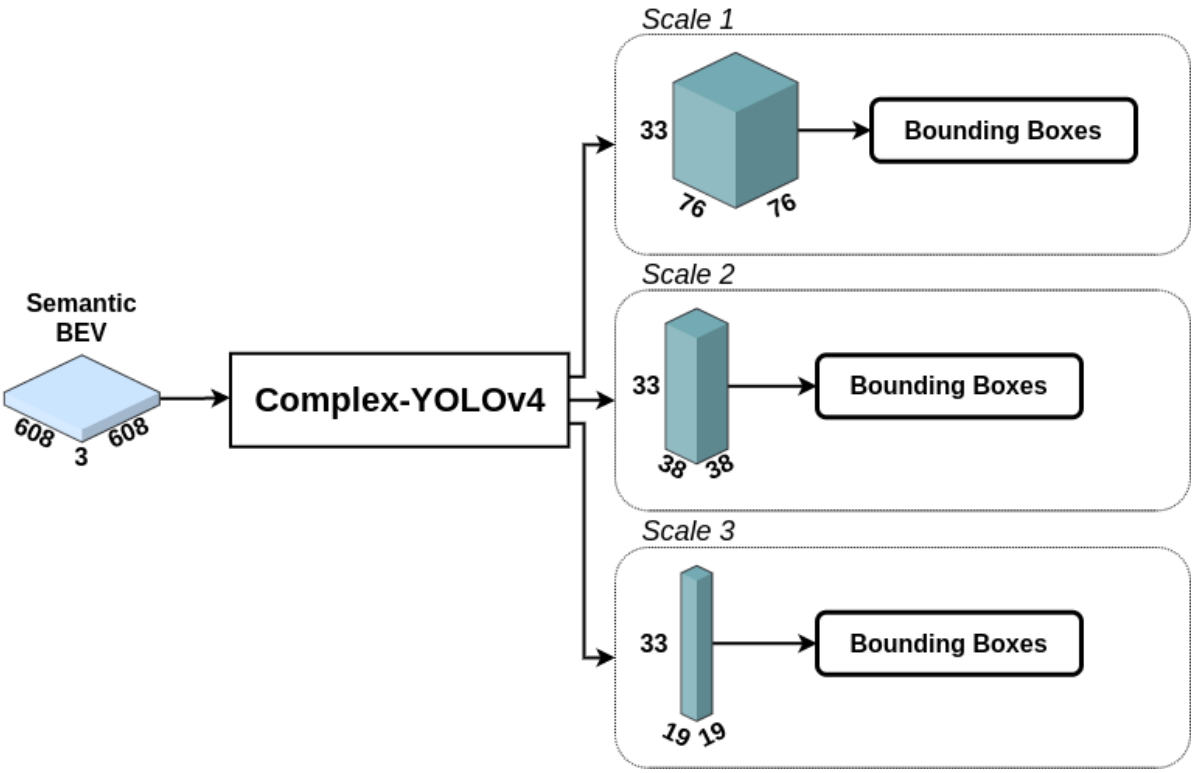
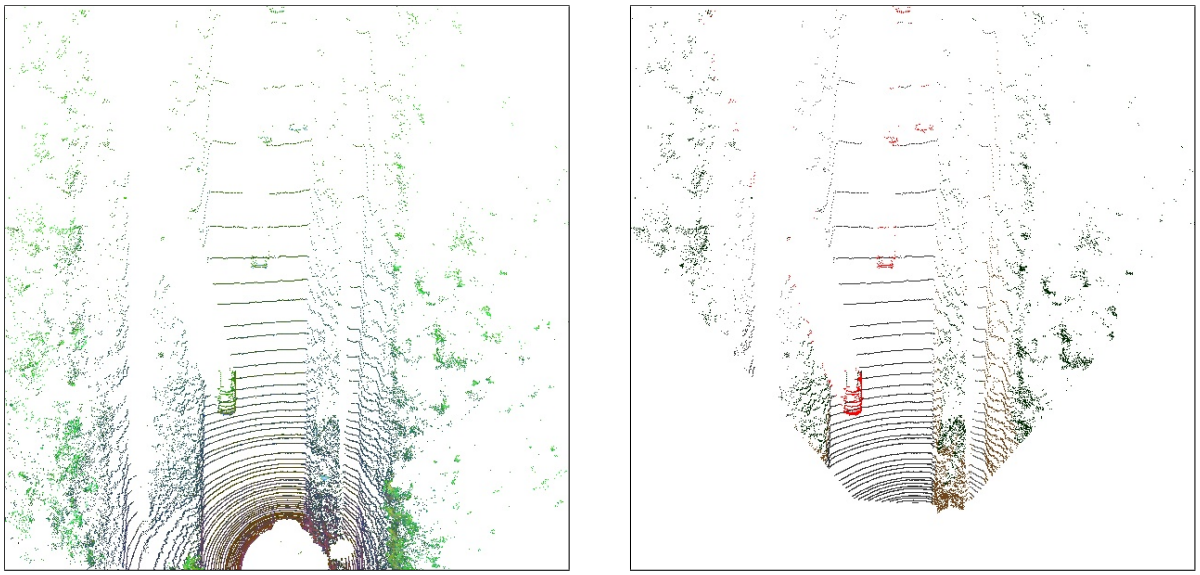


Figure 4.7: The 3D Object Detection Module takes as input a single square image consisting of a semantic BEV snapshot of the point cloud. The three output feature maps share the same number of channels, N, given by (3.6). With $B = 3$ predicted bounding boxes and $C = 4$ predicted classes, $N = 33$. The output spatial dimensions result from downscaling the input image with factors $R = 8$, $R = 16$, and $R = 32$, for scales 1, 2, and 3, respectively.

classes Truck and Tram were ignored by the model. Since Trams are rarer than Trucks on the dataset (see Table 4.2), only objects whose class was Truck were made to be predicted/detected by the network. The change consists of incrementing the number of classes, denoted by C, in (3.6). A side-by-side comparison between the test results, on a frame containing Trucks, before and after the change took place is illustrated in Fig. 4.9.

In short, the 3D object detection module comprises the Complex-YOLO pipeline, based on YOLOv4, whose input RGB encodes semantic labels and allows the detection of objects from the Truck class.

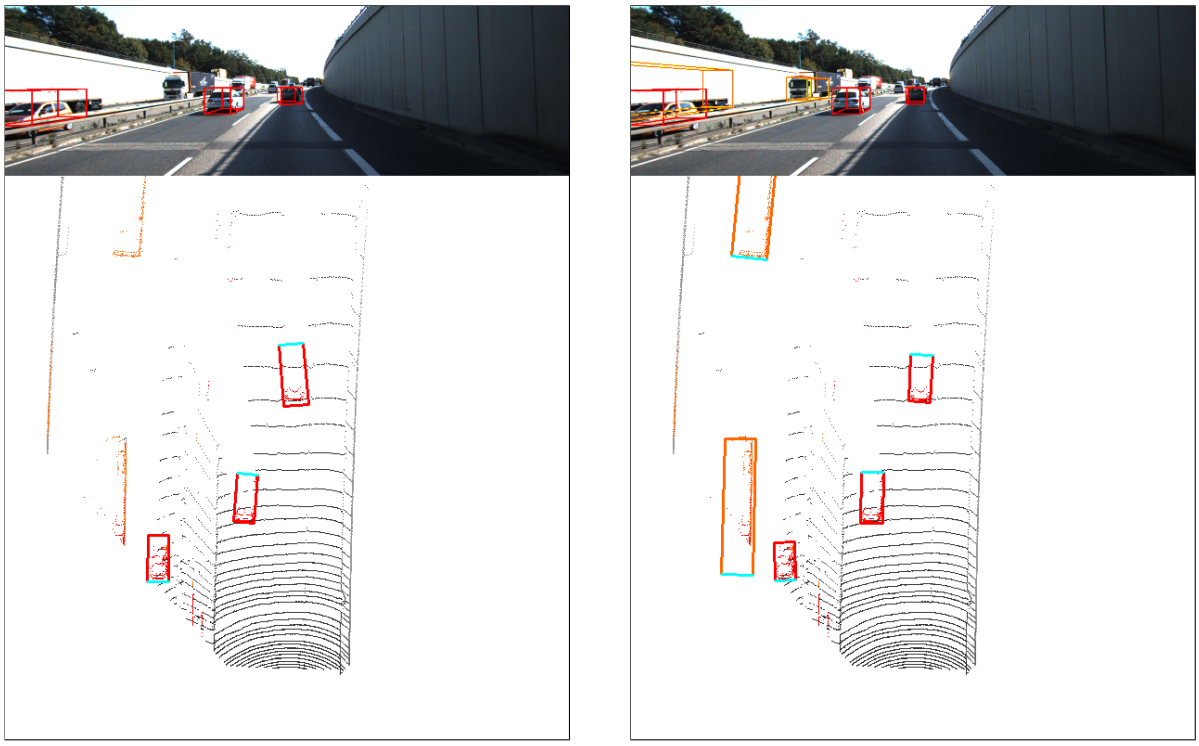


(a) Handcrafted RGB Map (b) Semantic RGB Map

Figure 4.8: Example of the same frame BEV RGB map encoded with: (a) Complex-YOLO's handcrafted features and (b) semantic class labels (used in our approach). The handcrafted RGB map depicts every point inside the 50x50m square, whereas the semantic RGB map is restricted to the 2nd Camera's FoV, as described in Sec. 4.2.2.

4.3.2 Obtaining the BEV RGB Map

In Velodyne coordinates (illustrated in Fig. 4.1), the BEV perspective of the semantic point cloud is obtained by discarding the z-coordinate and mapping the xy plane (ground plane) onto an image plane. The labelled points whose coordinates fall into the square delimited by $x \in [0, 50]m$ and $y \in [-25, 25]m$ are mapped onto a square image with 608×608 resolution, thus preserving the aspect ratio. The mapping is simply done by converting the points' coordinates in meters to pixel coordinates, where each pixel corresponds to approximately $0.08m$. Finally, the pixels are assigned a color according to the respective point's class label previously obtained in the Semantic Segmentation module. The class-color pair association is given by a dictionary that reduces the 193 different classes output by MSeg to nine classes and consequently nine different colors. This dictionary is detailed in Table 4.3. Figs. 4.8b and 4.9 depict BEV RGB maps obtained by this semantic class-color dictionary.



(a) Truck objects being ignored

(b) Truck objects being detected

Figure 4.9: A testing sample frame showing the difference between bounding box predictions, (a) before and (b) after the Truck class was detectable by the model. The detections are solely made on the BEV RGB map. The bounding boxes in the RGB image, captured by Camera 2, are for display purposes only.

4.4 3D Object Detection and Tracking Module

The 3D Object Detection and Tracking module picks up where the 3D Object Detection module left off. Further changes to the ones presented in Sec. 4.3.1 were promoted to embed the tracking task into the 3D object detection model based on Complex-YOLOv4. Adapting the Complex-YOLO architecture to predict object's offsets (dx, dy) implies adding two additional parameters to their bounding box tensor, as illustrated in Fig. 4.11. This predicted offset between two consecutive frames is the core mechanism for our tracking method, since it allows the prediction of objects' velocity, as detailed in Sec. 4.4.3. Besides the offset prediction, heatmap prediction was also embedded within the network, and the input was changed to accept two consecutive BEV frames and a

Table 4.3: Dictionary responsible for reducing the 193 semantic class labels to 9 meaningful classes for the road environment. Irrelevant classes, such as indoors' object classes, that don't show up in the road scenario, are discarded, thus painted white. Minute classes such as Wall, Pillar, and Window convert to the unique Building class. Bright and contrasting colors are attributed to the four classes to be detected (Car, Pedestrian, Cyclist, Truck), whereas dim and similar colors are attributed to non-detectable classes.

Class Label	**Pixel Color**
Car	Red
Pedestrian	Green
Cyclist	Blue
Truck	Yellow
Road	Dark Grey
Sidewalk	Grey
Building	Light Grey
Terrain	Brown
Vegetation	Dark Green
Everything Else	White

heatmap from the previous frame.

Based on CenterTrack [40], the goal is to predict object-wise bounding boxes, their offsets relative to the previous frame, and a heatmap depicting the objects' locations in the current frame. The input consists of (i) the frame whose objects will be detected, (ii) the previous frame, and (iii) the heatmap with the objects' location on the previous frame. In our case, the current and previous frames consist of Semantic BEV RGB maps. The 3D Object Detection and Tracking module is illustrated in Fig. 4.10. At test time, it comprises a feedback loop that feeds the pipeline on instant t with the heatmap predicted on the previous instant, *t-1*. During training, the input heatmap consisted of the ground truth. Consequently, a heatmap dataset was produced to train the network.

4.4.1 Heatmap Dataset and Prediction

The goal of the heatmaps is to feed the network the location in the previous frame of the objects to be detected in the current frame. A dataset of ground truth heatmaps was generated so the network could compare its output with the ground truth, therefore, learning how to predict heatmaps. Ground truth heatmaps are also used during training as the input heatmap from the previous frame. In other words, the feedback loop does

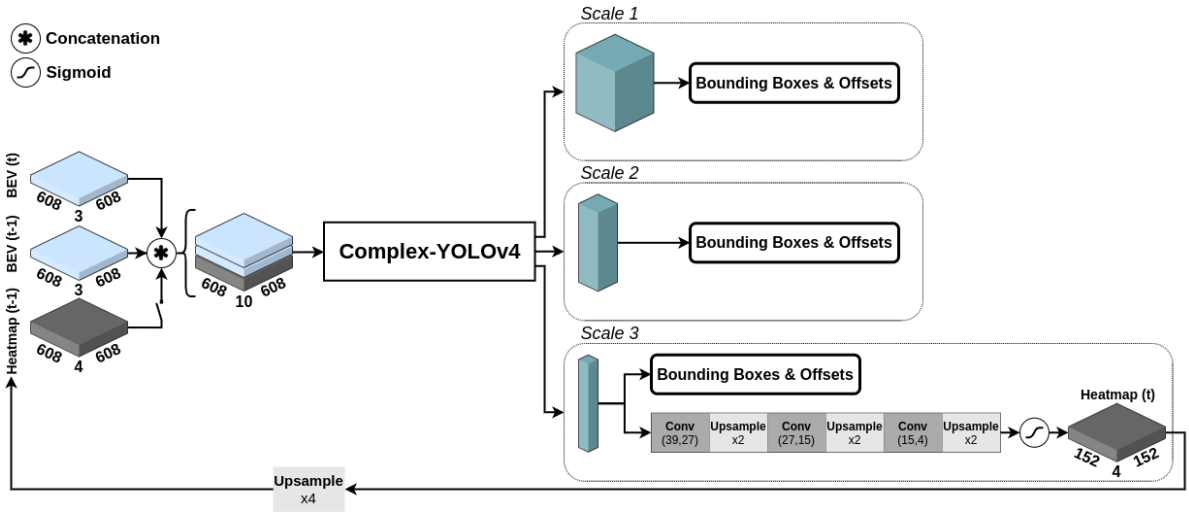


Figure 4.10: 3D Object Detector and Tracking module. The three-fold input is concatenated to enter the network as a single stacked tensor. The circuit switch breaker before the heatmap concatenation highlights the conducted ablation studies relative to the model input. Heatmap prediction is solely made on the deepest feature maps, i.e., on the third scale. The sigmoid activation function after the upsample operation bounds the values of the heatmap between 0 and 1.

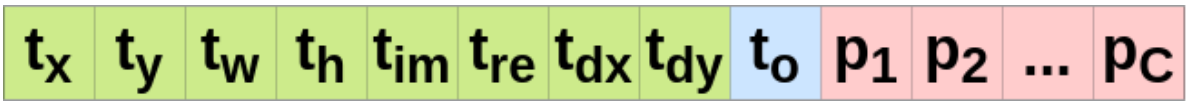
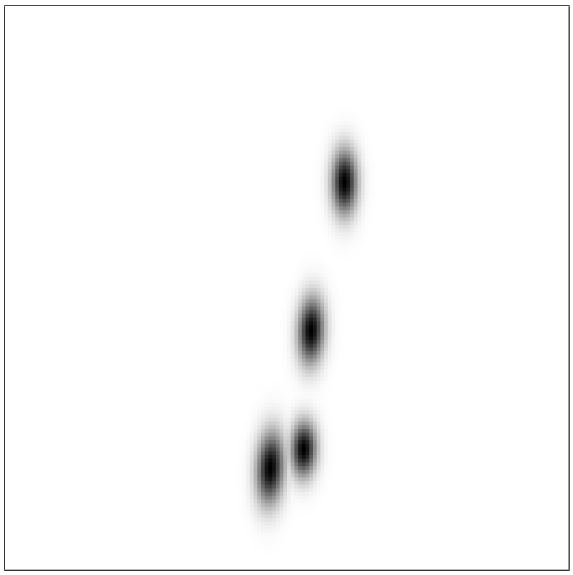


Figure 4.11: A bounding box predicted by the 3D Object Detector and Tracking module. Two extra parameters are learned in order to predict a bounding box's displacement between two consecutive frames, t_{dx} and t_{dy}.

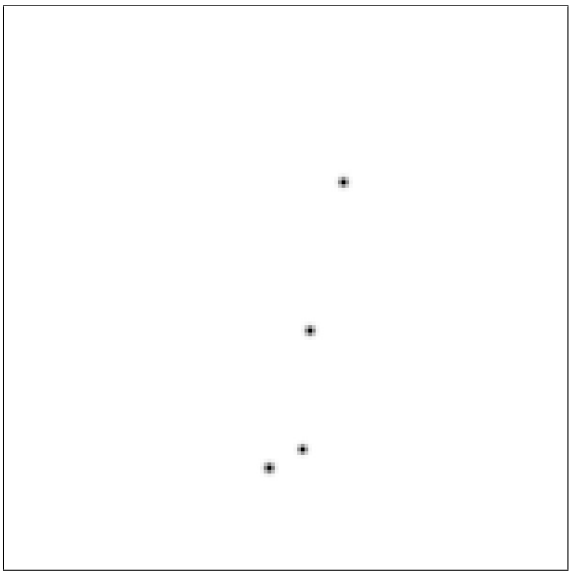
not exist during training. Thus, it is performed in an offline manner.

Ground truth heatmaps are fed to the network with the same orientation as the Semantic BEV RGB maps to make the learning task easier. Should the heatmaps have a different orientation than the Semantic BEV inputs, the network would have to learn this rotation as well. A heatmap is a 4 channel tensor, where each channel corresponds to the $C = 4$ different classes. Each of these channels consists of a grayscale image where the background is white and the splattered objects' centroids are colored in shades of gray.

Two different heatmap styles were generated and tested. Initially, centroids were depicted as ellipses to better resemble the rectangular bounding boxes. Results showed



(a) Elliptical Heatmap (Ours)



(b) Circular Heatmap (CenterTrack)

Figure 4.12: Comparison between (a) an heatmap with elliptical centroids and (b) the same heatmap with circular centroids. Both heatmaps depict the same frame illustrated in Fig. 4.8. The difference in their generation lives in the standard deviation parameter which directly defines the shape of the gaussian blob.

that this idea didn't work so well, as detailed in Sec. 5.4. Consequently, the second type of heatmaps was tested where the centroids depict circular shapes adaptive to the object's dimensions, similarly to CenterTrack's method. A comparison between the two heatmap styles is illustrated in Fig. 4.12.

Objects' locations are splat onto the heatmap using a 2D gaussian kernel

$$g(x, y) = A \cdot \exp\left(-\left(a(x - x_0)^2 + 2b(x - x_0)(y - y_0) + c(y - y_0)^2\right)\right) \tag{4.5}$$

where,

$$\begin{aligned} a &= \frac{\cos^2\theta}{2\sigma_X^2} + \frac{\sin^2\theta}{2\sigma_Y^2} \\ b &= -\frac{\sin 2\theta}{4\sigma_X^2} + \frac{\sin 2\theta}{4\sigma_Y^2} \\ c &= \frac{\sin^2\theta}{2\sigma_X^2} + \frac{\cos^2\theta}{2\sigma_X^2}. \end{aligned} \tag{4.6}$$

In (4.5), the pair (x_0, y_0) denotes the mean value of the normal distribution and it

equals to the coordinates of the splattered object's centroid. The amplitude value $A = 1$ in order to get normalized values, i.e., within the range $[0, 1]$. In (4.6), the standard deviation parameter is what gives the centroids' gaussian blobs different shapes. For the elliptical centroids, $\sigma_X = w/2$ and $\sigma_Y = l/2$, where w and l depict the object's width and length, respectively. For circular centroids, $\sigma_X = \sigma_Y$ and their value is a function of the object size [14]. Finally, the angle $\theta = r_y$, where r_y depicts the rotation over the y axis in camera coordinates (as illustrated in Fig. 4.1).

The generated ground truth heatmaps have a resolution of 152×152, i.e., a resolution four times smaller than the input heatmap resolution of 608×608. Comparing lower resolution heatmaps, in the loss function, lead to smaller loss values for the heatmap loss, and, consequently, overall better loss convergence. During the training stage, the input ground truth heatmaps were upsampled using nearest-neighbor interpolation, in order to keep the centroid's peak value equal to 1. At test time, the predicted heatmaps are also upsampled using nearest-neighbor interpolation, for the same reason.

As illustrated in Fig. 4.10, the network predicts a heatmap directly from the output feature map. This is achieved by downsampling the number of channels via convolution operations and upsampling the spatial dimensions using bilinear interpolation. Bilinear interpolation upsampling keeps fine-grained features better due to its smaller sampling step compared to Nearest Neighbor interpolation which would lose fine-grained features from the output feature map, consequently, producing a worse prediction for the heatmap. Finally, the heatmap is fed to a sigmoid activation function to bound the values in the range $[0, 1]$.

4.4.2 Loss Function

The goal of the 3D Object Detection and Tracking module is to predict (i) bounding boxes, (ii) their displacements between consecutive frames, and (iii) a heatmap with the objects' centroids location. Therefore, the loss function is given by

$$L = L_{\text{YOLOv4}} + L_{\text{Euuler}} + L_{\text{Offsets}} + L_{\text{Heatmap}} \tag{4.7}$$

where L_{YOLOv4} is given by (3.12), L_{Euler} is given by (3.8),

$$L_{\text{Offsets}} = \frac{1}{N} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(dx_{ij} - \hat{dx}_{ij})^2 + (dy_{ij} - \hat{dy}_{ij})^2 \right] \tag{4.8}$$

is similarly computed to the localization loss in (3.11). L_{Offsets} consists of the MSE loss for every responsible bounding box (as detailed in Sec. 3.1.1), and

$$L_{\text{Heatmap}} = -\lambda_{hm}\frac{1}{M}\sum_{x=1}^{X}\sum_{y=1}^{Y}\sum_{c=1}^{C}\begin{cases}(1-\hat{H}_{xyc})^{\alpha}\log(\hat{H}_{xyc}) & \text{if } H_{xyc}=1\\ (1-H_{xyc})^{\beta}(\hat{H}_{xyc})^{\alpha}\log(1-\hat{H}_{xyc}) & \text{otherwise}\end{cases} \tag{4.9}$$

depicts the loss function for the predicted heatmap. It consists of an adapted focal loss function, introduced in a CenterTrack's predecessor [14], with parameters $\alpha = 2$ and $\beta = 4$. It iterates over each cell of every channel in the predicted heatmap, with $C = 4$ classes and $X = Y = 152$ pixels/cells. M denotes the number of objects in the current frame, and $\lambda_{hm} = 0.5$ is responsible for decreasing the weight of the heatmap loss value. The minus sign, in the beginning, is to invert the logarithmic growth. Fig. 4.13 plots the loss values against the predicted value $\hat{H}_{xyc}$, for several examples of ground truth values H_{xyc}.

4.4.3 Velocity Vector and Track ID

The predicted offset (dx, dy) for a given object consists of its displacement relative to the ego-vehicle. Knowing the sampling frame rate is $f = 10Hz$ $(T = 0.1s)$, the velocity of an object relative to the ego-vehicle is given by

$$\vec{v}_{\text{object\_ego}} = \begin{bmatrix}\frac{dx}{T}\\ \frac{dy}{T}\end{bmatrix}. \tag{4.10}$$

Applying the relative velocity equation

$$\vec{v}_{\text{object\_ego}} = \vec{v}_{\text{object}} - \vec{v}_{\text{ego}}, \tag{4.11}$$

and using the ego-vehicle velocity (given by the IMU depicted in Fig. 4.1)

$$\vec{v}_{\text{ego}} = \begin{bmatrix}v_{\text{ego\_right}}\\ v_{\text{ego\_forward}}\end{bmatrix}, \tag{4.12}$$

we can compute the velocity of an object

$$\vec{v}_{\text{object}} = \vec{v}_{\text{object\_ego}} + \vec{v}_{\text{ego}} = \begin{bmatrix}\frac{dx}{T}\\ \frac{dy}{T}\end{bmatrix} + \begin{bmatrix}v_{\text{ego\_right}}\\ v_{\text{ego\_forward}}\end{bmatrix}. \tag{4.13}$$

As detailed in Sec. 4.3.2, a pixel in the BEV RGB map corresponds to $0.08m$. Given this ratio, the predicted velocities of the detected objects can be drawn onto the BEV RGB map, as shown in Fig. 4.14.

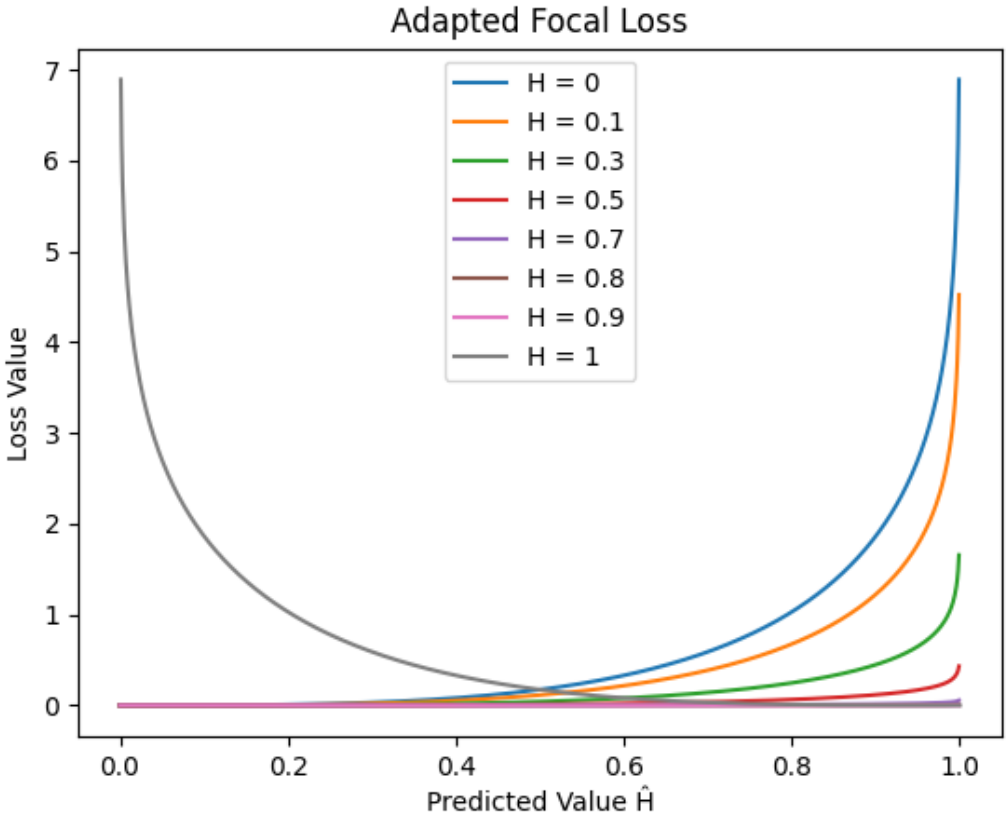


Figure 4.13: Adapted Focal Loss plot. For a given cell of the heatmap, the different curves, depicting variable ground truth values, plot the loss value against the predicted value in that cell of the heatmap. It is visible that a high penalty is given when a ground truth peak ($H = 1$) is wrongly predicted and a low ground truth value is predicted as a peak or a neighboring point to the peak. However, any predicted values near the peaks ($0.7 < H < 0.9$) are barely and equally penalized for either a right or wrong prediction.

The track ID's are obtained by matching bounding boxes between frames. The matching is done using Euclidean distance between centroids. However, before comparing distances, the predicted offsets are subtracted for each object before computing the distance to the centroids' location in the previous frame.

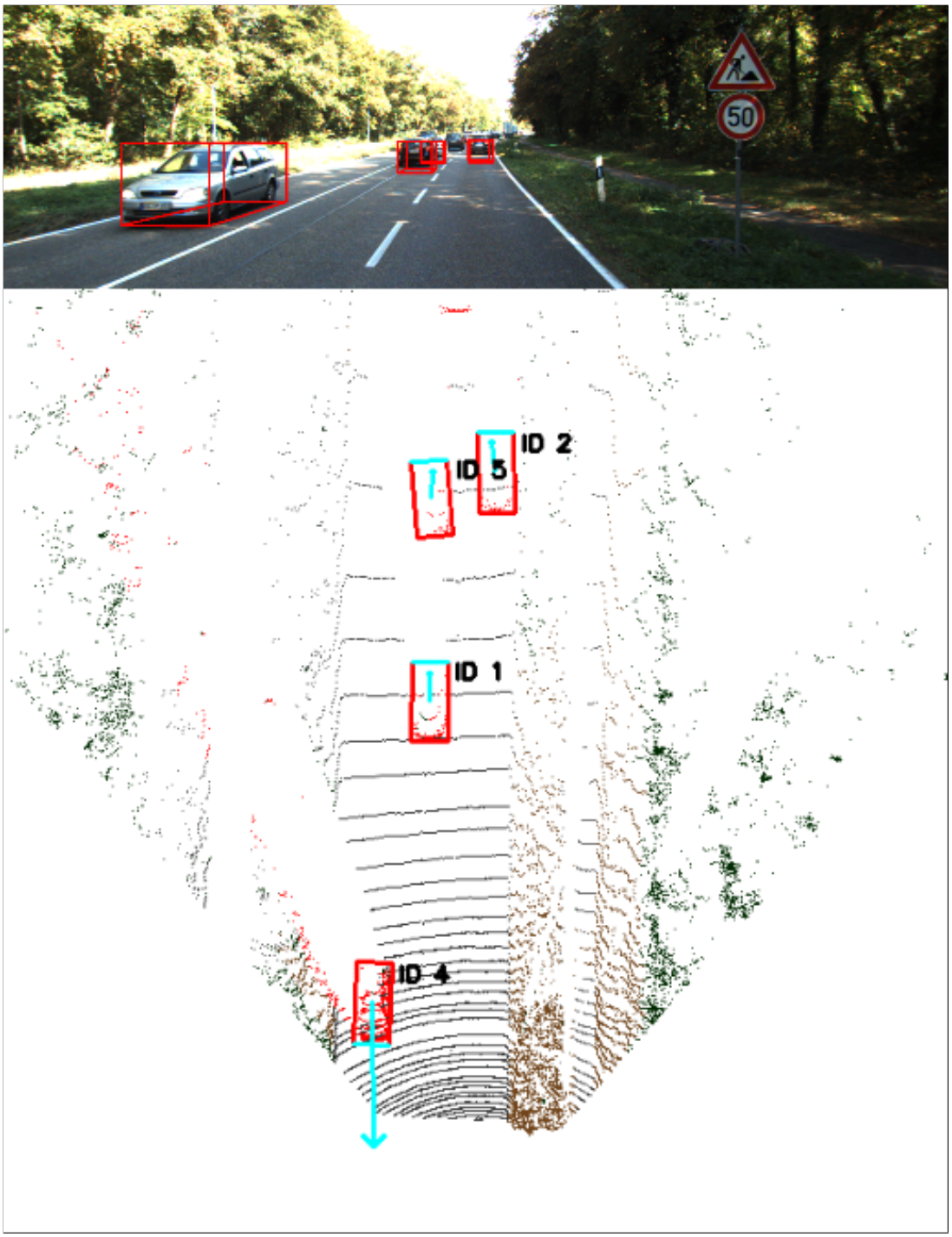


Figure 4.14: Example of the output predicting the objects' bounding boxes, track ID's, and velocity. The predictions are uniquely based on the Semantic BEV RGB map. The RGB image by camera 2 is for illustration purposes only.

Chapter 5

Experimental Results

In this chapter, the metrics needed for the evaluation of an object detector are detailed. Then, the detection results obtained for the 3D Object Detection module are presented. Lastly, the results obtained for the 3D Object Detection and Tracking module are detailed along with empirical results that justify design choices and an ablation study for the model architecture. A final comparison between our model and the baseline model, Complexer-YOLO, is also performed.

5.1 Metrics

Intersection over Union (IoU) between a predicted bounding box (B_{pred}) and the ground truth box (B_{truth}) is given by

$$IoU = \frac{\text{area}(B_{\text{pred}} \cap B_{\text{truth}})}{\text{area}(B_{\text{pred}} \cup B_{\text{truth}})}, \tag{5.1}$$

thus, $0 \leq IoU \leq 1$. Computing the IoU between the predicted and ground truth boxes gives us a metric for how good the overlapping between the two boxes is.

The confidence score predicted by the 3D object detector (3.2) depicts a probability that a given predicted box contains an object.

Predicting a bounding box for a given object implies predicting its (i) location, (ii) dimensions, (iii) confidence score, and (iv) class of the object. A detection is considered a True Positive (TP) if it satisfies three conditions: (i) the predicted confidence score is above a given threshold (e.g., $\text{Conf}_{\text{thresh}} = 0.5$), (ii) the predicted bounding box has an IoU greater than a given threshold (e.g., $IoU_{\text{thresh}} = 0.5$) with the ground truth,

(iii) the predicted class matches the ground truth class. Failing either of the latter two conditions makes the detection a False Positive (FP). A detection is considered a False Negative (FN) when the confidence score of a given box that is supposed to detect a ground truth object is lower than a given threshold (e.g., $\text{Conf}_{\text{thresh}} = 0.5$). A True Negative ($TN$) detection occurs when a predict box whose confidence score is below a given threshold and there's no object captured inside the predicted bounding box. The TN metric is irrelevant in the object detection task.

The precision metric is given by

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{5.2}$$

It represents the ratio between correctly predicted positives and the total number of predicted positives. For example, if there's a total of 120 ground truth objects to detect, the model makes 100 positive predictions, and 20 of those predicted positives are incorrect. This example returns a precision of 80%. Should the number of ground truth objects be 4000, the precision value would still be 80% because the precision metric does not account for the total number of existing objects in the ground truth and should be detected. Another metric needs to be used to give us a better understanding of a model's performance. The recall metric is given by

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{5.3}$$

It represents the ratio between correctly predicted positives and the total number of ground truth positives. For the example given above, when there are 120 ground-truth objects to detect, recall equals 67%. When there are 4000 ground-truth objects to detect, the recall value is equal to 2%. The recall metric gives us a better look at the model's capability of detecting all objects in the data.

Setting the confidence threshold at different levels results in varying values for the pair precision and recall. Plotting the precision values against the recall values while varying the confidence threshold results in the precision-recall curve. Comparing the precision-recall curves of two object detection models can be tricky when the curves intersect each other. Therefore, a better way to compare these models is by comparing the area under the precision-recall curve, which gives us the Average Precision (AP) metric.

Average Precision is computed for each class. Object detectors usually detect multiple classes. The mean Average Precision (mAP) metric is given by

$$mAP = \frac{1}{C} \sum_{i=1}^{C} AP_i \tag{5.4}$$

where C depicts the number of classes. It represents the arithmetic mean of the Average Precision values across all C classes. F1 score is given by

$$\text{F1\_score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.5}$$

and it depicts a weighted average between precision and recall values. It equally weights precision and recall values. However, for object detection recall is usually given more emphasis than precision, thus, the AP metric is more relevant than the $F1$ score.

5.2 Training and Hyperparameters

All models were trained for 300 epochs on an NVIDIA GeForce GTX 1070, with a batch size equal to 2, using the Adam optimizer. A cosine learning rate decay is applied and its initial value is 0.001, as illustrated in Fig. 5.1. Other hyperparameters include the weights associated to the loss functions (3.11) and (4.9).

5.3 3D Object Detection Module

In this section, the stock Complex-YOLOv4 is compared to the different modified versions that were tested before reaching the final 3D Object Detection module, which is described in Sec. 4.3.

Initially, the 3D object detection Complex-YOLOv4 was only altered to accept Semantic BEV RGB maps instead of the handcrafted features encoded in the RGB map of stock Complex-YOLOv4. Then, as shown in Fig 4.9, we added the Truck class as an extra class to be detected. Finally, the FoV filtering was applied to filter out ground truth objects whose locations were outside the camera FoV. If the objects live outside the FoV they do not have semantic features, therefore, do not appear in the BEV RGB map. Before the FoV filtering, the network was penalized for not predicting objects not visible in the BEV RGB map, i.e., feature-less objects. This change was promoted

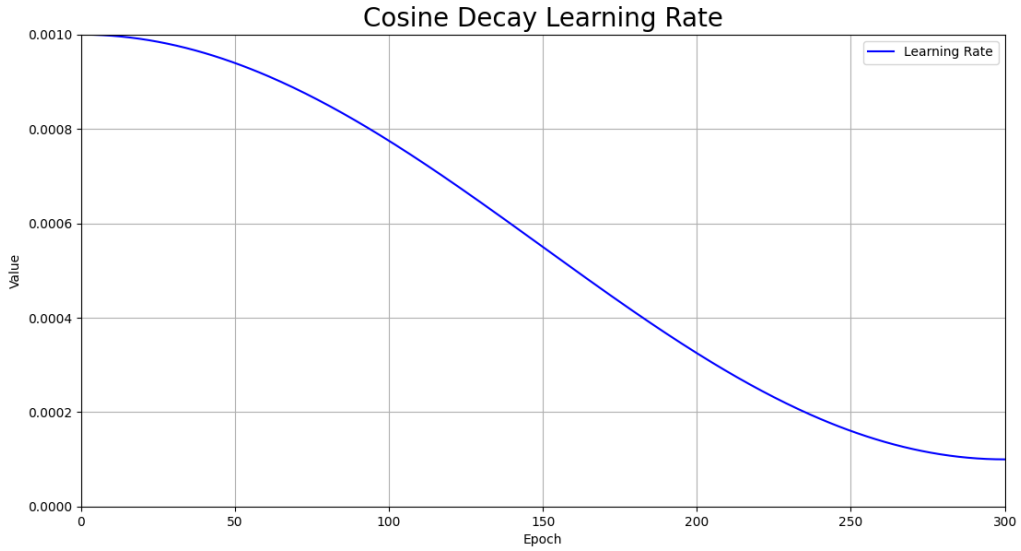


Figure 5.1: Cosine learning rate decay starting at 0.001 and decaying throughout the 300 epochs of the training stage.

to both the 3-class and 4-class models, and in both cases, it increased the mAP. The experimental results are detailed in Table 5.1.

Average Precision (AP) values for each class and model are depicted in Fig. 5.2. It is visible that the overall better-detected class is the Car class, for any of the tested object detection modules. This occurs due to the imbalanced class distribution (see Table 4.2) present in the 3D Object Detection of the KITTI dataset. The Car class (Car and Van together) make up for over 80% of the available ground truth objects, while Pedestrians, Cyclists, and Trucks comprise 11%, 4%, and 2% of the labeled objects, respectively. Eventhough there are more Pedestrian than Cyclist and Truck instances combined in the dataset, the tested 3D object detection models yielded better results for Cyclist and Truck detection than for Pedestrian detection. This happens because the size of the objects also plays a role in the model's performance at detecting them. In the point clouds, the Pedestrian object instances comprise the least amount of points among the four detectable classes.

Although the stock Complex-YOLOv4 model achieves the highest mAP (88.77%) out of all the tested models, unlike Complex-YOLOv4, our method does not use handcrafted features to build the BEV RGB maps. Instead, it exploits semantic information to obtain

Table 5.1: For every model, the thresholds detailed in Sec. 5.1 are set to $Conf_{thresh} = 0.5$ and $IoU_{thresh} = 0.5$.

Model	Input	Class	Precision↑	Recall↑	AP↑	F1↑	mAP↑
Complex YOLOv4 (3 classes)	Handcraft	Car	**91.15**	97.53	**96.88**	**94.23**	**88.77**
		Pedestr.	69.61	93.06	78.48	79.64	
		Cyclist	80.00	93.77	90.96	86.34	
		Truck	-	-	-	-	
Ours (3 classes)	Semantic	Car	85.50	96.52	95.47	90.68	83.47
		Pedestr.	56.45	90.16	71.24	69.43	
		Cyclist	61.20	93.04	83.71	73.84	
		Truck	-	-	-	-	
Ours (3 classes FoV)	Semantic	Car	85.53	**97.74**	96.66	91.23	85.65
		Pedestr.	51.92	92.66	72.10	66.55	
		Cyclist	59.71	93.82	88.18	72.97	
		Truck	-	-	-	-	
Ours (4 classes)	Semantic	Car	**88.27**	97.15	96.43	**92.50**	84.91
		Pedestr.	65.35	78.59	67.06	71.36	
		Cyclist	65.86	89.74	85.83	75.97	
		Truck	73.83	92.44	90.31	82.09	
Ours (4 classes FoV)	Semantic	Car	87.95	**97.35**	**96.45**	92.41	**85.52**
		Pedestr.	56.56	86.82	68.11	68.50	
		Cyclist	65.51	94.59	89.41	77.41	
		Truck	70.00	91.59	88.09	79.35	

the BEV RGB maps and achieves an on-par mAP result (85.52%).

5.4 3D Object Detection and Tracking Module

In this section, empirical results that dictated the choice of the interpolation method used to upsample the heatmaps are presented, as well as empirical results that validate the chosen loss function and respective weight tuning. Finally, the results obtained from the conducted ablation study for the 3D Object Detection and Tracking module are presented and discussed.

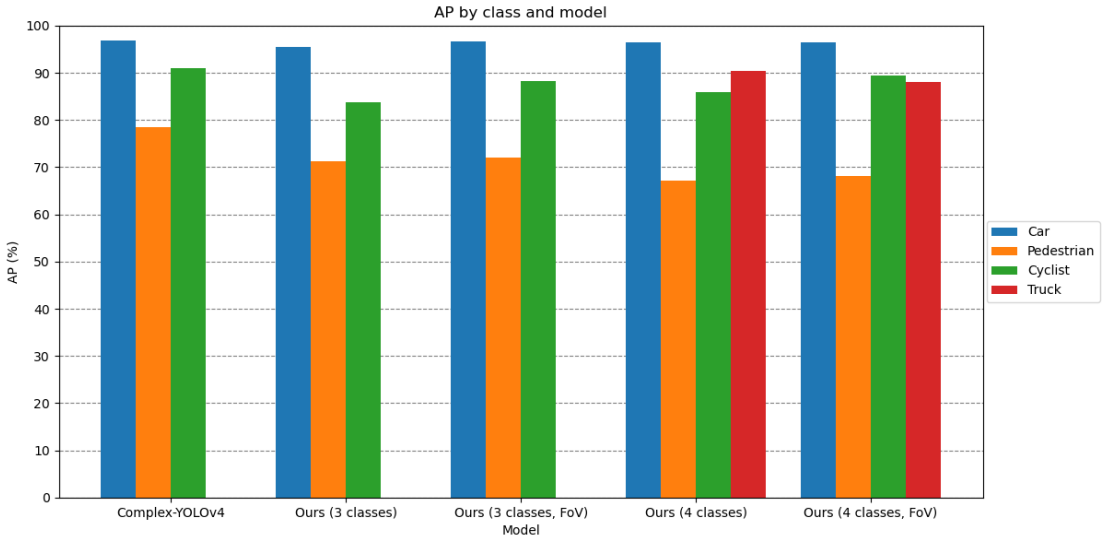


Figure 5.2: AP per class per 3D Object Detection Model.

5.4.1 Heatmap Generation Across Scales

As illustrated in Fig. 4.10, the bounding boxes and respective offsets are predicted across every scale. Then, from a large number of predictions, only a few are selected, to display detections, by the NMS algorithm. Following the same principle, we tested heatmap prediction across the three output scales. Through empirical observation, it was noted that the deeper the scale, the better the heatmap prediction was. This follows the underlying logic of neural networks that higher-level features live in the deepest layers of a network.

5.4.2 Heatmap Loss Function and Tuning

Initially, the heatmap loss function was implemented as the MSE loss for each cell in the output heatmap. Since the heatmaps are mostly comprised of background pixels whose value is close to 0 and every pixel of the heatmap is bounded between 0 and 1, the MSE loss leads to the heatmap loss decreasing rapidly and establishing itself in a divergent cycle of nearly zero values, as illustrated in Fig. 5.3.

To try to fix this issue, a higher weight value was attributed to the heatmap loss by multiplying its value by a factor λ_{hm}. Several values were tested, and through empirical

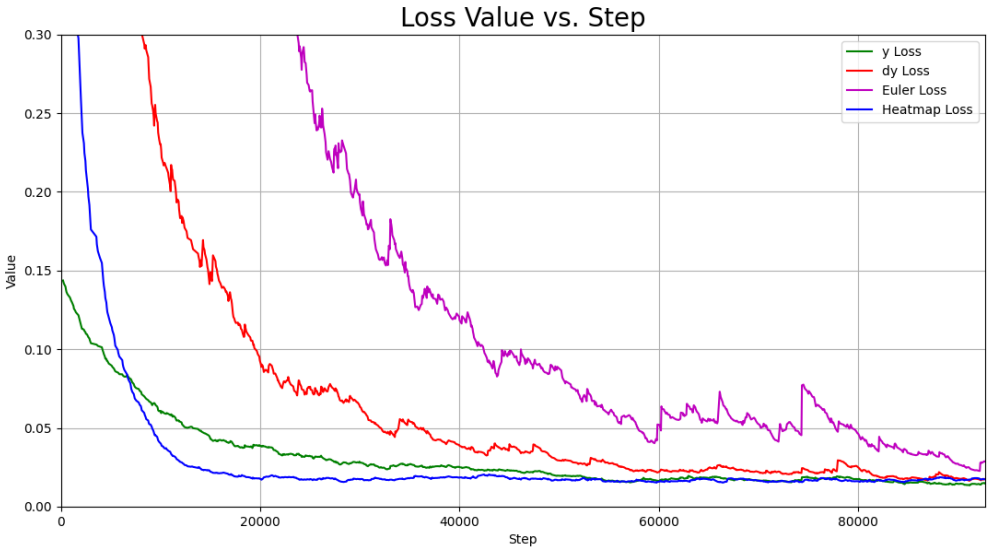


Figure 5.3: Heatmap MSE loss diverging (blue curve). The loss plummets and does not converge afterwards, thus, the training was stopped at the 30^{th} epoch.

observation, it was noted that changing this value would result in good early convergence for the heatmap loss curve. However, it only delayed the divergent course for the heatmap MSE loss curve. Identical results were obtained by changing the loss function to the L1 loss. Finally, the adapted focal loss proposed in [14] was implemented, tested, and resulted in convergence for the heatmap loss. Contrary to the MSE loss, which penalized errors equally, the focal loss penalized predictions more heavily when the predicted and ground truth values were far apart (see Fig. 4.13). Although converging, the loss values for the heatmap loss were still too high compared to the other losses. Ideally, the loss curves should converge at about the same rate. Therefore, the factor λ_{hm} was set to 0.5. Thus, decreasing the weight given to the heatmap loss function. Fig. 5.4 depicts the loss curves converging as intended.

As detailed in Sec. 4.4.1, computing the loss function for each cell of a high-resolution heatmap leads to high loss values for the heatmap loss. Consequently, a dataset of low resolution (4 times smaller) ground truth heatmaps was generated to compare the output heatmap with the target heatmap directly. This lead to a decrease by a factor of 16 of the heatmap loss value since the square-shaped heatmaps decreased both their spatial dimensions by a factor of 4.

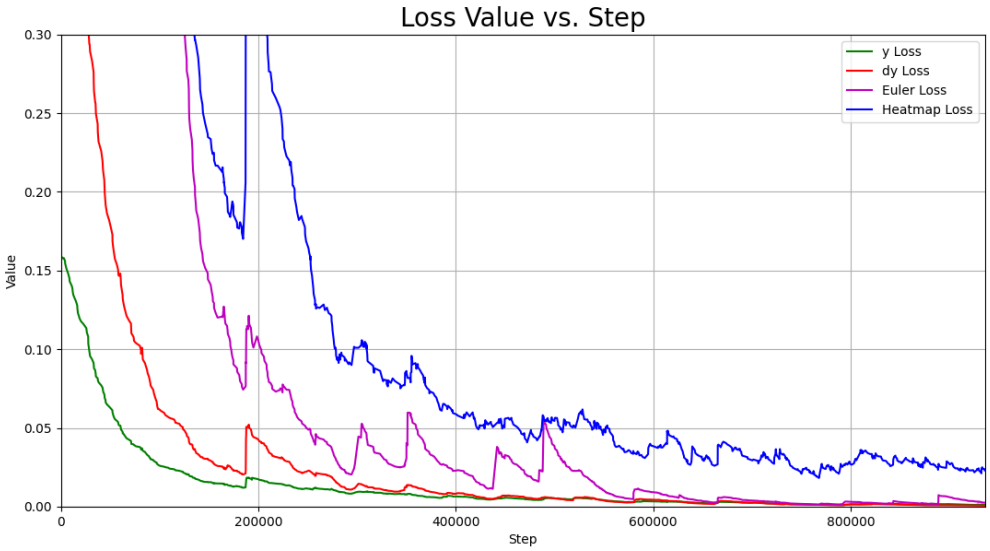


Figure 5.4: Heatmap adapted focal loss converging (blue curve).

5.4.3 Heatmap Upsampling

As explained in Sec. 5.4.2, the predicted heatmaps output by the network have a low resolution of 152×152. Therefore, they must be upsampled to the input heatmap resolution of 608×608, to close the feedback loop present during the testing phase. The generated ground truth heatmaps also have a resolution of 152×152 to prevent the need for downsampling them when computing the heatmap loss function between the target and the predicted heatmap. Therefore, both of these heatmaps (the target during training and the predicted during testing) need to be upsampled to match the BEV RGB maps resolution of 608×608 before concatenating with them. Through empirical observation, the nearest-neighbor interpolation method was chosen for these upsampling steps since it kept the heatmap peaks (object' centroids whose value is 1 on the heatmap). Keeping these peaks during the training stage is crucial because the adapted focal loss applied to the heatmap loss needs to have these peaks where the target heatmap is 1, to correctly compute the loss and, consequently, correctly predict the heatmaps.

As depicted in Fig. 4.10, the output feature map of scale 3 has its spatial dimensions upsampled and the number of channels downsampled to reach the desired output

(a) Nearest Neighbor Interpolation (b) Bilinear Interpolation

Figure 5.5: Comparison between (a) nearest-neighbor interpolation and (b) bilinear interpolation of the output feature map, from scale 3, to obtain the predicted heatmap.

heatmap shape of $4 \times 608 \times 608$. Applying the coarser nearest-neighbor interpolation to the output feature maps would render the loss of features, whereas the application of the finer bilinear interpolation to the output feature maps leads to maintaining the important features needed to mimic the gaussian blobs of the target heatmaps. Through empirical observation, bilinear interpolation leads to better heatmap prediction, as shown in Fig. 5.5.

5.4.4 Train-Test Split

The 3D Multi-object Tracking branch from the KITTI dataset (detailed in Sec. 4.1) contains 21 labeled sequences of frames. These sequences can be divided into four different categories, according to the environment where they were obtained. These categories include (i) urban, (ii) fast-lane, (iii) highway, and (iv) pedestrian zones.

The train-test split was performed following the rule of thumb that 80% of the samples should be used for training and the remaining 20% for testing. This results in 17 sequences used for training and 4 sequences used for testing purposes. Besides the 80/20 split, each of the four sequences selected for the testing stage belongs to the four

Table 5.2: Class distribution of the test split from the 3D Multi-Object Tracking Dataset.

Class	**Test Split Class Distribution (%)**
Car	81.52
Van	3.58
Truck	0.59
Cyclist	3.71
Pedestrian	10.60
Person Sitting	0.00
Tram	0.00

environment categories stated above. The resulting class distribution for the test split is detailed in Table 5.2.

5.4.5 Ablation Study

Initially, a dataset of elliptical style heatmaps was generated to train the network within the 3D Object Detection and Tracking module depicted in Fig. 4.10. The ablation study conducted came to be because the heatmap prediction, from this model, was sub-par. An illustrative experimental result is depicted in Fig. 5.6. It is visible that the network predicts the objects' centroids. However, it failed to predict the spread of the gaussian blob, dictated by the standard deviation parameter. The feedback loop was broken and the ground truth heatmap was fed to the network to test whether the poor location and offset predictions were caused by the poor heatmap prediction (which would feed the network on the next iteration). An illustrative result for this experiment is depicted in Fig. 5.7. This experiment confirmed that (i) the network could not predict the spread given by the centroids' standard deviation parameter and, consequently, (ii) the poor location and offset predictions were caused by the poor input heatmap coming from the feedback loop.

The next logical step was taken, a dataset of circular style heatmaps was generated. The circular centroids within these heatmaps have a reduced standard deviation value which decreases the spread of the gaussian blobs. The network was trained using this heatmap style. An illustrative result, obtained using the feedback loop to feed the predicted maps to the network, is depicted in Fig. 5.8. Just from observation, it is visible that (i) the predicted heatmaps, that go through the feedback loop, are quite

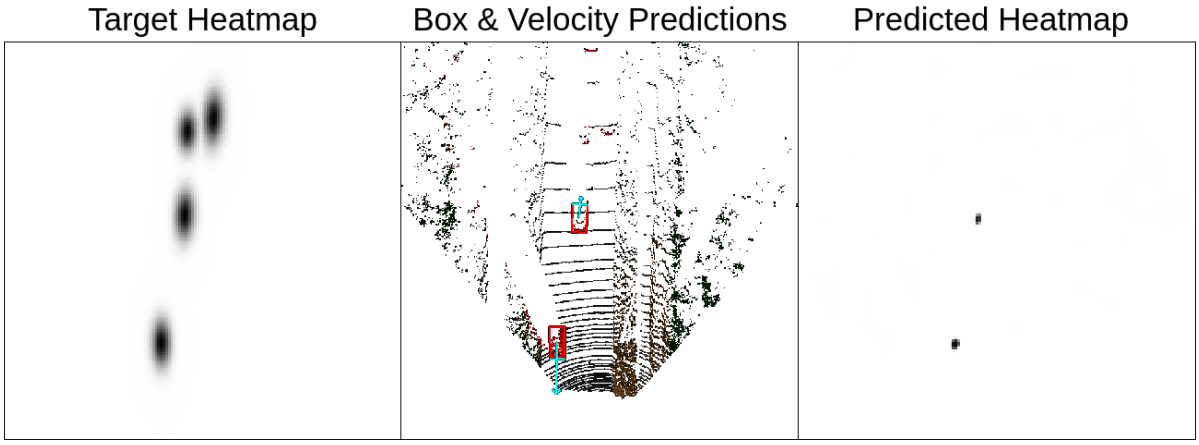


Figure 5.6: Elliptical heatmap model with predicted heatmap on the input.

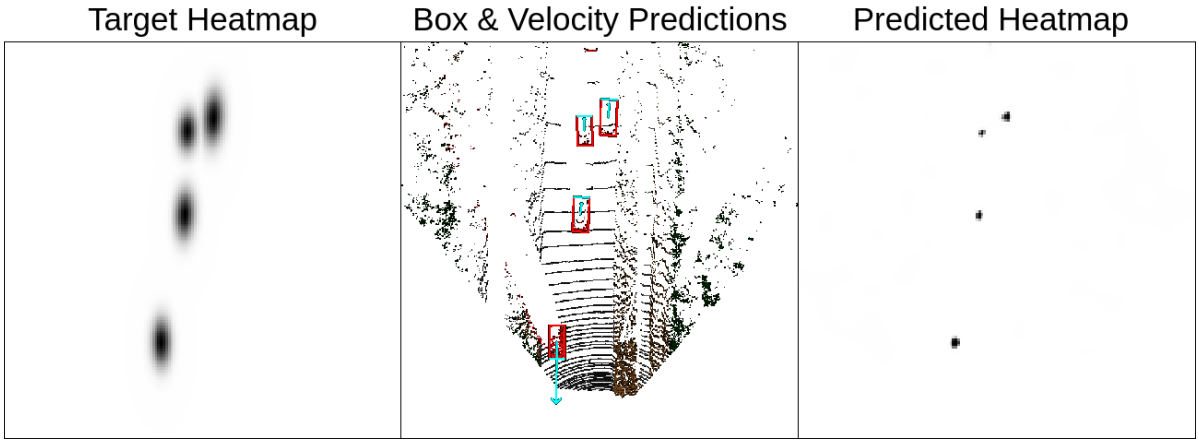


Figure 5.7: Elliptical heatmap model with ground truth heatmap on the input.

similar to the ground truth (target) heatmaps and, consequently, (ii) more objects are detected, using this type of heatmaps. Another experiment was conducted, in which the ground truth circular heatmap was fed to the network. An illustrative result is depicted in Fig. 5.9.

A final heatmap experiment was conducted in which the input heatmap at instant t is obtained via a post-processing of the heatmap predicted at $t-1$. Instead of directly injecting the predicted heatmap in the following instant, it injected a new heatmap generated from the predicted centroids in the heatmap and bounding box dimensions. It increased the mAP value for the elliptical style heatmap models, since it feeds the network a true elliptical style heatmap. However, many centroids are still missing in

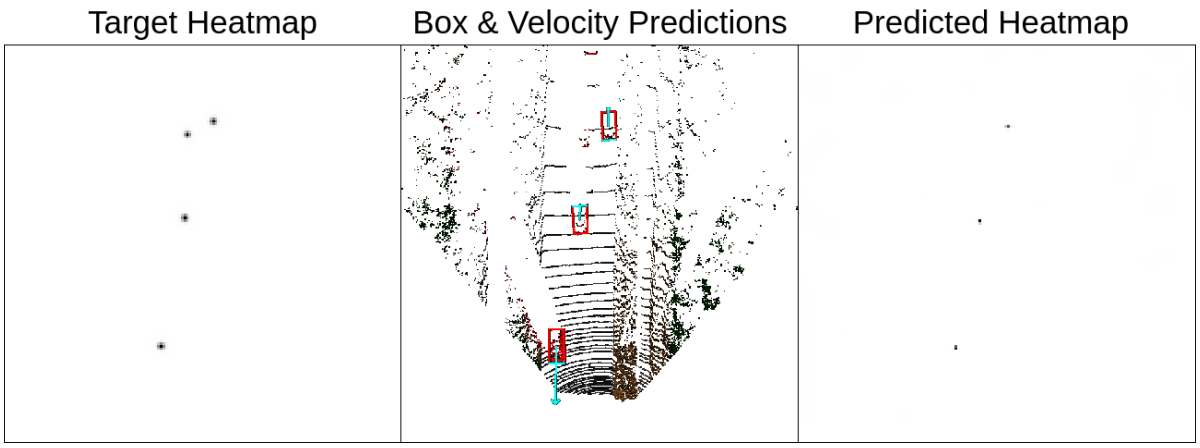


Figure 5.8: Circular heatmap model with predicted heatmap on the input.

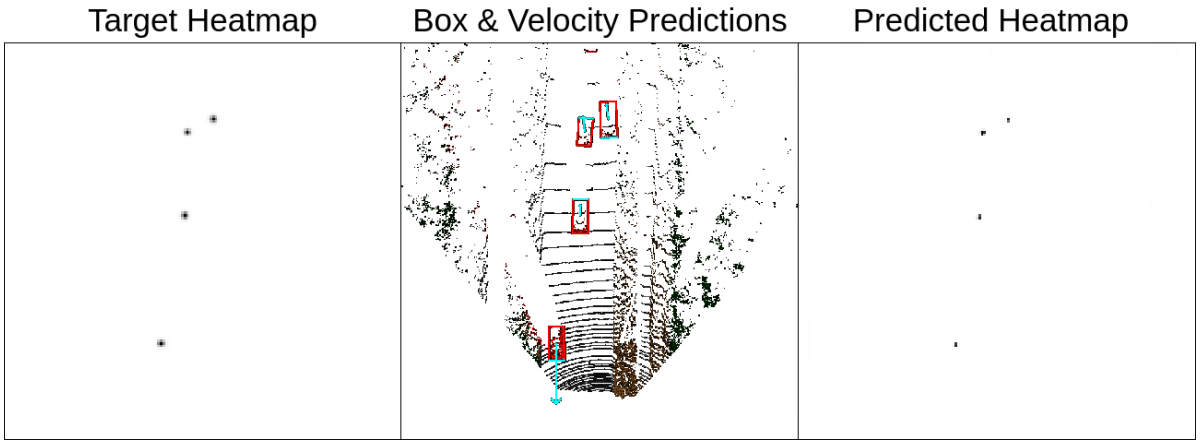


Figure 5.9: Circular heatmap model with ground truth heatmap on the input.

the predicted heatmap, thus, they are not present on the post-processing heatmap, consequently, degrading the object detection performance of the model. In the case of the post-processed heatmap for the circular heatmap model, the mAP yielded a similarly poor result to the direct feeding of heatmaps without post-processing.

Lastly, the heatmaps were completely removed from the 3D Object Detection and Tracking module. Therefore, the location and offset predictions were based on the semantic BEV RGB maps from instants t and $t-1$ only.

The results obtained for the three different models are detailed in Table 5.3. The domination of the Car class in the dataset makes every model perform significantly better for this class. The poor detection of the other classes is explained by the class

distribution as well (see Table 4.2). This lowers the mAP value significantly since it averages the AP value across all classes.

Feeding the heatmap models with the ground truth heatmap leads to a significant improvement of the recall value, i.e., more and more occluded objects are being detected correctly. This demonstrates that the features from the heatmap influence the detection and offset prediction. Moreover, the elliptical model fares better than the circular model when both take the ground truth heatmap as input ($mAP = 67.60\%$ vs. $mAP = 46.07\%$). This happens because the elliptical model comprises heatmaps whose centroids mimic the orientation and shape of the bonding boxes, thus, making the bounding box regression task easier.

The heatmap feedback improves the detection by feeding the network the location of the objects in the previous frame. However, the best performing model was the no-heatmap model ($mAP = 28.38\%$) because the network cannot predict heatmaps correctly, therefore, decreasing the performance of the heatmap-based models.

The confusion matrices for the different models are depicted in Fig. 5.10. The low recall values, caused by the small number of correct detections, are reflected in the confusion matrices. The majority of the objects are not detected at all, i.e., the predicted label is non-existent when there should be an object detected. The results detailed in Table 5.3, for the poorly detected Cyclist class, are visible in the confusion matrix. When the ground truth label is Cyclist, the detection is not achieved. Moreover, from the confusion matrices, it is visible that some Truck objects are detected as belonging to the Car class.

5.5 Complexer-YOLO vs. Ours

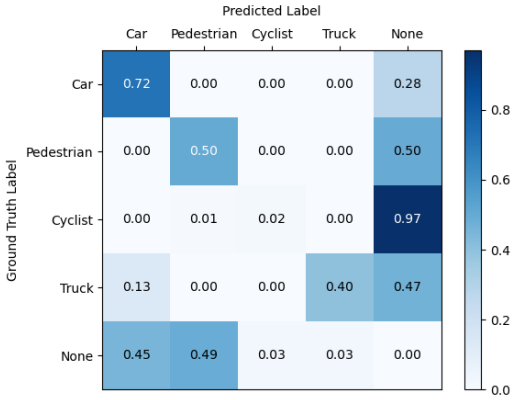
As a final evaluation, we are comparing our method to the baseline method, Complexer-YOLO. On one hand, Complexer-YOLO (see Fig. 2.7) voxelizes the semantic point cloud, which is computationally expensive. Instead, we project the semantic point cloud to the ground plane to feed it to the CNN. On the other hand, Complexer-YOLO uses a non-deep learning approach to the tracking method, whereas our tracking method is based on a deep learning approach which allows for an end-to-end model capable of joint 3D object detection and tracking. This results in a faster joint object detection and tracking module, as explained below.

Table 5.3: Only 4% of labelled objects belong to the Cyclist class (see Table 4.2). The resulting test split (detailed in Sec. 5.4.4), used to evaluate these models, contains only one cyclist across the four test sequences. All heatmap related tests were conducted by initializing the feedback loop with the ground truth heatmap of the very first frame in the sequence.

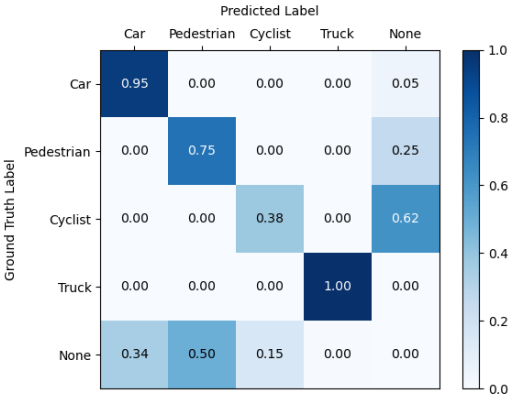
Model	**Input**	**Class**	**Precision↑**	**Recall↑**	**AP↑**	**F1↑**	**mAP↑**
No Heatmap	BEV (t) BEV(t-1)	Car	90.95	72.27	71.03	80.54	28.38
		Pedestrian	53.05	50.36	30.61	51.67	
		*Cyclist	16.00	1.89	1.65	3.38	
		Truck	22.22	40.00	10.22	28.57	
Elliptical Heatmap	BEV (t) BEV (t-1) H_{GT} (t-1)	Car	94.47	95.01	94.01	94.99	67.60
		Pedestrian	63.88	74.58	54.52	68.82	
		*Cyclist	42.78	37.74	24.39	40.10	
		Truck	83.33	100.00	97.50	90.91	
	BEV (t) BEV (t-1) H_{Pred} (t-1)	Car	92.55	13.79	13.48	24.01	4.69
		Pedestrian	45.45	1.81	1.14	3.48	
		*Cyclist	0.00	0.00	0.00	0.00	
		Truck	28.57	13.33	4.13	18.18	
	BEV (t) BEV (t-1) H_{Post} (t-1)	Car	76.56	25.74	22.51	38.53	10.41
		Pedestrian	45.43	19.76	9.92	27.54	
		*Cyclist	28.57	0.94	0.37	1.83	
		Truck	14.29	20.00	8.83	16.67	
Circular Heatmap	BEV (t) BEV (t-1) H_{GT} (t-1)	Car	87.22	92.81	90.47	89.93	46.07
		Pedestrian	53.84	66.75	38.69	59.60	
		*Cyclist	26.04	32.55	15.43	28.93	
		Truck	47.62	66.67	39.69	55.56	
	BEV (t) BEV (t-1) H_{Pred} (t-1)	Car	89.62	54.1	52.58	67.47	15.56
		Pedestrian	10.30	17.23	3.01	12.89	
		*Cyclist	0.00	0.00	0.00	0.00	
		Truck	100.00	6.67	6.67	12.50	
	BEV (t) BEV (t-1) H_{Post} (t-1)	Car	85.37	38.78	37.56	53.33	13.43
		Pedestrian	59.32	16.87	13.48	26.27	
		*Cyclist	100.00	0.47	0.47	0.94	
		Truck	25.00	6.67	2.22	10.53	

Complexer-YOLO runs the real-time ENet for semantic segmentation at 90 FPS. We applied the MSeg model since it was the one readily available to deploy. Complexer-YOLO performs object detection (predicting on voxelized input) at 15 FPS and runs the tracking method at 100 FPS. Consequently, their bottleneck relies on the detection

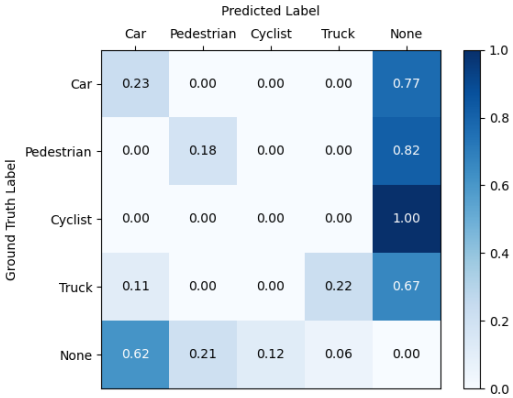
task, which added to the tracking task amounts to 13 FPS. Our method runs the proposed joint detection and tracking network at 18 FPS. Therefore, our method improves the bottleneck computational time of the Complexer-YOLO pipeline, while performing detection and tracking with an end-to-end trainable deep network.



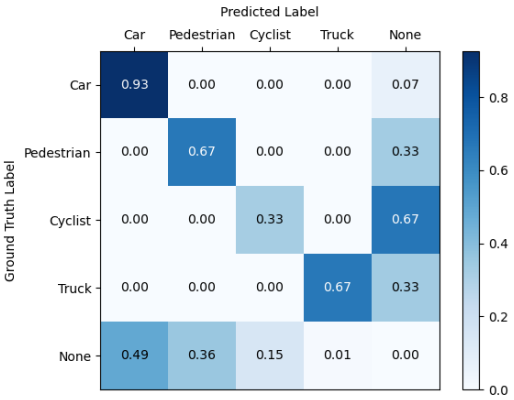
(a) No Heatmap



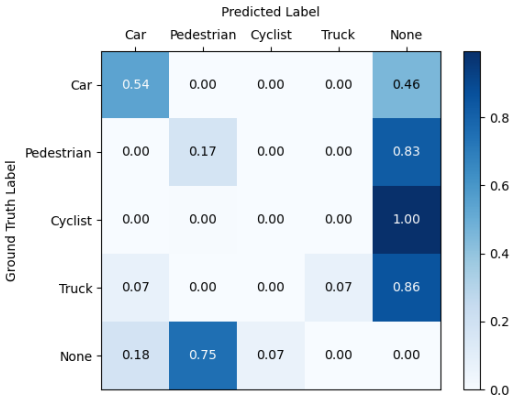
(b) Elliptical Heatmap with ground truth as input



(c) Elliptical Heatmap with prediction as input



(d) Circular Heatmap with ground truth as input



(e) Circular Heatmap with prediction as input

Figure 5.10: Confusion matrices from the ablation study conducted.

Chapter 6

Conclusion

In recent years, deep learning methods have surpassed classic methods in terms of accuracy and speed, especially when using powerful GPUs capable of massively parallel computations. The goal of this thesis consists of applying deep learning methods to perform semantic segmentation and tracking, in the road environment. Uses for this technology include obstacle avoidance, assisted braking, among other tasks that improve safety on the road. To this end, we based our method on the state-of-the-art YOLO object detector. Since we aimed to perform tracking as well, we followed the Complexer-YOLO method. However, we ditched the voxelization of the point cloud and replaced the classic tracking method with a deep learning approach, consequently, achieving a higher inference speed.

This work exploits the fusion of dense 2D semantic information with 3D geometric data to achieve robust object detection in the three-dimensional space. Then, the tracking paradigm from CenterTrack was embedded within the Complex-YOLO architecture to achieve a joint 3D object detection and tracking pipeline. One of the major contributions of this thesis is the proposed pipeline. The input now consists of a concatenation of two consecutive BEV frames of the point cloud and the predictions are made on three different scales, to improve the prediction of differently sized objects.

The KITTI dataset was used to train the deep learning method. Based on the raw data from KITTI, a dataset of heatmaps was generated to train the network to predict heatmaps depicting the objects' locations. The semantic information obtained in the 2D image was projected to the 3D point cloud, to obtain a 3D semantic point cloud that would become the input to the proposed pipeline. As a rule of thumb, the 80/20 split

was applied to divide the labeled dataset into train and test samples. The test samples were used to evaluate the model and were never seen during training.

Results demonstrated that input heatmaps, containing objects' locations in the previous frame, help the detection task because they explicitly offer a feature map that helps to detect more occluded objects. However, the achieved heatmap prediction presented poor results, which degraded the model's performance. Therefore, the heatmap absent model yielded overall better results. Comparing with the baseline method of Complexer-YOLO, our method improved their bottleneck computational time from 13 FPS to 18 FPS, by performing joint object detection and tracking in an end-to-end fashion using a deep neural network.

Future work can be done towards experimenting with other heatmap loss functions to improve the heatmap prediction. Another possible solution could be the integration of an Auto-Regressive model as a way to improve the results of heatmap prediction, when using this model. The network would be fed with more than just one past frame, for instance, $k = 5$ past frames, in order to obtain more consistent heatmaps. Lastly, data augmentation techniques could be used to counter the bad class distribution in the dataset, thus, improving the model generalization.

Bibliography

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 19, 28

[2] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 1907–1915, 2017. 13

[3] F. Engelmann, T. Kontogianni, J. Schult, and B. Leibe. Know what your neighbors do: 3d semantic segmentation of point clouds. In *European Conf. Computer Vision (ECCV)*, pages 0–0, 2018. 10, 35

[4] O. Faugeras. *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993. 34

[5] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research (IJRR)*, 32(11):1231–1237, 2013. 33

[6] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012. 15, 31

[7] B. Graham, M. Engelcke, and L. van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 9224–9232, 2018. 8

[8] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. 29

[9] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017. 29

[10] M. Jiang, Y. Wu, T. Zhao, Z. Zhao, and C. Lu. Pointsift: A sift-like network module for 3d point cloud semantic segmentation. *arXiv preprint arXiv:1807.00652*, 2018. 5

[11] J. Lambert, Z. Liu, O. Sener, J. Hays, and V. Koltun. MSeg: A composite dataset for multi-domain semantic segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020. 1, 36

[12] L. Landrieu and M. Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 4558–4567, 2018. 5, 11, 12, 35

[13] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 12697–12705, 2019. 5, 15

[14] H. Law and J. Deng. Cornernet: Detecting objects as paired keypoints. In *European Conf. Computer Vision (ECCV)*, pages 734–750, 2018. 45, 46, 55

[15] F. J. Lawin, M. Danelljan, P. Tosteberg, G. Bhat, F. S. Khan, and M. Felsberg. Deep projective 3d semantic segmentation. In *International Conference on Computer Analysis of Images and Patterns*, pages 95–107. Springer, 2017. 7, 35

[16] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. In *Advances in neural information processing systems*, pages 820–830, 2018. 5, 11, 35

[17] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017. 14

[18] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 8759–8768, 2018. 29

[19] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. 7, 35

[20] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. Rangenet++: Fast and accurate lidar semantic segmentation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019. 7, 8

[21] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016. 17

[22] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 918–927, 2018. 13, 14

[23] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, July 2017. 5, 9, 10, 11, 14, 15, 35

[24] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017. 5

[25] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017. 19, 21

[26] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 26

[27] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 15

[28] S. Shi, X. Wang, and H. Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 770–779, 2019. 5

[29] M. Simon, K. Amende, A. Kraus, J. Honer, T. Samann, H. Kaulbersch, S. Milz, and H. Michael Gross. Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds. In *Proceedings of the IEEE Conf. Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 0–0, 2019. 1, 5, 17, 18, 19

[30] M. Simon, S. Milzy, K. Amendey, and H.-M. Gross. Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *Proceedings of the European Conf. Computer Vision (ECCV) Workshops*, 2018. iii, 1, 17, 19, 23, 24, 25, 37

[31] P. Sturm. *Pinhole Camera Model*, pages 610–613. Springer US, 2014. 34

[32] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou. Tangent convolutions for dense prediction in 3d. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 3887–3896, 2018. 8, 35

[33] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *2017 international conference on 3D vision (3DV)*, pages 537–547. IEEE, 2017. 8

[34] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *IEEE Int'l Conf. Computer Vision (ICCV)*, pages 6411–6420, 2019. 5

[35] Velodyne. What is lidar?, 2020. 6

[36] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE Conf. Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 390–391, 2020. 29

[37] Z. Wang and K. Jia. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. *arXiv preprint arXiv:1903.01864*, 2019. 5

[38] B. Yang, W. Luo, and R. Urtasun. Pixor: Real-time 3d object detection from point clouds. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 7652–7660, 2018. 14

[39] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia. Std: Sparse-to-dense 3d object detector for point cloud. In *IEEE Int'l Conf. Computer Vision (ICCV)*, pages 1951–1960, 2019. 12

[40] X. Zhou, V. Koltun, and P. Krähenbühl. Tracking objects as points. In *European Conf. Computer Vision (ECCV)*, pages 474–490, 2020. iii, 1, 16, 42

[41] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 4490–4499, 2018. 15

Appendix A

Full Proposed Pipeline

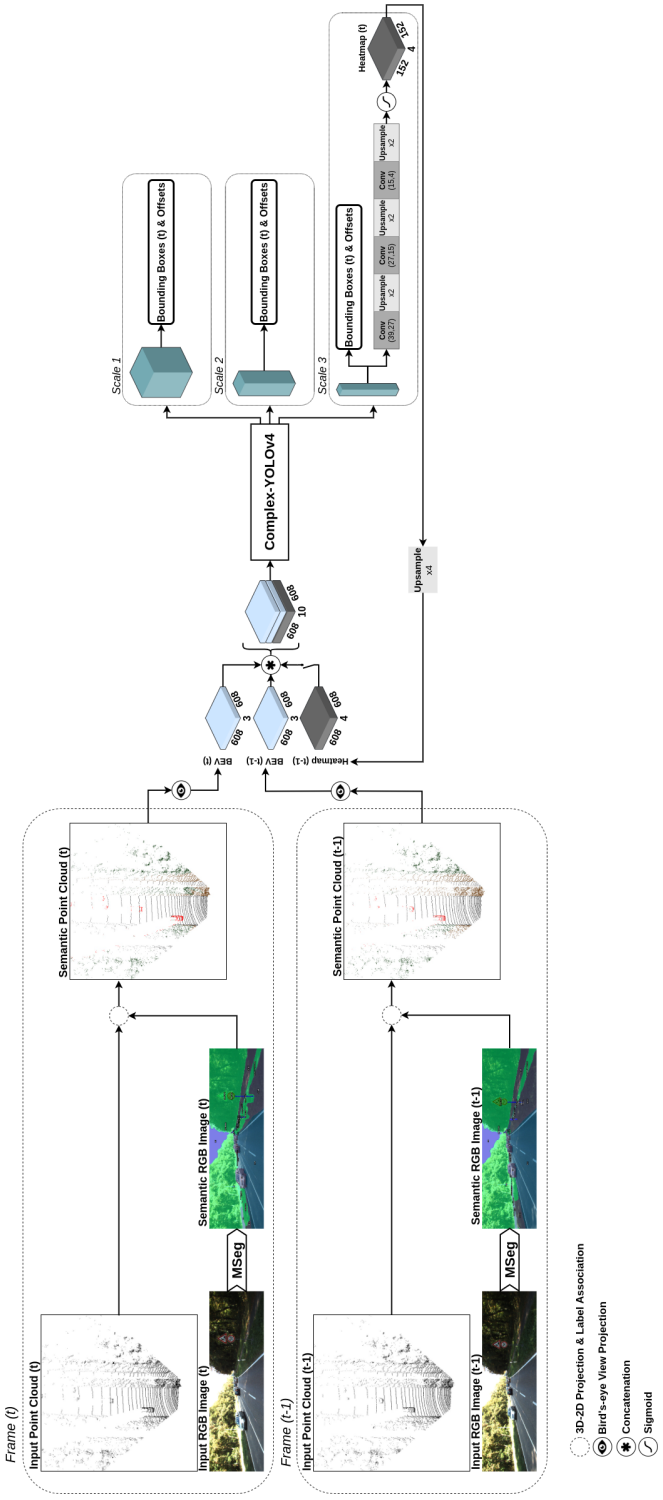


Figure A.1: Pipeline for the proposed 3D Object Detection and Tracking module.