DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Real-Time and Multi-Modal 3D Object Detection for Autonomous Driving

Xavier Diaz Ortiz

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Real-Time and Multi-Modal 3D Object Detection for Autonomous Driving

Echtzeit- und multimodale 3D Objekterkennung für autonomes Fahren

Author:	Xavier Diaz Ortiz
Supervisor:	Prof. Dr. Alois Knoll
Advisor:	M.Sc. Walter Zimmer
Submission Date:	15.10.2021

I confirm that this Master's Thesis in Robotics, Cognition, Intelligence is my own work and I have documented all sources and material used.

Munich, 15.10.2021 Xavier Diaz Ortiz

Acknowledgments

I would like to thank first my parents Lorena and Javier, as well as close friends for their great emotional support and encourage during the development of this thesis, even when things seemed difficult at times. I appreciate also the collaboration with other student colleagues and the guidance of my supervisor Walter with the literature research and his helpful suggestions. Last but definitive not least, I am very grateful to the authors of the methods employed in this thesis who are pushing the limits of the fields Artificial Intelligence and Autonomous Driving even further. They kindly helped me to understand their work better and provided me with valuable feedback when I had questions to be able to move forward with my own ideas.

Abstract

This thesis has focused on the multi-modality 3D detection of objects in real-time, particularly combining the complementary information from camera images and LiDAR point clouds. The baseline for this project is the state-of-the-art deep learning architecture *CenterPoint* that is LiDAR-only based, meaning it operates solely on point cloud data to predict 3D bounding boxes with class classification around the located objects. The available *CenterPoint* models have been trained on popular autonomous-driving datasets like *nuScenes* and *Waymo* and have scored among the top-best performers on their respective 3D detection benchmarks. Furthermore, a late-fusion strategy of images and point-cloud frames has been applied to increase the accuracy of the 3D object detector following the proposal described in the recent *CLOCs (Camera-Lidar Object Candidates fusion)* network.

In this thesis *CLOCs* has been extended for multi-class fusion, training and inference and the method has proven to be valid for a new type of 3D detector like *CenterPoint*. Because speed matters for this project, in parallel with the LiDAR-based 3D object detection, *YOLOv4* has been selected to process the image stream and deliver 2D object instances very fast to the next fusion stage with *CLOCs*, therefore transforming the *CLOCs* method from an offline into an online real-time implementation. Moreover, ablation studies describe how well the detection pipeline does inference with *CenterPoint* alone vs. after the fusion step made by *CLOCs*, analyzing quantitatively and qualitatively the accuracy gained plus the latency introduced into the system. All experiments have been conducted on the *KITTI* dataset. Nevertheless, since this thesis should serve as a contribution to a larger project, the *Providentia*++ Intelligent Infrastructure System (IIS), the conclusion section also remarks how the experiment results can be applied to the use case of *Providentia*++.

Contents

1. Introduction

1.1. Motivation

While detection in 2D might be valid for some applications that consider object recognition and classification only within a plane, there are other scenarios where reasoning must happen in 3D nature like the robotic manipulation or navigation within a volumetric world by an intelligent agent. Operations in 2D do not suffice any longer because the estimation of depth implies the perception work must be conducted clearly in a three-dimensional space. Let us consider the field of autonomous driving that is experiencing major advances, especially when dealing with the perception of the environment state. For instance, an autonomous car travelling on the highway has to be able to localize and track with high accuracy other traffic participants in safety-critical tasks, e.g. lane change and collision avoidance, so that it can react instantaneously to unforeseen events. Therefore, 3D object detection in these cases becomes imperative to determine the exact location coordinates of objects and their real dimensions. Based on these 3D object attributes it is much more convenient for an autonomous car to make sense of what is happening in its surroundings and objects can be discriminated simply by size, Fig 1.1. However, local information gathered by only one vehicle may not be enough for a complete understanding of traffic scenes as a local point of view suffers from occlusions and limited range. With the aid of an Intelligent Infrastructure System (IIS) that can perceive the current traffic situation from a better global perspective a digital twin of the highway can be simulated in real-time. This simulation can then be shared with other road users thereby extending its field of view, which in turn could be beneficial for them to execute more suitable and safer maneuvers. To realize a digital twin as truthful as possible, 3D detection is a crucial task for building a reliable virtual representation that can map the position, orientation and dimensions of traffic participants from the real world to the simulator.

The German state-funded project *Providentia++* [1] is an example of such an IIS that counts with a test-bed on some selected portions of the highway A9 between the cities of Munich and Nuremberg. Different sensors such as cameras, radars and LiDARs are mounted at strategic high spots to capture the dynamic environment that involves the constant movement of incoming and outgoing vehicles on different sides of the

Figure 1.1.: Example of the 3D object detection task where an object is classified as 'Car' and its attributes like size, location and orientation are predicted as well in the LiDAR sensor reference frame

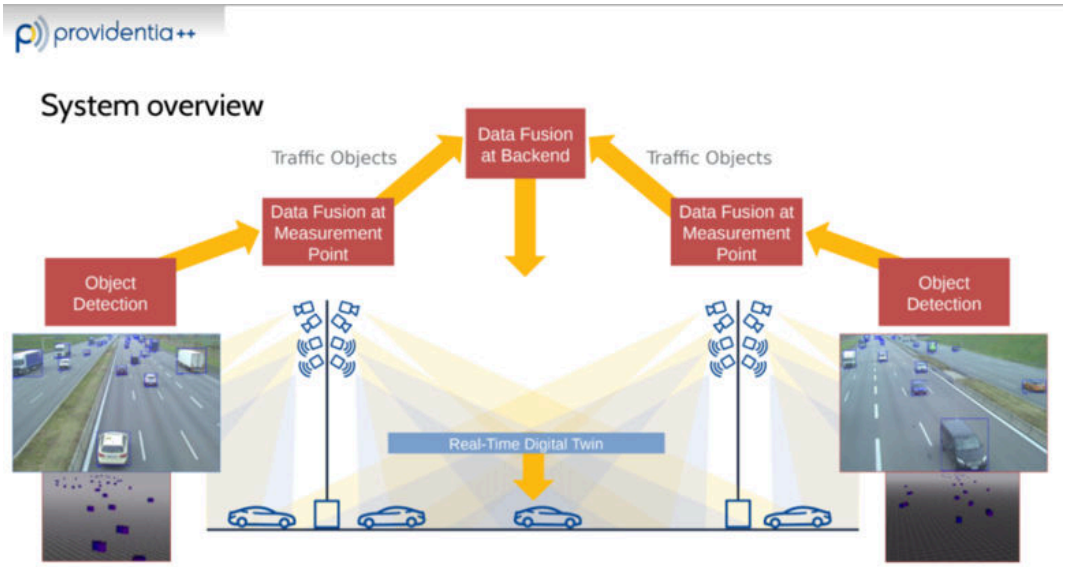


Figure 1.2.: Illustration of data flow in the *Providentia++* system

highway. Edge workstations process and fuse the sensor readings and object detections from a certain local measurement point; afterwards in the central backend, data from all measurement points is fused together to have an overall landscape of the whole stretch. The information about detected vehicles is transmitted via *ROS (Robot Operating System)* [2] interfaces to the *CARLA* [3] simulator for autonomous driving, where the detected vehicles are spawned and visualized. In *CARLA* by means of a HD map the highway has been depicted with high fidelity as well, thus the end result is the realization of a digital twin from the highway.

This thesis aims to support the *Providentia++* research team to improve the current 3D object detection solution. Up until now a fusion algorithm that relies on camera and radar was running on the backend system for 3D detection and tracking with satisfactory results for the most common small-size vehicles, the car class, but for larger vehicles (i.e. vans, trucks and similar), the estimation of dimensions was not precise enough. To further enhance the 3D perception capability of the system, LiDAR sensors were tested successfully in 2020 and finally installed in the summer 2021.

1.2. Contributions

Although classic methods for computer vision and sensor fusion algorithms with regard to the Kalman Filter and its variants have proven to be effective to some extent for decades, since the emergence of Deep Learning, the vast majority of state-of-the-art techniques rely on the development of artificial neural networks targeted to object detection, whether in 2D or 3D. Therefore, in this thesis a Deep Learning based solution has been proposed.

After extensive literature research, the real-time 3D object detection pipeline conceived is the outcome of coupling several state-of-the-art networks that will be described in more detail in Chapter 3. The actions taken to improve them or make them operate together as an ensemble are highlighted as follows:

- ***CenterPoint***: It is a network that regards objects as points for 3D object detection and tracking, it takes as input a LiDAR point cloud and predicts 3D bounding boxes for various classes of objects present in the scene. Actually, *CenterPoint* itself refers mainly to the detection head type which runs on top of either two backbones, for this project the *PointPillars* version was chosen as it is the optimal compromise between speed and accuracy. *CenterPoint* has been tested on large autonomous driving datasets, namely *nuScenes* and *Waymo*, ranking in the top places of their respective 3D detection benchmarks, thus its success is beyond doubt. However, in this thesis the *KITTI* dataset has been chosen to conduct experiments as it is of manageable size to work with and evaluate results locally in one computer. Furthermore, using the *KITTI* dataset helps to study how *CenterPoint* behaves with a new dataset it has not been officially tested before. Despite this, notice that it is not a goal of this thesis to develop a *CenterPoint* model that can perform with high-accuracy on *KITTI*.

- ***YOLOv4***: the fourth iteration of the YOLO (You Only Look Once) family has been employed to detect objects in 2D from *KITTI* image recording streams. *YOLOv4* has been made available pretrained on the *COCO* dataset, thus, it can already detect persons, cars and trucks out-of-the-box, but a new model has been trained targeted to the more specific classes available on *KITTI* like cars, vans, pedestrians, cyclists, etc. In this project it is used as a real-time tool due to its unbeatable inference speed to supply the next fusion-stage with very fast and quite accurate 2D detections.

- ***CLOCs***: the *Camera-LiDAR Object Candidates for 3D object detection* method is a tiny neural network that, as it name suggests, was designed to perform late-fusion of

any combination of 2D and 3D detectors. Among the multi-modality methods it ranks among the best ones in the car class of the *KITTI* test set. Simply speaking, this network takes as input the predicted 2D and 3D bounding boxes before NMS (Non-Maximum-Suppresion) with the same semantic label and learns to fuse both confidence scores into one final 3D score. Consequently, the synergy of images and point cloud data with *CLOCs* helps to increase the accuracy by removing false positives and minimizing the number of missed 3D detections. For this project the *CLOCs* core functionality has been extended to multi-class fusion and adapted for integration into the codebase of the 3D detector *CenterPoint* to allow training and inference of the fusion-stage with multiple *CLOCs* instances, one instance for every object class of interest. The *CLOCs* fusion method has been proven to work for a different kind of 3D detector it was not tested before, an anchor-free network like *CenterPoint*. With *YOLOv4* playing the role of a super fast 2D detector, *CLOCs* is well-suited for an online real-time application if some hardware requirements are satisfied.

Furthermore, ablation studies will determine the following aspects: the final average inference speed in FPS of the whole detection pipeline, the accuracy of *CenterPoint* alone vs. after the fusion step with *CLOCs*, and the latency introduced into the system by *CLOCs* and *YOLOv4* running in parallel with *CenterPoint*. A qualitative analysis of common fusion effects is given as well to interpret graphically the benefits and drawbacks of the proposed solution. At the end in Chapter 6, it is briefly explained how the *Providentia++* team could profit from the results found in this thesis.

2. Background

On the one hand, this Chapter presents concepts in relation to the topics of Sensor modalities, Sensor fusion schemes and description of the most popular Autonomous Driving datasets. On the other hand, the necessary condensed theory to understand the fundamentals of Deep Learning applied to point clouds and camera-LiDAR sensor fusion for multi-modal 3D object detection is given.

2.1. Sensor modalities

There exist many kind of sensors with distinct operating principles that are useful to sense different physical properties. On the one hand, there are sensors that give information about the own state of a system. For example IMU, acceloremeters, GPS, and wheel encoders can measure orientation, inertial forces, global position and distance travelled for odometry purposes. On the other hand, for 3D object detection, tracking and in general perception of the environment, the most common sensors found today in the automotive industry are LiDAR, Camera and RADAR, which are also present at the measurement points of the *Providentia++* IIS.

2.1.1. LiDAR

LiDAR (Light Detection and Ranging) is a device that makes periodically rotations of 360 degrees, at a rate of 10-20 Hz while sending out beams of light and measuring how long it takes for the beam of light to come back to determine the distance to an object. Each laser ray is in the infrared spectrum, and is sent out at many different angles, with a spatial resolution in the order of 0.1° due to the short wavelength of the emitted IR light. Laser intensity value is also received and can be used to evaluate material properties of the object the laser reflects off. The result after one spin of the LiDAR is a point cloud (Fig. 2.1), this data is usually stored in a pcd file format that contains the x, y and z coordinates of each scanned point in the device reference frame, plus some other attributes like the intensity value. Because the lasers in a LiDAR are concentrated in vertical layers, the number of layers has a big impact on the resolution of the point cloud, meaning the more layers a LiDAR has, the more points will be captured in the vertical field-of-view and hence, there will be less blind spots and objects will have

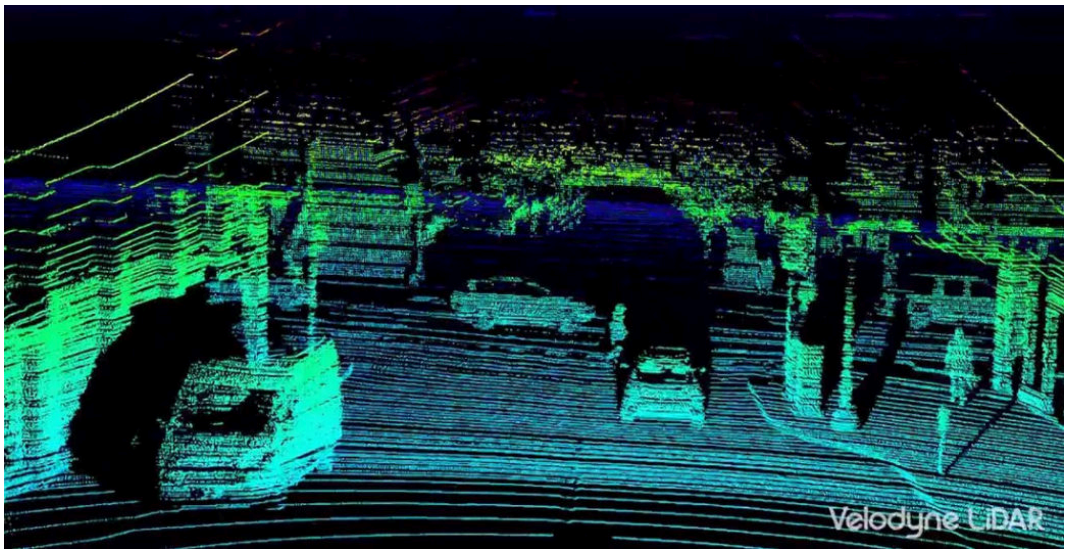


Figure 2.1.: Example of point cloud from a traffic scene generated by a LiDAR of the brand Velodyne.

more points on their surface. Also a LiDAR can send light beams up to a certain range, so the point density distribution will become lower with increasing distance from the sensor. While LiDAR sensors are not much affected by adverse weather conditions and they give us very high-resolution models for the world around us in 3D, they are currently the most expensive perception sensor.

2.1.2. Camera

When an image is captured by a camera, light passes through the lens and falls on the image sensor. This sensor consists of light sensitive elements that register the amount of light that falls on them and convert it into a corresponding number of electrons. Once the exposure time is complete, the generated electrons are converted into a voltage, which is finally transformed into a discrete number by means of an A/D-converter. Currently, there are two main image technologies, CCD (Charge-Coupled Device) and CMOS (Complementary Metal-oxide Semiconductor). Both technologies convert electrons into voltage and are inherently color blind, as they cannot distinguish the different wavelengths which generate the electrons. To enable color vision, tiny filter elements (i.e. micro-lenses) are placed in front of each pixel to allow only a certain wavelength to pass through. One common way to map wavelength to color is to arrange the filter elements in an RGB (Red, Green, Blue) pattern to allow the primary colors to pass through individually, which gives us three pixel arrays - one for each primary

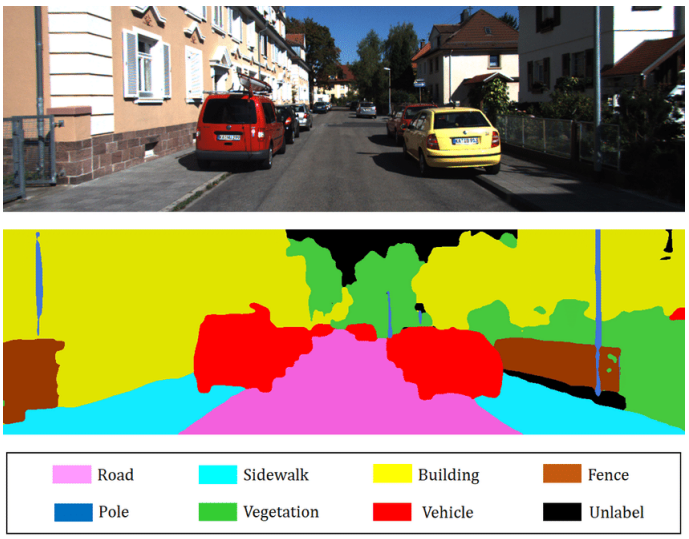


Figure 2.2.: Example of 2D image semantic segmentation, i.e. pixel-level object classification. (Top) input image. (Below) prediction

color. Cameras provide rich texture and color information. This is one of the prime advantages of camera systems and recent advances in AI emphasize this even stronger, that's why cameras are preferred for objection detection and classification in 2D or image semantic segmentation (Fig. 2.2). Deriving 3D geometry using 2D camera data is challenging as they do not provide depth (except for the case of stereo-cameras), but because they are affordable devices readily available in the market there are some algorithms that can leverage 2D features from a monocular camera to carry out 3D object detection, though the accuracy obtained lies far below what could be achieved by LiDAR-based methods.

2.1.3. RADAR

RADAR (Radio Detection and Ranging) works using the transmission and detection of electromagnetic waves, measuring the time of flight of the signal to compute the distance. Electromagnetic waves are reflected if they meet an obstacle, if these are received again at the place of their origin, then that means an obstacle is in the propagation direction.The range of detection can be from a few meters up to more than 200 m. RADAR lacks the capability to generate a high resolution point cloud, but it can highly accurate estimate velocity of the targets exploiting the Doppler effect of frequency shift. This is one its primary advantages over other sensors. In Fig. 2.3 RADAR points on some encountered objects can be seen. The frequency of electromagnetic energy used

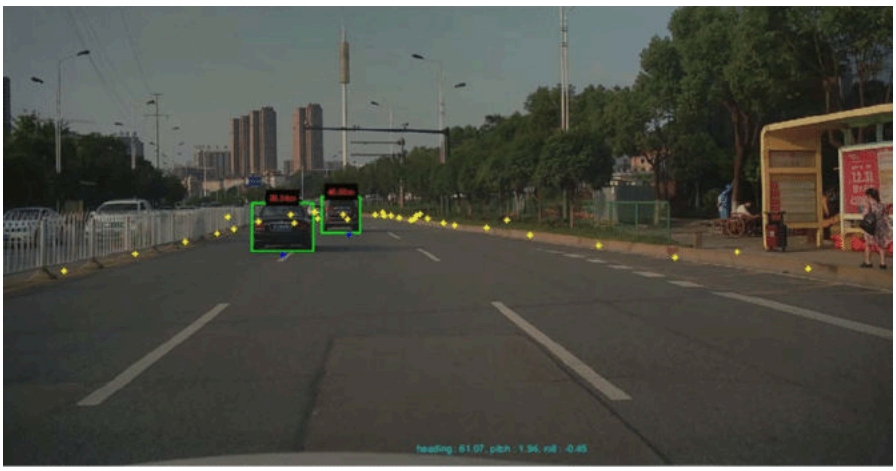


Figure 2.3.: Example of RADAR points spread in all directions projected onto an image. Two detected vehicles have RADAR points on the rear side.

for RADAR is unaffected by darkness and penetrates fog and clouds. This permits radar systems to determine the position of road targets invisible to the naked eye because of distance, darkness, or weather. A key factor to bear in mind is the low manufacturing cost for a RADAR, one unit can cost as low as a few hundred dollars, allowing a car manufacturer to deploy multiple RADAR sensors for 360 degree perception.

2.2. Sensor fusion schemes

Depending on the stage at which the data from several sensors, i.e. modalities, is merged together in a neural network, one can distinguish three fusion approaches: early-, deep- and late-fusion.

2.2.1. Early-fusion

The raw data from all sensors, after some preprocessing, is fused at the input level. This means usually every independent set of raw data is transformed first into an input feature vector, and then all feature vectors are concatenated or stacked together, Fig. 2.4. Only after the heterogeneous data have been transformed to a uniform ground codification, they can be fed into a machine learning algorithm. The main issue with early-fusion is devising a common feature representation space for all modalities with matching dimensions. Also, early-fusion requires that data from different streams is

well synchronized with the same timestamps, for which the data should be collected at a common sampling rate.

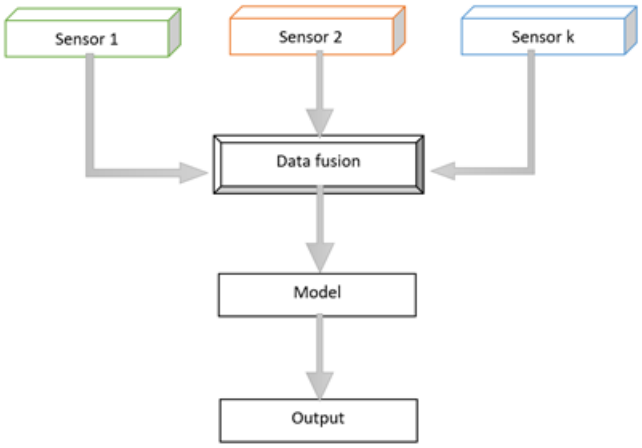


Figure 2.4.: Early-fusion illustration

2.2.2. Late-fusion

In contrast to before, now every stream of input data for all modalities is first passed individually to a corresponding data-type appropriate network model for inference, then all output results are fused at the decision-level, see Fig. 2.5. Late-fusion is a simpler and more flexible technique than early-fusion in the sense that non-homogeneous raw input data does not need to be converted to some common feature space, instead the individual models are more specific and tailored to the domain application of the sensor modality, for instance one model handles best only camera images, another model is specialized only in point clouds, etc. Several criteria can be used to define the optimal way about making a decision on how to finally merge each of the independent predictions from the trained models. Bayes rules, max-fusion and average-fusion are popular employed late-fusion rules among others.

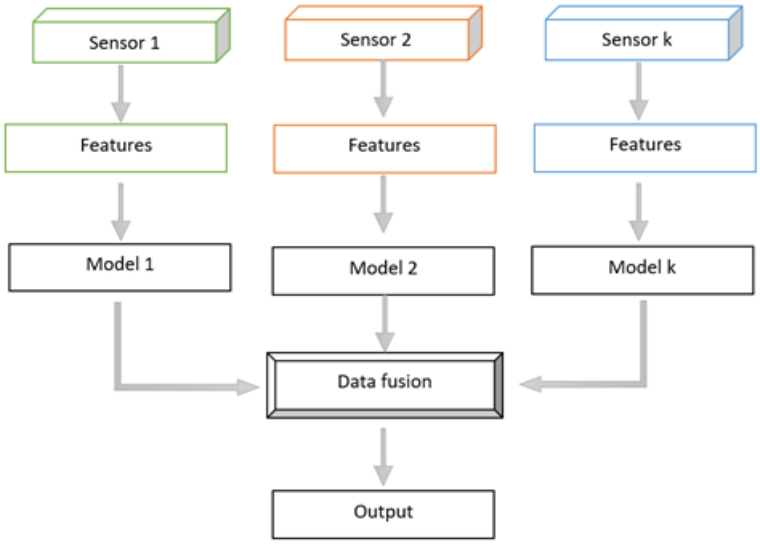


Figure 2.5.: Late-fusion illustration

2.2.3. Deep-fusion

This approach can be considered as something in-between the previous schemes. The data from different modalities is transformed to an intermediate feature space at the hidden layers of the neural network. The fusion of features is embedded into the network structure and can occur even at multiple stages of depth along the data downstream, see Fig. 2.6. Deep-fusion is more flexible than early-fusion, and enables the network to learn a joint-representation of each each modality before the final prediction. Features can be extracted from the typical kinds of layers, namely 2D convolution, 3D convolution and fully connected. The layer where the fusion of different modality features has taken place is called a fusion layer or a shared representation layer.

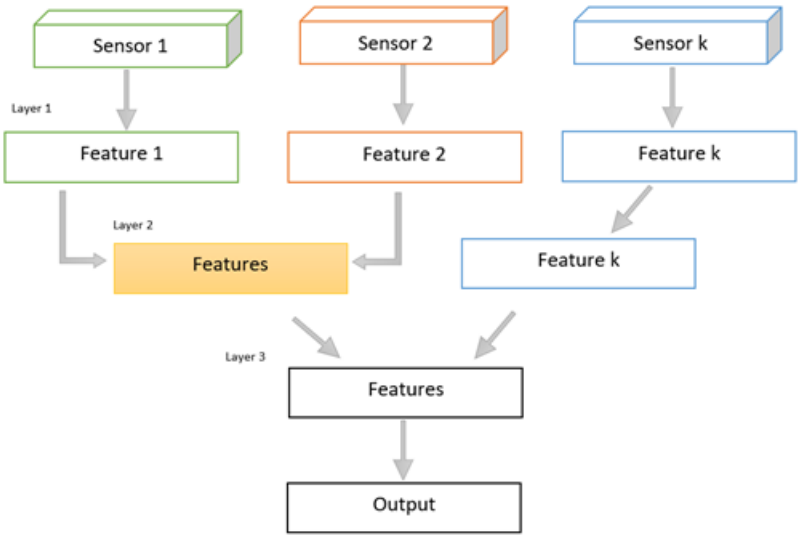


Figure 2.6.: Deep-fusion illustration

2.3. Autonomous Driving datasets

The conception of a satisfactory neural network model often requires a vast collection of training data for the domain and task of interest. In the field of Autonomous Driving the most relevant datasets made public available recorded from the ego-vehicle perspective and their evaluation metrics will be described next.

2.3.1. *KITTI*

KITTI [4] has been perhaps the dataset of reference for developing Autonomous Driving algorithms, as it was the first of its kind early released in 2012 in a conjunction project by the Karlsruhe Institute of Technology and the Toyota Technological Institute at Chicago. The *KITTI* dataset was captured by a VW vehicle that was driving in both urban and rural areas in the German city Karslruhe. In total, 6 hours of traffic scenarios were recorded at 10-100 Hz using a variety of sensor modalities such as color and grayscale stereo cameras, a Velodyne 3D laser scanner and GPS/IMU inertial navigation system. The dataset in *KITTI*, almost 15.000 frames total in size, is calibrated, synchronized and timestamped. Labels in 2D and 3D are provided for objects of different classes, being the most dominant ones 'Car' and 'Pedestrian'. *KITTI* offers benchmarks for various tasks like stereo, optical flow, depth completion, tracking, semantic segmentation, etc.

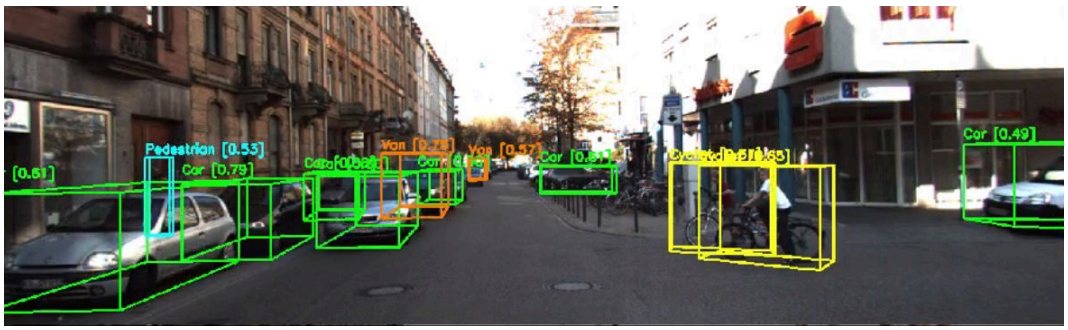


Figure 2.7.: Example of 3D object detection in *KITTI*

For the challenges of BEV (Fig. 2.8) and 3D object detection (Fig. 2.7), the evaluation metric used in *KITTI* is the Average Precision (AP) and mAP (mean Average Precision). For defining a positive match the bounding box Intersection over Union (IoU) in 3D or 2D for BEV - Fig. 2.9 - of at least 70% is necessary for Car, and 50% for Pedestrian/Cyclist. The AP can be understood as the area under the precision-recall curve using K recall sampling points for calculation with the Riemann integral. Originally 11 recall points were considered, but currently 40 points are taken for the AP estimation in the official test server. *KITTI* further divides the detection challenge into three categories: easy, moderate and hard. The difficulty degree is defined with respect to the bounding box height in pixels in the image plane, the occlusion level and truncation percentage.

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

2.3.2. *nuScenes*

The *nuScenes (nuTonomy Scenes)* [5] dataset was developed and released in 2019 by the company Motional (formerly named nuTonomy), which aims at producing driverless reliable vehicles. 1000 scenes, each 20 seconds long at 2 Hz were compiled by the fleet of the company in the diverse areas of the cities of Boston and Singapore. It is the first dataset that covers the full-suite of devices found in today's most common configurations of autonomous cars, thus apart from 6 cameras, 1 LiDAR, IMU, GPS, it also includes the Radar sweeps in its sensor data to encourage research in sensor fusion. Unlike *KITTI*, *nuScenes* is a huge dataset with 40.000 keyframes that contains 7x more object annotations of 23 classes. In *nuScenes* the 360 degree field-of-view of the

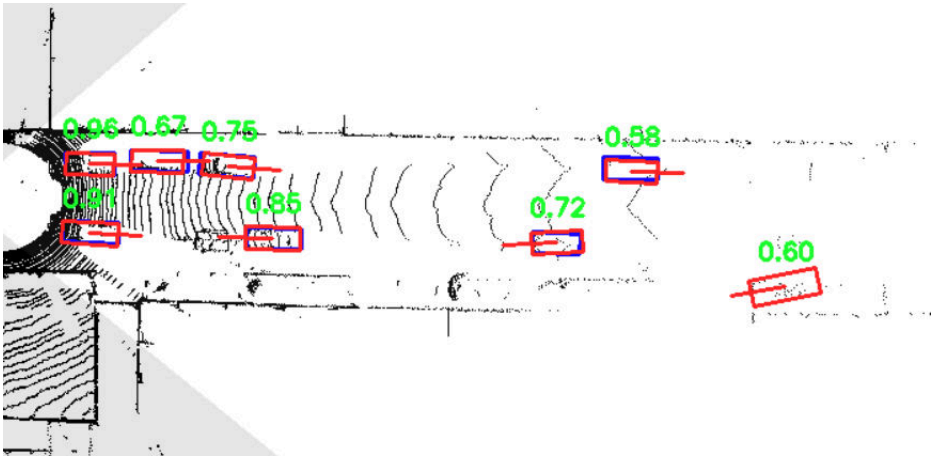


Figure 2.8.: Example of BEV object detection in *KITTI*

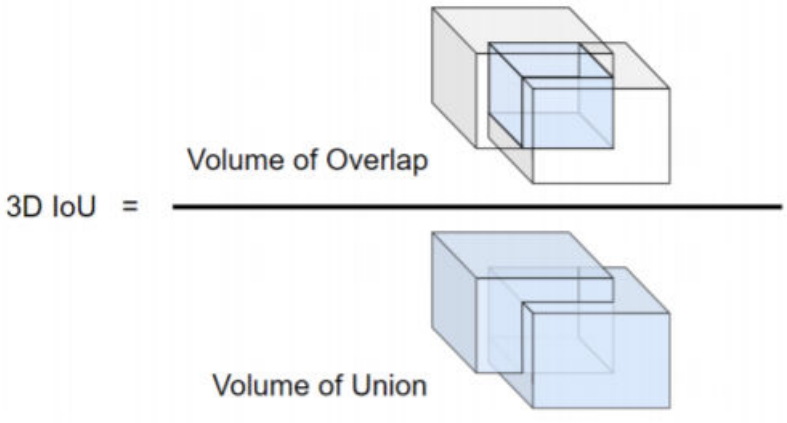


Figure 2.9.: Intersection over Union in 3D. The higher the IoU value for a certain threshold, the better a predicted bonding box matches the ground-truth label for a particular object

vehicle is covered, whereas in *KITTI* only the front-view of the camera distance range is taken into account. Moreover, *nuScenes* is the first multimodal dataset that contains data from nighttime and rainy conditions, road maps and with object attributes and scene descriptions in addition to object class and locations.

nuScenes offers benchmarks for object detection, tracking, lidar-segmentation and recently for trajectory prediction. As for the 3D object detection task they use the AP metric in a slightly different way, a match is defined by considering the 2D center distance on the ground plane rather than IoU-based criterion. Further, they define metrics for a set of true positives (TP) that measure average translation / scale / orientation / velocity and attribute errors. These TP errors are then converted to TP scores, the mean over classes is taken to calculate the mTPs and together with the mAP are condensed into a single metric, the *nuScenes* detection score (NDS), computed as the normalized sum of mAP with weight 5 and the rest of TP scores, each one with a weight of 1.

$$NDS = \frac{1}{10}[5mAP + \sum_{mTP \in TP}(1 - min(1, mTP))]$$

2.3.3. *Waymo Open Dataset*

The *Waymo Open Dataset (WOD)* [6] was first launched in August 2019 with a perception dataset comprising high resolution sensor data and labels. In March 2021, it was expanded to also include a motion dataset comprising object trajectories and corresponding 3D maps. The *WOD* was publicly released by *Waymo* - a company of Alphabet with roots in the Google self-driving car project of 2009 - to aid the research community in making advancements in machine perception and autonomous driving technology. They provide more than 100.000, 20s long (over 20 million frames) recordings at 10Hz from diverse geographies and conditions of 6 American cities like San Francisco, Los Angeles, Detroit, to name a few. The data acquired by 1 mid-range LiDAR, 4 short-range LiDARs, 5 cameras (front and sides) were labeled in the camera image and LiDAR point cloud for the classes Vehicles, Pedestrians, Cyclists and Signs.

The positive IoU thresholds are set to 0.7, 0.5, and 0.5 for evaluating vehicles, cyclists, and pedestrians, respectively. Moreover, the mean average precision is also used in the *WOD*, but it is weighted by heading to give as result the mAPH metric, since the regular average precision AP does not take the heading angle into account.

$$APH = 100 \int_0^1 max\{h(r')|r' >= r\}dr$$

where $h(r)$ is the PR curve weighted by heading accuracy. The prediction is from $[-\pi, \pi]$. Angle correction is between 0 and π. To get a weight between 0 and 1, a division by π is performed.

2.4. Deep Learning on point clouds for 3D object detection. A short overview.

The application of Deep Learning (DL) in the 2D domain is a very mature field. In Fig. 2.10 a typical 2D convolution is shown, a 3x3 filter is applied to a grid data structure as it could be the case for image features. Bello et al. [7] underline in their review the shortcomings of point cloud data as well as the initial history of Deep Learning to overcome the potential challenges. Unfortunately, using the same operations for images on point clouds is not so straightforward, since point cloud data, contrary to an array of pixels in 2D, is not regular organized in a grid-like fashion. Point clouds can have regions with varying density, they do not follow necessarily any order pattern whatsoever, Fig.2.11.

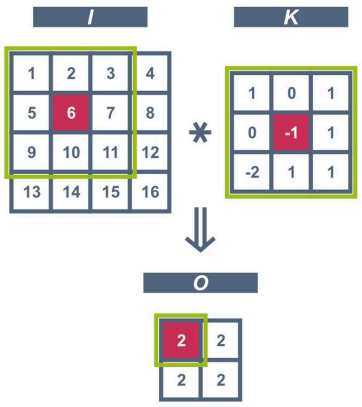


Figure 2.10.: Example of a typical convolution operation on a 4x4 input I by a 3x3 filter K that produces the result O of size 2x2.

These properties are difficult to handle for a normal convolutional neural network (CNN). Images are discretized in 2D as a grid of pixels, in a similar way initial at-

tempts were made to regularize point clouds by shaping them into ordered cubic cells, conversion known as voxelization (Fig. 2.12), to be later convolved with a 3D CNN. However, this leads to an inefficient high memory consumption due to the sparsity caused by voxelization as most voxels in the cubic grid are empty. For this reason, it was necessary that Deep Learning experts came up with algorithms to operate directly on the raw points of the point cloud instead.

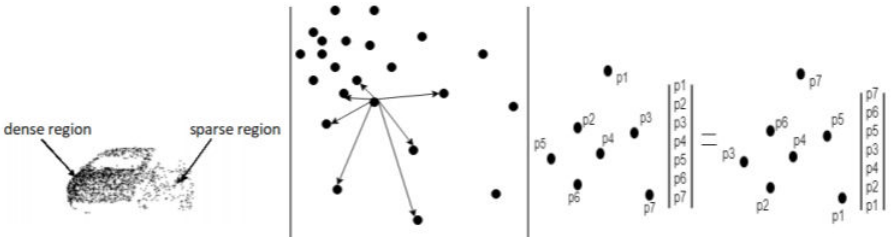


Figure 2.11.: Challenges in handling point cloud data [7]. Left: Point clouds can have regions with varying density. Middle: Lack of structure in the points distribution, the distance between points can be random and independent for each pair of points. Right: The order of points within a set is arbitrary and meaningless to encode them as a vector

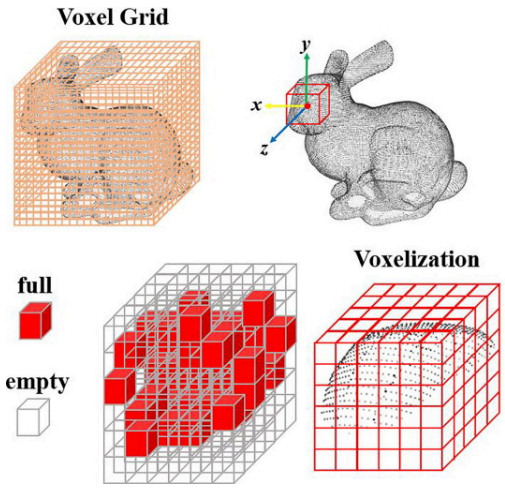


Figure 2.12.: Voxelization of a point cloud which transforms the data into an ordered data structure, a grid of cubic cells

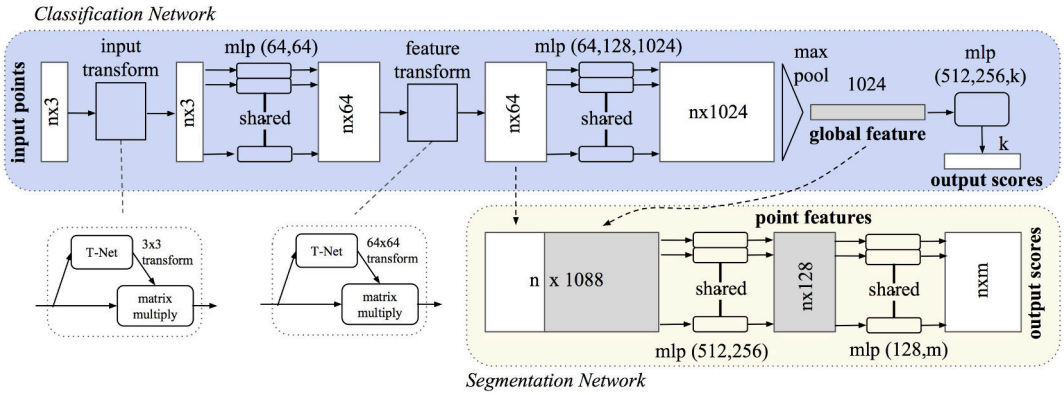


Figure 2.13.: *PointNet* [8] is composed of multilayer perceptrons (MLPs), which are shared point-wise, and two spatial transformer networks (STN) of 3×3 and 64×64 dimensions which learn the canonical representation of the input set. The global feature is obtained with a winner-takes-all principle and can be used for classification and segmentation tasks.

PointNet [8] is a pioneer technique worth mentioning because it inspired the foundations for other subsequent networks that were developed afterwards. The mulitlayer perceptron (MLP) and the max-pooling function are the key components of *PointNet*, basically the raw points are mapped by MLPs from an input d (d often equals 3 for x, y, z coordinates but could be larger for more channels) to a D = 1024 dimensional space in which all points share the same weights in each layer. The max-pooling symmetric function (symmetric meaning the output is not affected by the input order at all) is used to obtain a global feature vector descriptor of the input that could be passed to a classification or segmentation block, Fig. 2.13. One downside of *PointNet* is that it fails at capturing the local point structure because the local correlation among points is totally ignored. Later works tried to overcome this issue through sampling the point cloud to obtain centroids of the sampled regions and aggregating the k-nearest neighbours of these centroids into a local group. Then *PointNet* could learn to extract local features related to the local neighbourhood of point clusters.

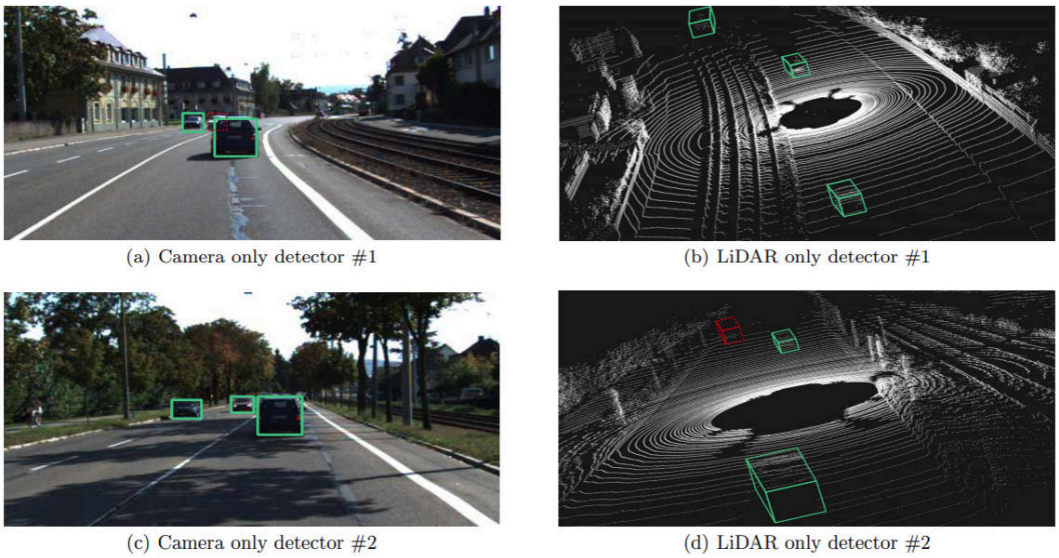
As already said, *PointNet* was the fist milestone in the application of Deep Learning to point cloud data, since its emergence a vast amount of sophisticated networks have arisen for more dedicated tasks such as 3D shape classification, 3D object detection, tracking, scene flow estimation and 3D semantic segmentation. Down below the taxonomy of state-of-the-art neural networks tailored to 3D object detection will be presented as defined by Guo et al.[9]. According to the underlying working principle, these networks can be generally sorted into:

- **Region Proposal-based methods**: In this case first several potential regions containing objects (called proposals, hence the name) are proposed by the network and then features at the region-level are extracted to predict the category label of each proposal. These methods can further be divided into:
  - **Multi-view based methods**: Proposal-wise featues from different views (eg. LiDAR front view, Bird Eye View (BEV) or even images) are fused at the early stage of the newtork. The computation cost is quite high.
  - **Segmentation based methods**: Semantic segmentation techniques are used to separate the foreground from the background points, then high-quality proposals are generated only on the foreground points reducing the computation burden. These methods are suitable for complex scenes with occluded and crowded objects.
  - **Frustum based methods**: 2D candidate regions of objects are taken from available 2D detectors, then a 3D frustum proposal for each 2D candidate region is extracted. Their performance is limited by the 2D image detector.
  - **Other methods**: *PointVoxel-RCNN* [10] stands out with excellent results on the car class of the *KITTI* 3D detection challenge. Its core idea is the combination of both a 3D CNN for generation of high-quality proposals on the voxelized point cloud, and a *PointNet*-based set abstraction for the learning of point cloud features.
- **Single-shot methods**: By means of a single-stage network, the class probabilities and bounding boxes for each object are regressed directly without any proposal general generation or post-processing, which makes possible for these methods to run faster. According to the input data, they can be categorized into:
  - **BEV-based methods**: These methods operate on the Bird Eye View of the point cloud and apply a FCN to estimate the heading angles and locations of objects as if the point cloud were a 2D image.
  - **Discretization-based methods**: The point cloud is converted into a regular discrete representation where a CNNs are applied to predict both the categories and 3D boxes of objects. *VoxelNet* [11] is one good example of such methods that partitions the point cloud into equally spaced voxels and the features within each voxel are encoded into a 4D tensor. A region proposal network is then used to produce the detection results. The performance achieved is high, but slow due to the sparsity of the voxels and the 3D convolutions. *PointPillars* [12] also lies in this category, but it will be explained in more detail in the next chapter.

- **Point-based methods**: These methods take the raw point cloud directly as their input. *3DSSD* [13] introduces a novel fusion sampling strategy for distance (D-FPS) and a feature FPS with the purpose of removing the time-inefficient feature propagation layers and the refinement module of PointRCNN [14]. Next, a Candidate Generation layer extracts the representative points which are fed into an anchor-free regression head to predict the 3D objects.
- **Other methods**: the dense Range View (RV) of the point cloud is taken as the input to *LaserNet* [15] and a fast mean-shift algorithm is employed to lower the noise produced by the per-point probability distribution prediction of bounding boxes. Another input representation was proposed in *Point-GNN* [16] by converting the point cloud into a graph of near neighbours with fixed radius to predict the categories of objects as well as the boxes.

2.5. Camera-LiDAR sensor fusion for multi-modal 3D object detection

The vast majority of multi-modal fusion networks found in the literature attempt to leverage the sensor modalities camera and LiDAR, only a few works deal with Camera-RADAR or LiDAR-RADAR combinations, examples of these are *CenterFusion* [17] and *RadarNet* [18] respectively. A reason for this could be that researchers worked early on with the *KITTI* dataset that contains merely Camera and LiDAR within its sensor suite, just recently *nuScenes* was the first dataset to include Radar point clouds, but still up-to-date the methods that occupy the top places in the benchmarks for 3D object detection are either primarily LiDAR-only based or to a lesser extent camera-LiDAR based. Wang et al. [19] in its survey discuss the pros and cons of using a single-modality detector. See Fig. 2.14 that shows an example of the issues inherent to single-modality detectors. On the one side a monocular camera-based 3D object detector does not provide reliable 3D geometry, it may demand high-computational cost and suffers from occlusion and bad weather and unfavorable light conditions such as rain, fog, dark shadows, bad contrast etc. On the other side a LiDAR-based 3D object detector provides naturally a better suitable 3D geometry, being an active sensor a LiDAR is not affected by insufficient light and is more resistant to adverse weather. But LiDARs are much more expensive devices than cameras, the point cloud data cloud generated by a LiDAR can become so sparse in the long range that objects with few points are barely recognizable, the point cloud can have low resolution for spinning LiDARs with less than 128 channels (however an exception are the solid-state LiDARs manufactured with a new technology), and the real-time perception could be limited by the refresh



(a) Camera only detector #1

(b) LiDAR only detector #1

(c) Camera only detector #2

(d) LiDAR only detector #2

Figure 2.14.: Illustration of typical problems for Single-Modal detectors. For scene #1, (a) shows a single camera cannot avoid the occlusion problem while the detection result of LiDAR only detector in (b) is correct; For scene #2, camera only detector in (c) performs well while LiDAR only detector shows the difficulty of detecting faraway vehicles with just a few LiDAR points in (d). Note that dashed red boxes stand for missed objects [19]

rate of a LiDAR, usually maximum 20 or 25 Hz, much lower than cameras. Therefore, the use of complementary information from camera and LiDAR inspires the field of multi-modal 3D object detection so as to take advantage of the strengths of both sensors while trying to diminish their shortcomings.

In the survey the authors identified that for feature fusion, i.e. early- or deep-fusion networks, the following are the fusion input representations most common: point cloud view (Bird Eye, Front or Range View) plus image feature map (a map here refers to the output of an image convolution layer), point cloud voxels plus image feature map, raw point cloud data plus image feature map, raw point cloud data plus image mask (a mask here means one portion of the pixel-wise image semantic segmentation), point cloud voxels plus image mask, point cloud voxels plus point cloud view plus image feature map, point cloud voxels plus image feature map plus image pseudo-LiDAR (here pseudo-LiDAR representation refers to the backprojection into LiDAR frame of the depth-map prediction for stereo or monocular camera images). The authors coin the

term 'fusion granularity' to define the level of detail at which fusion happens, which could be: RoI-wise via RoI (Region of Interest) pooling or acquiring 3D frustums from 2D RoIs, Voxel-wise when point cloud voxels are projected onto the image plane followed by feature extraction within 2D RoIs plus concatenation of pooled image features, Point-wise when the 3D points are projected onto the image using a known calibration matrix followed by feature extraction from a pre-trained 2D CNN plus concatenation of image or mask features; and Pixel-wise when the native representation of LiDAR, the Range View, is merged with an RGB image for posterior feature extraction with a 2D CNN.

In the survey also some of the challenges in multi-modal fusion are mentioned and the authors suggest solutions to alleviate these issues, namely:

- **Multi-sensor calibration**: Doing fusion of camera and LiDAR data implies the transformation from point cloud data to pixels or vice-versa, to the coordinate system of the other sensor via a calibration and projection matrix. For datasets like *KITTI*, *nuScenes* or *Waymo* this transformation between sensor frames is already provided to the research community. But in practice a self-made calibration for a new application scenario, when done manually with a target, could be a cumbersome process prone to a lot of errors and oftentimes the calibration must be performed as a routine because the sensors are subject to vibrations and forces from the environment. Assembling the sensors together to restrain the relative displacement as well as opting for an online targetless (aka automatic) calibration process would be a good idea.
- **Data alignment**: Pixels from camera images and point cloud data are acquired from different perspectives and usually there is not perfect correspondence between a point in 3D and the pixel domain in 2D, so this leads to quantization errors that could be mitigated with bilinear interpolation.
- **Information loss**: Each level of fusion granularity leads inevitably to some reduction of the input representation capacity, the finer the granularity the lesser this reduction. Perhaps the design of a loss function specific for multi-modal fusion could be a solution.
- **Multi-Modality Data Augmentation**: Operations to increase the size and variability of the training dataset are used in single-modality detectors to improve the training and generalization ability of the network. Though in multi-modal fusion these operations cannot be applied straightforward without an explicit and careful mapping from one sensor stream to the other. *MoCa* [20], described later, is a pioneer work with new strategies that address this issue.

- **Metrics**: Currently the object detection benchmarks focus only on the AP metric with no distinction if the submitted results come from a single-modal or multi-modal detector. It would be useful to have means to evaluate the special case of multi-modal fusion networks for comparison of other important metrics such as computation overhead, latency and robustness introduced by fusion.

Next, some remarkable state-of-the-art multi-modal fusion methods published in 2020 and 2021 will be described. Vora et al. in their work posed the question of why LiDAR-only based detectors significantly surpassed the accuracy of every prior network that at the time attempted to fuse LiDAR point clouds and camera images. This question seems reasonable as more redundant information gathered from multi-modal perception of the environment should in theory boost the final accuracy, or at least match, but not worsen, the same performance of the baseline LiDAR detector. They invented a sequential fusion technique named *PointPainting* [21] that can improve the accuracy of any detector. Essentially, it works as follows: projecting the point cloud into the corresponding image semantic segmentation result. After finding a mapping between the cloud points and pixels, the semantic segmentation scores of the corresponding match point-pixel are appended to each point, thus augmenting the dimensionality of the point cloud. *EPNet* [22] proposes a LiDAR-guided Image Fusion (LI-Fusion) module to establish a point-wise correspondence between raw point cloud data and the camera image, and flexibly give more or less weight to the image semantic features so that useful image features are utilized to enhance the point features whereas noisy image features are suppressed. *EPNet* also derived a consistency-enforcing loss (CE loss) to tackle the paradox of boxes being predicted with high classification confidence but low location score (low IoU with the ground-truth box) or the other way around, thus the CE loss encourages the network to optimize better the location and classification prediction. *3D-CVF* [23] combines the camera and LiDAR features using a cross-view spatial feature fusion strategy. First, the 2D camera features are transformed to a smooth spatial feature map with the highest correspondence to the LiDAR features in BEV. An adaptive gated fusion network helps to resolve the objects in BEV and balances the contributions from the two sensor sources using attention mechanism. In the first stage the region proposals are found based on the joint camera-LiDAR feature map. Then in a second stage 3D region of interest (RoI)-based pooling is applied to aggregate low-level LiDAR and camera features with the joint camera-LiDAR feature map, which leads to a better proposal refinement. *MAFF-Net* [24] is an end-to-end trainable single-stage multimodal feature adaptive network that exploits the raw RGB features from images. With *PointPillars* as the baseline and based on the channel-attention mechanism the authors of *MAFF-Net* designed two fusion modules to choose from, PointAttentionFusion (the channel

attention mechanism performs point-wise feature fusion of the two modalities) and DenseAttentionFusion (it converts the image and point cloud into three modalities, and then performs pillar-wise feature fusion of these multi-modalities), which help to filter the amount of false positives in the 3D detections while preserving the true positives. Xu et al. came up with *FusionPainting* [25] that can be considered a child and upgrade of *PointPainting* that brings even superior detection improvement. *FusionPainting* not only relies on 2D image semantic segmentation to extract features for each point, but a similar action is done via 3D semantic segmentation as well. Both 2D and 3D semantic segmentation score results are appended to each point and the extended point cloud passes through an adaptive attention-based semantic fusion module that learns context features at the voxel-level. Single-modality detectors increment the performance when data augmentation techniques are used during training, but for multi-modal networks this is not a trivial thing to do because one object sampled in the point cloud might appear in a location that does not make sense in the image or vice versa. For that matter Zhang et. al presented *Multi-mOdality Cut and pAste (MoCa)* as a new data sampling method very beneficial for multi-modal detectors that ensures consistency of the object cut-and-paste augmentation by avoiding occlusions and checking beforehand its plausibility both in the point cloud and image domains. For other operations like flipping, scaling, rotation, translation the authors contributed also a so-called multi-modality transformation flow in which the parameters and orders of transformations used by each augmentation are recorded. Then during fusion, any point in the LiDAR coordinates could find its corresponding image pixel coordinates by reversing the point cloud transformations and replaying the image transformation. In this way, any augmentation operation can be applied and be consistent for the multi-modality case as long as it can be reversed and replayed.

3. Related Work

In this Chapter the Deep Neural Networks that serve as the framework for the development of this thesis - namely *PointPillars*, *CenterPoint*, *YOLOv4* and *CLOCs* - are described for a better comprehension of the proposed detection pipeline solution that combines them all.

3.1. *PointPillars*

PointPillars [12] is one of the most efficient networks for the task of 3D object detection when it comes to maintain a balanced trade-off between run-time and accuracy. In fact, back then at the date of its release, it was considered the best method for 3D and BEV detection benchmarks in *KITTI*, outperforming previous attempts in terms of both accurate predictions and speed, by being 2-4 times faster than most methods running at 62 Hz. Roughly speaking, *PointPillars* uses a feature encoder to transform the point cloud into a sparse pseudo-image, then uses 2D convolutions to have a high-level representation of the pseudo-image; and a detection head is used to regress the 3D boxes, Fig. 3.1.

The conversion to pseudo-image is done by discretizing the point cloud - assuming the input dimension to be D=4 for x,y,z and reflectance r - into a grid only along the x and y ranges, leaving the z out as the common height of a set of pillars P. The points in each pillar are augmented with the arithmetic mean of the points inside the pillar and the offset to the particular pillar center, thus making the point cloud dimension D=9. Due to the sparsity of the point cloud a limit is imposed on the number of non-empty pillars (P) and the number of points-per-pillar (N), to create a tensor of size DxPxN, subject to random sampling or zero-padding to compensate for too much or too few points within the pillar respectively. A *PointNet* with a linear layer, Batch Norm, ReLU plus MaxPool is used to encode the features of the tensor and after a scatter operation a pseudo-image of size C,H,W is created, where C is the number of channels, H is the height, and W is the width of the canvas. Next. a backbone is applied to the pseudo-image in two steps, first a network produces features at decreasing resolution by a series of blocks (S,L,F) being S the stride size, L the number of 3x3 2D-conv-layers and F the output channels. A second network acts on these intermediate top-down

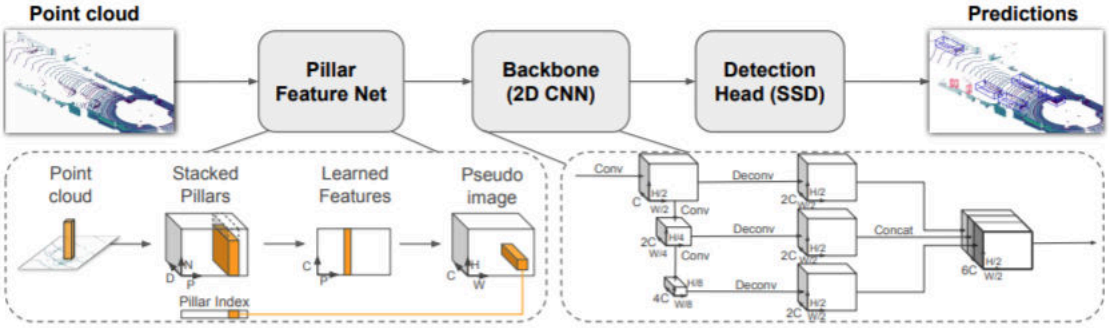


Figure 3.1.: Schema of the *PointPillars* network [12]

features to do an upsampling and concatenation of the features generated at different strides. Finally, a Single-Shot-Detector (SSD) using the matching of priorboxes with ground-truth via the 2D IoU performs the 3D object detection. The bounding box height and elevation become a target for regression given a box match in 2D.

PointPillars employs the same loss functions introduced in *SECOND* [26]. Ground truth boxes and anchors are defined by $(x, y, z, w, l, h, \theta)$. The localization regression residuals between ground truth and anchors are defined by:

$$\Delta x = \frac{x^{gt} - x^a}{d^a}, \Delta y = \frac{y^{gt} - y^a}{d^a}, \Delta z = \frac{z^{gt} - z^a}{h^a}$$
$$\Delta w = \log \frac{w^{gt}}{w^a}, \Delta l = \log \frac{l^{gt}}{l^a}, \Delta h = \log \frac{h^{gt}}{h^a}$$
$$\Delta \theta = \sin\left(\theta^{gt} - \theta^a\right),$$

where x^{gt} and x^a are the ground truth and anchor boxes and $d^a = \sqrt{(w^a)^2 + (l^a)^2}$. The total localization loss is then:

$$\mathcal{L}_{loc} = \sum_{b \in (x,y,z,w,l,h,\theta)} \text{SmoothL1}\left(\Delta b\right)$$

Since the angle localization loss cannot distinguish flipped boxes, a softmax classification loss is used on the discretized directions, $\mathcal{L}_{dir}$, which enables the network to learn the heading.

For the object classification loss, the focal loss is used:

$$\mathcal{L}_{cls} = -\alpha_a \left(1 - p^a\right)^\gamma \log p^a,$$

where p^a is the class probability of an anchor. with original paper settings of $\alpha = 0.25$ and $\gamma = 2$. The total loss is therefore:

$$\mathcal{L} = \frac{1}{N_{pos}} \left(\beta_{loc} \mathcal{L}_{loc} + \beta_{cls} \mathcal{L}_{cls} + \beta_{dir} \mathcal{L}_{dir} \right),$$

where N_{pos} is the number of positive anchors and $\beta_{loc} = 2$, $\beta_{cls} = 1$, and $\beta_{dir} = 0.2$.

3.2. *CenterPoint*

CenterPoint [27] is a novel network that has obtained outstanding results lately. This network deems 3D object detection as a task of finding, in a first stage, the centers of objects from which other attributes such as position, size and orientation are estimated by different regression heads. In a second stage, other point features, coming from the face centers of the initial bounding box estimate, are condensed into a vector and fed into a `MLP` for prediction of confidence scores as well as for further refinement of the box attributes, Fig. 3.2. By considering 3D objects as rotational-invariant points, this network is more flexible in the prediction of the orientation. This flexibility leads to more accurate estimates of the heading angle than anchor-based methods, meaning for example in a straight road, both point-based and anchor-based methods perform similar, but when the axes of objects do not match the direction of the ego-vehicle coordinate frame, i.e. if the vehicle is turning, anchor-based methods struggle to fit the correct axis-aligned bounding boxes to rotated objects. In contrast, treating objects as points invariant to rotations alleviates this issue. Also, being an anchor-free method, there is no need to specify in advance predefined anchor sizes for each object class, with positive and negative thresholds in the 2D Box IoU for target assignment, thus, the center-based network does not require a priori anchor-parameters and training becomes easier and computational less expensive. Another advantage of *CenterPoint* is that performing tasks beyond 3D detection, such as velocity regression and tracking becomes much simpler with a point-based characterization of objects. The implementation of the detection head was inspired by the principle of *CenterNet* [28], that is, formulating object detection in 2D as a problem of keypoint estimation in the image to predict an output heatmap. In this manner, every local peak of such heatmap corresponds to the center of a detected object with confidence proportional to the heatmap value at the peak. The authors of *CenterPoint* were able to translate this idea from 2D to the 3D domain: after processing the input point cloud with either the *PointPillars* or *VoxelNet* backbones for feature extraction, the result is a map-view feature-map $\vec{M} \in \mathbb{R}^{W \times L \times F}$ of width W and length L with F channels in a map-view reference frame. Both width and height directly relate to the resolution of individual voxel bins and the backbone network's stride. This map-view is then passed to a 2D CNN detection head to produce

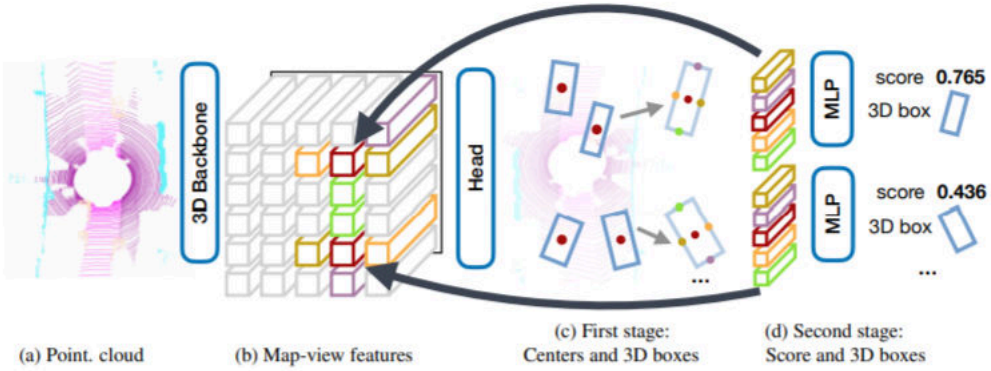


Figure 3.2.: Schema of the *CenterPoint* network [27]

a K-channel $w \times h$ heatmap $\hat{Y} \in [0,1]^{w \times h \times K}$ – one channel for each K class respectively - with peaks at the center location of the detected object. In the training phase, the centers of ground-truth bounding boxes are projected into a map-view so that the detection head can target a 2D Gaussian around these projections using a focal loss. Moreover, separate regression heads handle several object properties at center-features of objects: a sub-voxel location refinement $o \in \mathbb{R}^2$, height-above-ground $h_g \in \mathbb{R}$, the 3D size $s \in \mathbb{R}^3$, and a yaw rotation angle $(\sin(\alpha), \cos(\alpha)) \in \mathbb{R}^2$. *CenterPoint* aggregates both the heatmap and object properties regression losses in one common objective and jointly optimizes them.

The limitations of *CenterPoint* stem from the nature of the backbones used, that is, generally the *VoxelNet* flavour of *CenterPoint* is slower but more accurate than its *PointPillars* counterpart. Also, the second-stage refinement brings zero to little improvement depending on the dataset intrinsics; according to the authors, the second-stage was superfluous in the experiments with *nuScenes*, whereas for *Waymo* it did help to boost the accuracy. The big success of *CenterPoint* is supported by the facts that, at the time of writing, it ranks 1st in the 3D object detection benchmark of *nuScenes* with a `mAP` of 0.67 and a `NDS` of 0.714. The next place is another submission that is an extension of the core *CenterPoint*. Moreover, on the *Waymo* 3D detection challenge, it ranks 2nd with a mean `APH` of 0.7193 (average precision with heading, level 2 of difficulty).

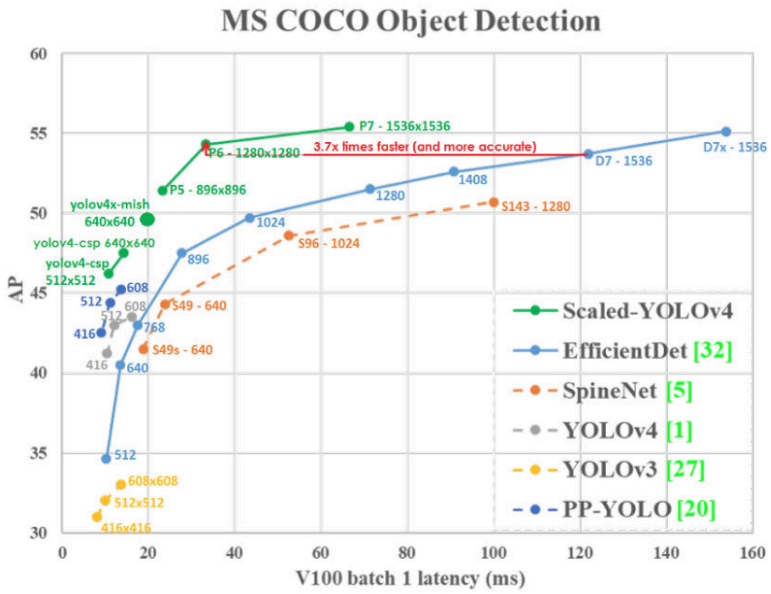


Figure 3.3.: Comparison of the *YOLOv4* peformance with other real-time detectors on the *MS COCO* dataset. Tested with a batch size of 1 on a GPU Tesla V100

3.3. *YOLOv4*

YOLOv4 - stands for "You Only Look Once" version 4 [29] - is a real-time 2D detection network that takes as input an image and predicts the bounding boxes and classes of objects present in the scene. The main reason why *YOLOv4* was developed is to yield high-quality predictions while achieving a very fast inference time for the most common practical industrial applications. Indeed, *YOLOv4* is able to get 43.5% Average Precision (65.7% AP50) for the MS COCO dataset at a speed of ~65 FPS on a GPU Tesla V100. The official implementation of *YOLOv4* is released under the open-source neural network framework *Darknet* [30] (written in the C language) and it is better both in terms of accuracy and speed than real-time neural networks Google TensorFlow *EfficientDet* [31] and Facebook *Detectron* [32], *RetinaNet* [33] on *MS COCO*. See Fig. 3.3 for a chart comparison, although recently some new architectures were released that seemingly are now on top of the performance ladder, such as *Scaled-YOLOv4* [34], *PP-YOLO* [35] and *PP-YOLOv2* [36].

It is beyond the scope of this thesis to expose in depth the operation principles of the YOLO family detectors, hence only a shallow description will be given and what

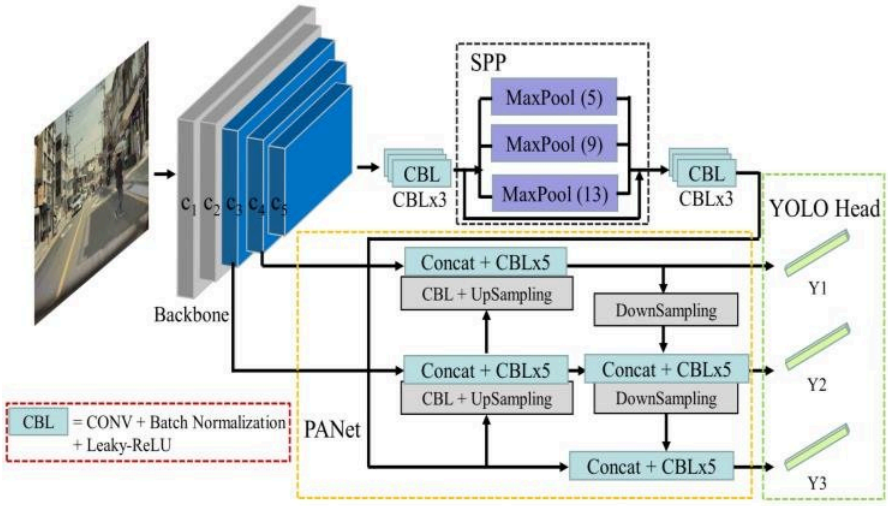


Figure 3.4.: Schema of the *YOLOv4* architecture

novelties were introduced since the third iteration. In general a 2D object detector is usually composed of a Backbone pretrained on the ImageNet dataset that can extract features from images, a Neck with a bottom-up and top-down structure of CNN layers to collect feature maps at different resolutions; and a Head which could be of types one-stage or two-stage. The latter head uses a Region Proposal Network to generate regions of interest then classifies and generates bounding boxes whereas the former head treats it like a regression problem by taking an input image and learning the class probabilities and bounding box coordinates. YOLO is the most popular representative of the one-stage category.

YOLOv4 improves *YOLOv3*'s AP by 10% and FPS by 12% due to the so-called Bag-of-Freebies and Bag-of-Specials, which are a compilation of techniques and strategies that optimize the performance. Bag-of-Freebies are methods that only increase the training cost and make the object detector receive better accuracy without increasing the inference cost, for instance the Cutmix and Mosaic data augmentations, CIoU-loss, DropBlock regularization, etc. Bag-of-Specials refers to a set of modules that only increase the inference cost slightly but significantly improve the accuracy of object detection, this could be for example employing the Mish activation function, Cross-stage partial connections (CSP), and Multi-input weighted residual connections (MiWRC). Additionally, the authors selected optimal hyper-parameters while applying genetic algorithms and a new method named Self-Adversarial Training (SAT) is used where first the original image is altered instead of the weights, then the network is trained

to detect the objects in this fake image. In short, *YOLOv4* uses CSPDarknet53 as the backbone, modified SPP (Spatial Pyramid Pooling Layer) and PANet (Path Aggregation Network) as the neck, and the anchor-based *YOLOv3* as the head, Fig. 3.4.

3.4. *CLOCs*

CLOCs - Camera-LiDAR Object Candidates Fusion for 3D Object Detection - [37] is a novel decision fusion (aka late fusion) method that enables the combination of any pair of 2D image-based and 3D LiDAR-based detectors with the purpose of improving the accuracy performance of the single-modality 3D detector. *CLOCs* itself is a network of tiny size that performs late fusion on the outputs of the 2D and 3D detectors, ideally but not necessarily before these apply non-maximum-suppression (`NMS`). In this way, all potential candidates coming from both modalities participate in the fusion step, so that in the end the final number of possible true detections is maximized. *CLOCs* relies on the principle that association of objects from different modalities is a reasonable procedure to do provided that the detection candidates satisfy geometric and semantic consistencies, Fig. 3.5. The former means 3D bounding boxes of detected objects in the point cloud domain, when its corners are projected onto the corresponding image frame, should have a high IoU with a particular 2D bounding box detection on the image plane. The latter refers to the concept that fusion of detection candidates should take place if and only if both the 3D and 2D detector predict the same class label for one pair of 3D and 2D intersecting boxes.

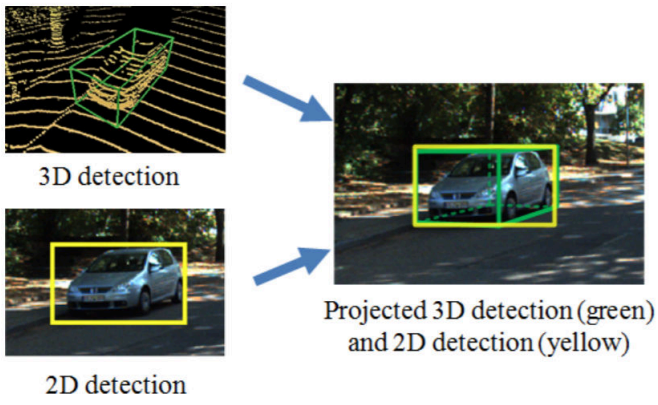


Figure 3.5.: Example of geometric consistency of the 2D and 3D detections applied to the semantic 'Car' category [37]

CLOCs encodes k and N detection candidates from 2D and 3D domains respectively, into a $k \times n \times 4$ tensor $\mathbf{T}$ of joint-detection candidates, in which one element has four features as follows:

$$\mathbf{T_{i,j}} = \{IoU_{i,j}, s_i^{2D}, s_j^{3D}, d_j\}$$

where $IoU_{i,j}$ is the IoU between i_{th} 2D detection and j_{th} projected 3D detection, s_i^{2D} and s_j^{3D} are the confident scores for i_{th} 2D detection and j_{th} 3D detection respectively. d_j represents the normalized distance between the j_{th} 3D bounding box and the LiDAR in xy plane. Elements $\mathbf{T_{i,j}}$ with zero IoU are discarded as they are geometrically inconsistent. It may occur that a projected 3D bounding box does not intersect a 2D box because the 3D detector found a candidate not available in the image, therefore the 3D box information remains valid and useful; and in such a case the tensor element $\mathbf{T_{i,j}}$ is still filled with $IoU_{k,j}$ and s_k^{2D} as -1, but not zero to help the network learn to distinguish the actual condition when the $IoU_{k,j}$ is very low.

As only few 2D detections may intersect some projected 3D detected box, this results in a sparse tensor with most entries null. However, only the non-empty elements of the input tensor are processed by the network after their indices are saved in a cache. The fusion network consists of a set of 1×1 2D convolution layers. Conv2D($c_{in}, c_{out}, \mathbf{k}, \mathbf{s}$) represents a 2 dimensional convolution operator where c_{in} and c_{out} are the number of input and output channels, $\mathbf{k}$ is the kernel size and $\mathbf{s}$ is the stride. *CLOCs* employs four convolution layers sequentially as Conv2D(4, 18, (1,1), 1), Conv2D(18, 36, (1,1), 1), Conv2D(36, 36, (1,1), 1) and Conv2D(36, 1, (1,1), 1), which yields a tensor of size $1 \times p \times 1$, where p is the number of non-empty elements in the input tensor $\mathbf{T}$. For the first three convolution layers, after each convolution layer applied, ReLU is used. Since the indices of these non-empty elements (i, j) are saved, a tensor $\mathbf{T_{out}}$ of shape $k \times n \times 1$ can be constructed by filling p outputs based on the indices (i, j) and putting negative infinity elsewhere. Finally, this tensor is mapped to the desired learning targets, a probability score map of size $1 \times n$, through maxpooling in the first dimension, see Fig. 3.6. During training cross-entropy loss was used for target classification, modified by the focal loss to address the large class imbalance between targets and background.

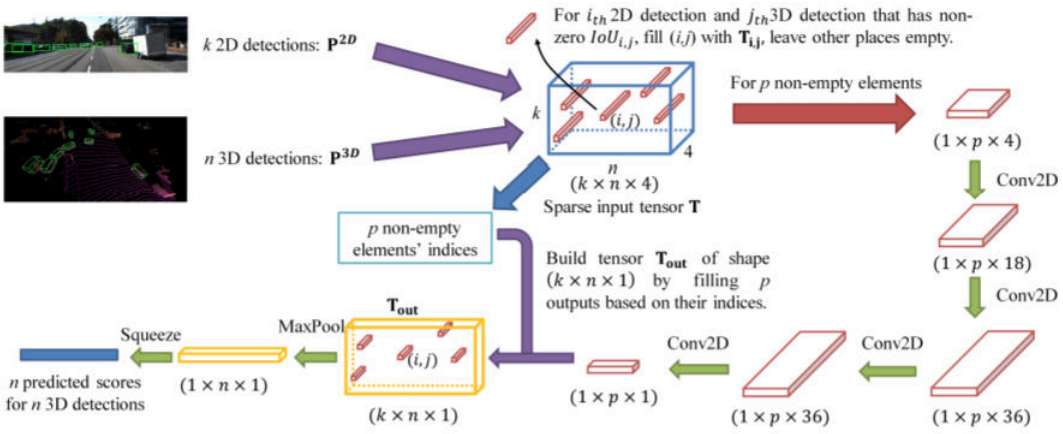


Figure 3.6.: Schema of the *CLOCs* fusion network [37]

4. Solution approach

This Chapter outlines the architecture of the proposed solution developed in this thesis for the task of real-time multi-modal 3D object detection. As already mentioned previously, the architecture is an ensemble after coupling several blocks under the premise that one GPU or two GPUs are powerful enough to run the detection networks simultaneously in parallel: *CenterPoint* as the 3D object detector, *YOLOv4* as the 2D object detector and *CLOCs* as the late-fusion method, see Fig. 4.1. The authors of *CLOCs* released their work as an offline implementation strongly attached to the codebase of the 3D detector *SECOND*. With offline meaning that the 2D detections required for fusion, i.e. the results of *Cascade-RCNN*, were stored and merely read from text files in *KITTI* format. Therefore, it was necessary to adapt the core functionality of *CLOCs* to be integrated into the codebase of *CenterPoint* to make possible the training, evaluation and online inference of the fusion network.

CLOCs was released to perform fusion between *SECOND* and *Cascade-RCNN* for one class only, the Car class in *KITTI*. In order to extend *CLOCs* with a multi-class fusion capability the following steps were taken: a single *CenterPoint* model and a *YOLOv4* detector were trained on the *KITTI* dataset to perform inference on multiple classes at once. For training and evaluation the *YOLOv4* results after NMS for each image in the train + valid partitions were put in text files, not exactly in *KITTI* format, but just the essential information about class category, 2D box coordinates (top, left, right, bottom) and the prediction score was stored, only if such score was bigger than a 0.3 lower-bound threshold. In *CenterPoint* each class of interest is assigned a separate detection head for itself alone in the configuration of the network. The point cloud range, voxel size and stride of the backbone defined for a particular *CenterPoint* model establish that each detection head before NMS produces the exact same amount of boxes, H x W, (equals the size of the predicted center-heatmap), so in general for N classes the total number of raw 3D boxes and heatmap scores predicted by *CenterPoint* is N x H x W. Recall that in *CenterPoint* a heatmap score represents the probability of a 3D box center with confidence proportional to the local peak value at the location of the heatmap. Before fusion, these 3D boxes must go trough a preprocessing function, first they are transformed from the LiDAR frame into the camera frame and projected into the image plane with the sensor calibration matrices provided by *KITTI*. Thus,

from original 3D boxes predictions, after projection into the image, we get pseudo-2D boxes predictions. Then for each class, a sparse input tensor is built with the 4 channels described in the *CLOCs* paper, namely, the score of the 3D box, the score of the 2D box, the IoU between the pseudo-2D boxes generated from the 3D boxes predicted by *CenterPoint* and the actual 2D boxes predicted by *YOLOv4* corresponding to the same class, plus the normalized distance between a 3D bounding box and the LiDAR in xy plane. The indices for each sparse input tensor are recorded as well and both each tensor and its index chache are passed to a *CLOCs* instance module for fusion, again each class is handled independently by a separate *CLOCs* instance. One such *CLOCs* instance module refers here to the series of CNNs described in the *CLOCs* section of Chapter 3. The result for all *CLOCs* instances is the new fused heatmap scores that replace the initial 3D scores prediction.

Now the fused scores output of each CLOCs instance are treated differently depending on the current execution mode, i.e. training, evaluation or testing. For training the scores of each class are reshaped to a 4-dimensional tensor (B, H, W, C) and then permuted to (B, C, H, W), being B = 1 the batch size and C = 1 the number of classes in a single detection head. The justification of this change is to match the dimensions of the heatmap predicted by one detection head in *CenterPoint*. Unlike the original work in *CLOCs* where the authors used the focal loss with predefined thresholds for positives and negative targets based on the 3D IoU between 3D box predictions and ground-truth boxes, here instead, the same focal loss employed in *CenterPoint* for heatmap regression is used for training all the *CLOCs* instances together, applying the sigmoid function to the new fused scores. Recall that in *CenterPoint* the heatmap focal loss targets 2D Gaussians around the centers of the ground-truth bounding boxes projected into the map-view. It makes sense not to have hard-coded thresholds for training *CLOCs* given that *CenterPoint* is an anchor-free detector. It is important to point out that the *CLOCs* instances are grouped as a Fusion Layer module of which an already trained *CenterPoint* detector is a member and its checkpoint is first loaded. The Fusion Layer relies on the *CenterPoint* 3D predictions for training, but the weight parameters of the prior trained *CenterPoint* have to be freezed so as not to modify its already optimized weights during backpropagation of the gradients. Otherwise *CenterPoint + CLOCs* would be trained together in an end-to-end manner, but this is not desirable, solely and exclusively the Fusion Layer weights have to be updated in the fusion process during training. For evaluation or testing the fused scores of each class are reshaped to a 3-dimensional tensor (B, H x W, C), again here B = 1, C =1 and sigmoid is applied. Lastly the NMS step is done to obtain the final set of 3D boxes.

Notice that 3D boxes before NMS and 2D boxes after NMS are the input data for

fusion because experimentally it was found that this was the best combination. Feeding too many 2D detections from *YOLOv4* before NMS could spoil the fused scores for some classes downgrading the performance, even leading to worse accuracy. A reason for this might be that the number of 3D boxes predicted by *CenterPoint* is several orders of magnitude less than in *SECOND*, thus, also less 2D boxes for fusion may be more optimal. Besides, having too many duplicate 2D boxes before NMS can increase the inference time considerably to no avail as in the preparation of the input tensors for *CLOCs*, the IoU would be calculated between a projected 3D box and several 2D boxes with a huge redundant overlap that actually belong to the same detected object in the image plane.

As for the online implementation of the proposed architecture solution, *ROS* is used here as a communication middleware for testing inference in a real-time scenario. In this regard, a particular sequence of the raw *KITTI* dataset [38] is converted to a rosbag file that is the player of two sources of raw sensor data, camera images and point cloud frames synchronized at 10 Hz. *YOLOv4* is active in its own *ROS* node that subscribes to the camera stream, parses the image and broadcasts the 2D boxes detections on-the-fly. Simultaneously, *CenterPoint + CLOCs* (C+C) run both in a separate *ROS* node that subscribes to the point cloud channel, *CenterPoint* in this node first predicts the initial raw 3D boxes, then upon receiving the 2D boxes published by *YOLOv4* the fusion of candidates is done with the *CLOCs* Fusion Layer; and the final 3D boxes detections are delivered to another topic. To take advantage of *CenterPoint* running alongside *YOLOv4* and process the image and point cloud frames efficiently in parallel, *CenterPoint* in the C+C node does inference on the incoming point cloud as soon as it arrives without waiting for the 2D detections to be delivered. The raw 3D boxes predicted by *CenterPoint* are stored in a queue and later retrieved for fusion when the 2D detections are available. Otherwise, if the C+C node has to keep waiting for *YOLOv4* to publish its results, the process would be sequential and not parallel, introducing more latency due to *YOLOv4*'s inference time. Additionally, the 2D and 3D boxes are drawn onto the incoming images to be published by the *YOLOv4* and C+C nodes for visualization of qualitative results and debugging purposes, see Fig. 4.2. To ensure the time consistency among the 2D and 3D detections, these are supplied with the same timestamps of the corresponding image and point cloud respectively so that *ROS* can manage well the synchronization. To guarantee the feasibility of the whole detection pipeline the 2D and 3D detectors should have similar inference speeds for the 2D and 3D predictions to reach the fusion stage at the same pace, ideally the 2D detector should run a little faster. This is certainly true for the tandem *YOLOv4-CenterPoint*.

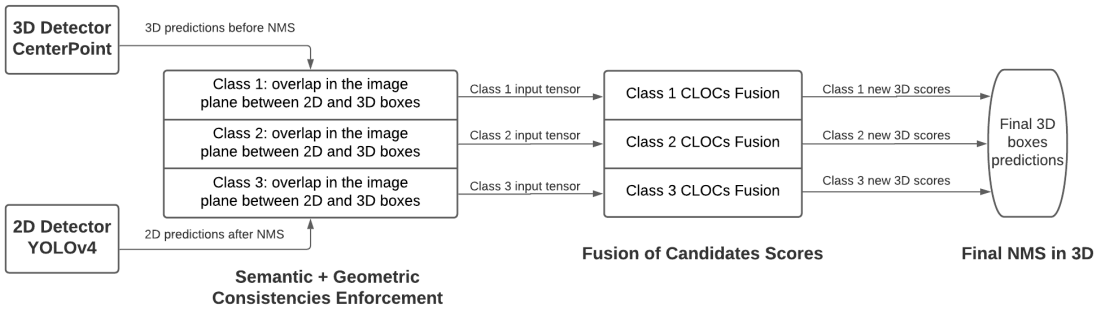


Figure 4.1.: Schema of the proposed architecture solution for the task of real-time multi-modal 3D object detection. Note this is only an example for the fusion of 3 classes, but it can be generalized easily to any N number of classes

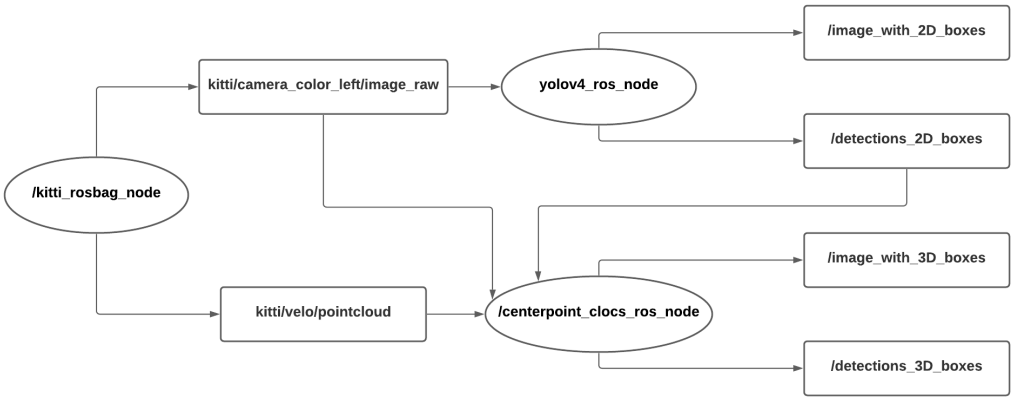


Figure 4.2.: *ROS* graph communication for the proposed solution. The oval shapes represent nodes and the rectangles topics. An arrow entering/leaving a node that connects a node to a topic means this node is a Subscriber/Publisher of that particular topic

5. Experiments

This Chapter presents experiments of the architecture solution proposed in Chapter 4 for real-time multi-modal 3D object detection. Quantitative and qualitative results of experiments conducted on the *KITTI* dataset are provided as well as discussion of its implications. Because the *CenterPoint* performance behaves differently depending on the number of classes of interest, here one model trained on 3 classes and another one trained on 4 classes are considered with analysis about the accuracy of *CenterPoint* alone vs. after the fusion with *CLOCs*, inference times, FPS and latency introduced. All experiments have been carried out on a single GPU NVIDIA RTX 3090 with 24 GB of memory. Also, a short summary of trials and errors on a small set of training data collected from the *Providentia++* sensors is given to highlight pitfalls and complex aspects when trying to build a custom multi-modal dataset.

5.1. Quantitative results

5.1.1. *YOLOv4* training and runtime

A *YOLOv4* model implemented in the *Darknet* framework has been trained on all relevant classes of *KITTI* with input resolution 512x512, batch size 64, momentum=0.949, decay=0.0005, learning rate 0.001 and maximum iterations 9600. The *KITTI* image dataset and its labels were divided into 5001 for training and 2480 for validation. The graph in Fig. 5.1 shows the training progress of this *YOLOv4* model. The loss converged steadily to a value below 1.5 whereas the mAP in the evaluation phase increased until 93 %. Perhaps these values could have been even better with more training iterations, but it seems the gains would have been insignificant and for the task at hand the model performance is good enough. Below, some more detailed metrics statistics:

- Class 'Pedestrian', AP = 78.49% (TP = 1066, FP = 181)
- Class 'Car', AP = 96.65% (TP = 9032, FP = 477)
- Class 'Cyclist', AP = 90.84% (TP = 443, FP = 56)
- Class 'Van', AP = 97.89% (TP = 911, FP = 42)

- Class 'Truck', AP = 98.66% (TP = 360, FP = 19)
- Class 'Tram', AP = 95.54% (TP = 155, FP = 13)
- For confidence threshold = 0.25, precision = 0.94, recall = 0.91, F1-score = 0.93,
  TP = 11967, FP = 788, FN = 1136, average IoU = 82.22 %
  IoU threshold = 50 %, used Area-Under-Curve for each unique Recall.
  Mean average precision (mAP@0.50) = 0.930120, or 93.01%

Aside from the hardware power, the inference time for a 2D detector depends among other factors on the input resolution and number of classes. On the RTX 3090 and having cuDNN (the NVIDIA CUDA® Deep Neural Network library for high-performance GPU acceleration) plus the fp16, aka half-precision, setting enabled and image resolution 512x512, this *YOLOv4* model takes 18 ms or 56 FPS on average for doing inference with a single image.

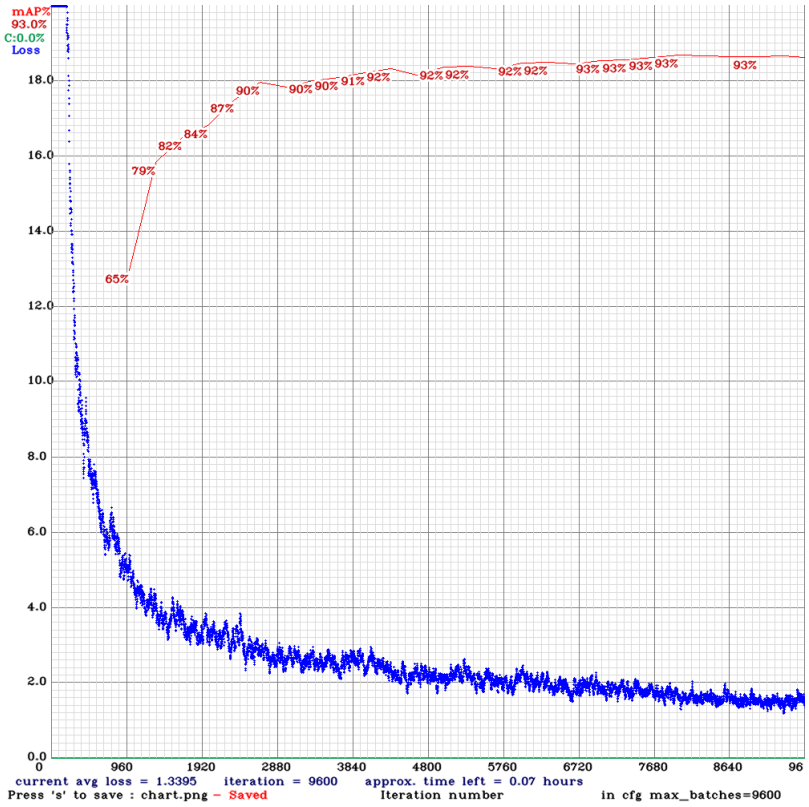


Figure 5.1.: Evolution of the training loss and mAP evaluation of the *YOLOv4* model trained on all classes of the *KITTI* dataset

5.1.2. *CenterPoint* trained on 3 classes + *CLOCs*

A *CenterPoint* model has been trained on the classes 'Car', 'Pedestrian' and 'Cyclist' of the *KITTI* dataset with voxel size [0.16, 0,16, 4], point cloud range = [0, -39.68, -3, 69.12, 39.68, 1] (units in meters), ground-truth sampling distribution 'Car' = 15, 'Pedestrian' = 20, 'Cyclist' = 20, batch size of 3, learning rate 0.003 and Adam optimizer. In this setting the size of the predicted heatmap is H=248, W=216, so the amount of raw boxes for one class generated by one detection head is HxW = 53568. The performance of the model can sometimes drastically change from one training epoch to the next one, therefore every two epochs the network was set to evaluation mode to get the evaluation metrics on the validation split of *KITTI*. The best overall results were obtained at epoch 92. From this model as the baseline, the *CLOCs* Fusion Layer was trained on top with the same settings as before, except the batch size was 1 and ground-truth sampling is disabled since the data augmentation would not be consistent with the stored 2D detections from *YOLOv4*. The best performance for *CLOCs* was attained at epoch 8. Table 5.1 shows the 3D and BEV Average Precision results of *CenterPoint* alone compared to the gains achieved by fusion with *CLOCs* on the *KITTI* validation set. Table 5.2 shows a breakdown of the inference time with half-precision enabled.

Table 5.1.: 3D and BEV Average Precision results of *CenterPoint* and *CenterPoint + CLOCs* on the validation set of *KITTI* for the classes 'Car', 'Pedestrian', 'Cyclist'. Both the old Recall 11 and the new Recall 40 *KITTI* evaluation metrics are presented

	Recall 11					
Method	Car 3D AP			Car BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	80.92	72.13	66.46	87.12	84.52	78.19
CenterPoint + CLOCs	81.91	73.39	67.31	87.57	85.58	78.63
Delta	**+0.99**	**+1.26**	**+0.85**	**+0.45**	**+1.06**	**+0.44**

	Recall 40					
Method	Car 3D AP			Car BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	82.29	72.51	69.07	90.05	84.57	82.47
CenterPoint + CLOCs	82.53	72.72	70.24	90.71	85.21	83.10
Delta	**+0.24**	**+0.21**	**+1.17**	**+0.66**	**+0.64**	**+0.63**

Method	Recall 11					
	Pedestrian 3D AP			Pedestrian BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	44.21	39.62	37.53	51.26	46.10	42.22
CenterPoint + CLOCs	54.17	48.35	42.41	58.65	52.26	51.25
Delta	**+9.96**	**+8.73**	**+4.88**	**+7.39**	**+6.16**	**+9.03**

Method	Recall 40					
	Pedestrian 3D AP			Pedestrian BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	41.14	36.80	33.65	48.71	43.80	40.34
CenterPoint + CLOCs	51.96	46.91	41.67	59.23	53.72	48.19
Delta	**+10.82**	**+10.11**	**+8.02**	**+10.52**	**+9.92**	**+7.85**

Method	Recall 11					
	Cyclist 3D AP			Cyclist BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	77.73	55.88	54.50	80.35	61.50	57.75
CenterPoint + CLOCs	82.77	65.81	65.72	85.46	68.16	67.86
Delta	**+5.04**	**+9.93**	**+11.22**	**+5.11**	**+6.66**	**+10.11**

Method	Recall 40					
	Cyclist 3D AP			Cyclist BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	80.26	57.08	54.06	84.10	61.10	58.17
CenterPoint + CLOCs	86.07	64.87	62.64	88.87	69.65	65.07
Delta	**+5.81**	**+7.79**	**+8.58**	**+4.77**	**+8.55**	**+6.90**

Table 5.2.: Runtime Performance on GPU RTX 3090 divided into blocks of the pipeline, in total for all 3 classes in this experiment. Averaged over 1000 frames

CenterPoint inference	*CLOCs* prep.	*CLOCs* fusion	Final NMS	Total
25 ms	10 ms	5 ms	3 ms	43 ms

From Table 5.1 it is obvious that fusion with *CLOCs* improved the 3D and BEV Average Precision results of *CenterPoint* for all classes, especially for 'Pedestrian' and 'Cyclist' by a large margin. Table 5.2 indicates that *CenterPoint* alone before NMS runs at 25 ms or 40 FPS, the latency introduced by *CLOCs* in this experinment is 20 ms of which 66.6% is due to the preprocessing functions to prepare the data for *CLOCs* (this refers to the projection of the raw 3D boxes into the image plane and building the 3 input tensors for fusion). The forward pass through all 3 *CLOCs* fusion instances is quite fast, only 5 ms, since one such *CLOCs* instance is a tiny neural network. Notice the final NMS to obtain the definitive 3D boxes predictions, which takes 4 ms, should not be considered as a fusion-related latency because this step was actually skipped in the *CenterPoint* inference. Thus, the total inference time is 43 ms or 23.3 FPS.

5.1.3. *CenterPoint* trained on 4 classes + *CLOCs*

A *CenterPoint* model has been trained on the classes 'Car', 'Pedestrian', 'Cyclist' and 'Van' of the *KITTI* dataset with voxel size [0.16, 0,16, 8], point cloud range = [0, -39.68, -4, 96, 39.68, 4] (units in meters), ground-truth sampling distribution 'Car' = 15, 'Pedestrian' = 25, 'Cyclist' = 25, 'Van' = 50, batch size of 3, learning rate 0.003 and Adam optimizer. In this setting the size of the predicted heatmap is H=248, W=300, so the amount of raw boxes for one class generated by one detection head is HxW = 74400. The performance of the model can sometimes drastically change from one training epoch to the next one, therefore every two epochs the network was set to evaluation mode to get the evaluation metrics on the validation split of *KITTI*. The best overall results were obtained at epoch 26. From this model as the baseline, the *CLOCs* Fusion Layer was trained on top with the same settings as before, except the batch size was 1 and ground-truth sampling is disabled since the data augmentation would not be consistent with the stored 2D detections from *YOLOv4*. The best performance for *CLOCs* was attained at epoch 20. Table 5.3 shows the 3D and BEV Average Precision results of *CenterPoint* alone compared to the gains achieved by fusion with *CLOCs* on the *KITTI* validation set. Table 5.4 shows a breakdown of the inference time with half-precision enabled.

Table 5.3.: 3D and BEV Average Precision results of *CenterPoint* and *CenterPoint + CLOCs* on the validation set of *KITTI* for the classes 'Car', 'Pedestrian', 'Cyclist', 'Van'. Both the old Recall 11 and the new Recall 40 *KITTI* evaluation metrics are presented

	Recall 11					
Method	Car 3D AP			Car BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	57.54	50.91	46.83	72.83	65.84	66.77
CenterPoint + CLOCs	67.14	59.39	55.14	82.04	74.56	75.22
Delta	**+9.60**	**+8.48**	**+8.31**	**+9.21**	**+8.72**	**+8.45**

	Recall 40					
Method	Car 3D AP			Car BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	56.46	47.74	47.34	71.49	65.31	64.30
CenterPoint + CLOCs	66.83	57.25	55.00	81.84	74.98	73.84
Delta	**+10.37**	**+9.51**	**+7.66**	**+10.35**	**+9.67**	**+9.54**

	Recall 11					
Method	Pedestrian 3D AP			Pedestrian BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	42.72	38.74	35.49	47.38	44.90	42.30
CenterPoint + CLOCs	56.48	50.36	48.15	62.18	58.60	53.98
Delta	**+13.76**	**+11.62**	**+12.66**	**+14.90**	**+13.70**	**+11.68**

	Recall 40					
Method	Pedestrian 3D AP			Pedestrian BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	39.63	35.88	32.97	45.85	41.98	40.00
CenterPoint + CLOCs	54.98	49.07	45.35	62.90	57.86	54.45
Delta	**+15.35**	**+13.19**	**+12.38**	**+17.05**	**+15.88**	**+14.45**

Method	Recall 11					
	Cyclist 3D AP			Cyclist BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	75.94	54.21	50.94	78.15	56.95	55.28
CenterPoint + CLOCs	85.19	59.38	58.90	85.98	67.61	59.98
Delta	**+9.25**	**+5.17**	**+7.96**	**+7.83**	**+10.66**	**+4.7**

Method	Recall 40					
	Cyclist 3D AP			Cyclist BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	77.22	53.56	50.47	78.77	56.78	54.41
CenterPoint + CLOCs	84.54	60.03	57.64	87.28	65.08	60.59
Delta	**+7.32**	**+6.47**	**+7.17**	**+8.51**	**+8.30**	**+6.18**

Method	Recall 11					
	Van 3D AP			Van BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	22.83	16.80	15.35	28.90	21.27	20.47
CenterPoint + CLOCs	36.58	23.02	24.55	49.36	34.06	35.09
Delta	**+13.75**	**+6.22**	**+9.2**	**+20.46**	**+12.79**	**+14.62**

Method	Recall 40					
	Van 3D AP			Van BEV AP		
	Easy	Mod.	Hard	Easy	Mod.	Hard
CenterPoint	22.27	14.40	13.04	28.41	18.88	18.35
CenterPoint + CLOCs	37.14	22.67	21.40	49.65	32.77	30.63
Delta	**+14.87**	**+8.27**	**+8.36**	**+21.24**	**+13.89**	**+12.28**

Table 5.4.: Runtime Performance on GPU RTX 3090 divided into blocks of the pipeline, in total for all 4 classes in this experiment. Averaged over 1000 frames

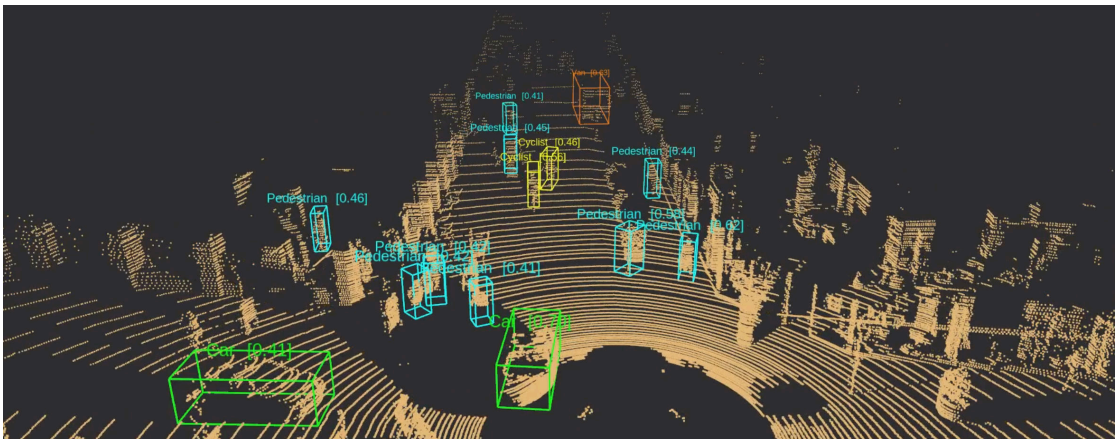
CenterPoint inference	*CLOCs* prep.	*CLOCs* fusion	Final NMS	Total
30 ms	15 ms	10 ms	5 ms	60 ms

From Table 5.3 we can see again that fusion with *CLOCs* improved the 3D and BEV Average Precision results of *CenterPoint* for all classes, even by a larger margin than in the previous experiment with huge gains over +10. From checking the evaluation history of the *CenterPoint* models it may happen that in one evaluation iteration the results are more favorable for a certain class or classes, i.e. the evaluation results dictate the network should perform in the current iteration better for 'Car' and 'Cyclist', but not as good for 'Pedestrian' as compared to past iterations or vice versa. This is sometimes a matter of letting the network train for more epochs to achieve stability, but also often one has no other choice to simply select a checkpoint that prioritizes the theoretical performance of a class / classes and sacrifice the accuracy of the others unfortunately. We can also see that in this experiment with 4 classes the results of *CenterPoint* alone are worse than trained only on 3 classes, this degradation is highly noticeable for the 'Car' class. In fact, it is a general behaviour of a network to perform worse the more classes it is trained on. Table 5.4 indicates that *CenterPoint* alone before NMS runs at 30 ms or 33.3 FPS, the latency introduced by *CLOCs* in this experinment is 25 ms of which 60% is due to the preprocessing functions to prepare the data for *CLOCs* (this refers to the projection of the raw 3D boxes into the image plane and building the 4 input tensors for fusion). The forward pass through all 4 *CLOCs* fusion instances takes 10 ms. The final NMS to obtain the definitive 3D boxes predictions, which takes 5 ms, should not be considered as a fusion-related latency because this step was actually skipped in the *CenterPoint* inference. Thus, the total inference time is 60 ms or 16.7 FPS.

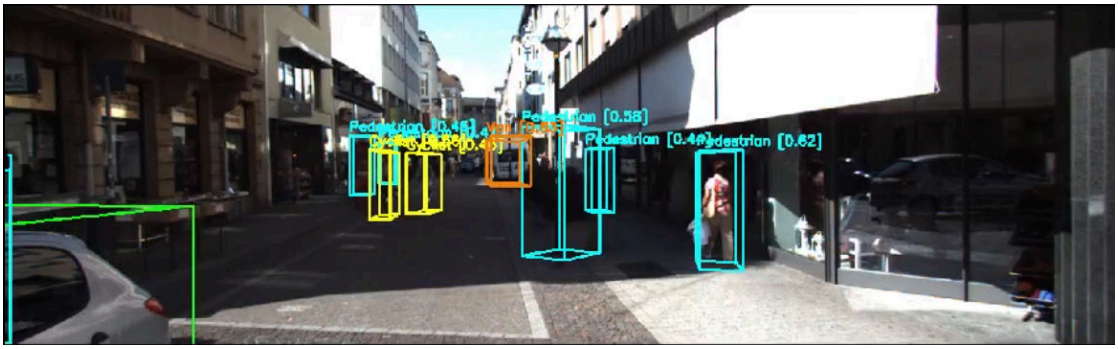
Notice that because *YOLOv4* runs in parallel with *CenterPoint*, the inference time of *YOLOv4* should not count as part of the total 3D detection plus fusion runtime. In this second experiment the total FPS got reduced by 6.6 compared to the first experiment. This is expected since one class more was included, consequently one more input tensor and a *CLOCs* instance for fusion are required. Because of the larger point cloud range in this second experiment, there are more raw 3D boxes generated in each detection head than before, 4x74400 = 297600 vs 160704 = 3x53568, therefore, both the inference of *CenterPoint* and the preprocessing function of detection candidates for *CLOCs* with more than twice the number of boxes take logically longer to be computed.

5.2. Qualitative results

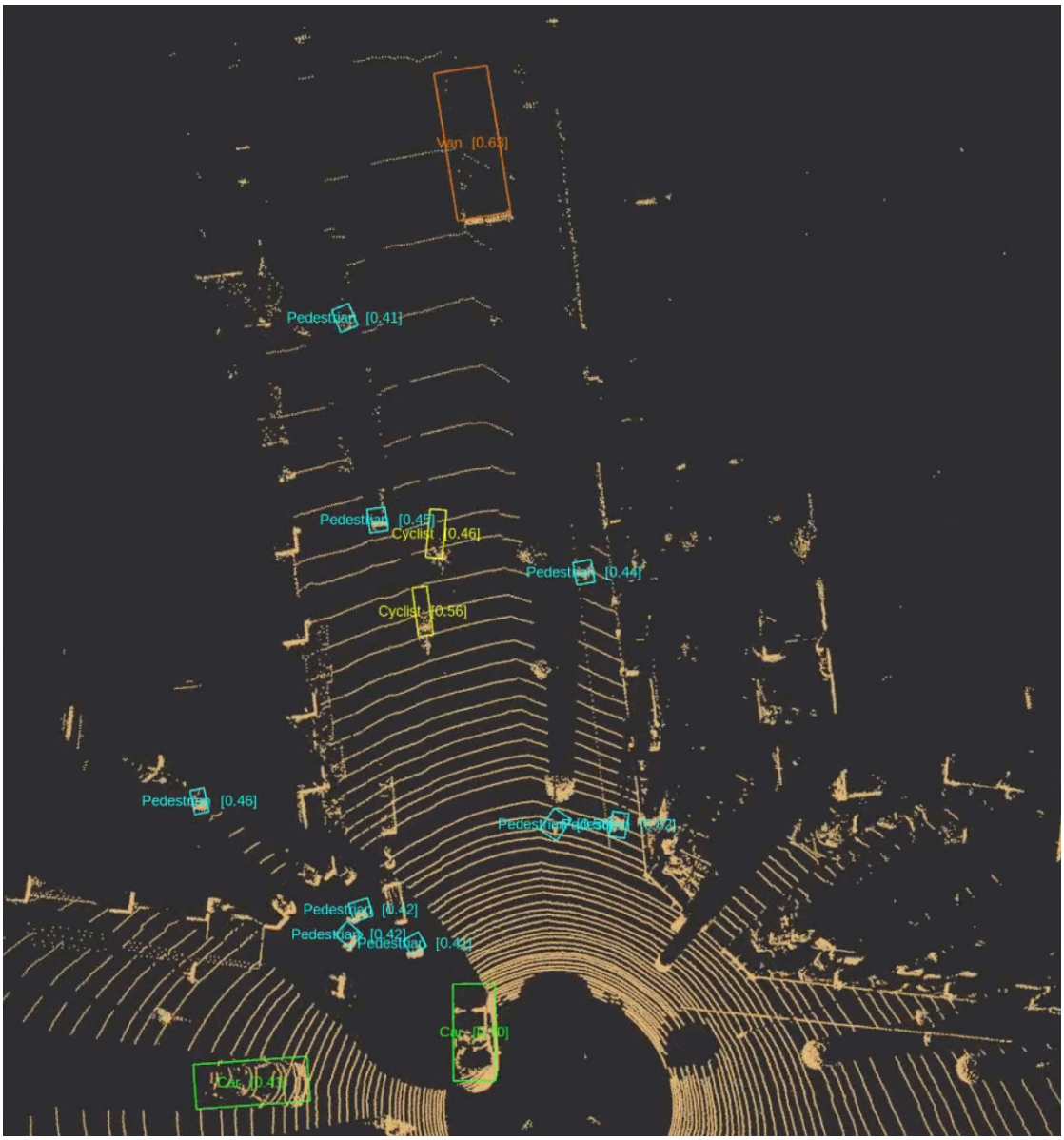
In this section the spotlight is placed at some of the pros and cons found in the predictions generated by the proposed multi-modal 3D object detection solution. It should be clear that the 3D output boxes come naturally from the reasoning of the neural network in the point cloud domain, the 2D detections in images only assist and contribute to a refinement of the primary inference results, see Fig. 5.2. Nevertheless, to grasp the analysis more intuitively in a visual way, the images with the projected 2D and 3D boxes are utilized.



(a) 3D detections displayed in the point cloud from the ego-vehicle perspective



(b) The same 3D boxes from the point cloud before are projected to the corresponding image. Yellow, cyan, green and orange boxes are assigned to the classes 'Cyclist', 'Pedestrian', 'Car' and 'Van' respectively



(c) 3D detections displayed in the point cloud, Bird-Eye-View (BEV)

Figure 5.2.: 3D boxes in different representations predicted by the proposed solution, *CenterPoint + YOLOv4 + CLOCs*

In the following figures always the top image has the 3D boxes predicted by *CenterPoint* alone before fusion, the bottom image has the 2D detections of *YOLOv4*, and the middle image has the 3D boxes predicted by *CenterPoint + CLOCs* after fusion with the 2D boxes from *YOLOv4*. The most noteworthy impact of the fusion process is the reduction of a lot of undesired False Positives (FPs) that were initially detected because in the point cloud these objects have a similar shape of the category that was mistakenly confused, Fig. 5.3. Another benefit is that the 2D detections can help to improve the recall of the 3D detector, Fig. 5.4 and also increase the confidence scores of the found objects, Fig. 5.5.

Despite these positive effects, fusion has some downsides too, such as the suppression of relevant true positives (TPs) when *YOLOv4* misses those detections in the image plane, Fig. 5.6, or the label predicted by *YOLOv4* for one object does not match the same label output of *CenterPoint*, Fig. 5.7. A likely explanation for this phenomenon is that in the absence of legitimate 2D detections intersecting a 3D box with identical semantic meaning, *CLOCs* tends to discard the TP regardless of its score, or in other words, *CLOCs* uses the 2D detections as proof of evidence to confirm the truthfulness of the 3D detection candidates from *CenterPoint*. When an object is good located both in 3D and 2D, but with contradictory class labels, the 2D detector can then cause a missclasification of the object that would have not occurred before fusion, Fig. 5.8. These artifacts stress the importance of having a 2D detector as good as possible, since the accuracy of the *YOLOv4* model and the quality of its detections play an essential role in the final 3D boxes predictions. Once trained, the Fusion Layer with all *CLOCs* instances in charge of their respective classes learned some criteria to leverage the 2D detections to either increase or diminish the confidence of the 3D boxes. We can observe that in general the behaviour of the Fusion Layer is to enhance the 3D boxes predictions if and only if they match the 2D boxes geometrically and semantically, otherwise there is a potential risk that even a TP in the camera field-of-view gets deleted after the final 3D Non-Maximum-Suppresion. But how *CLOCs* solves the fusion problem internally cannot be determined beforehand with exact rules as there are no hard-coded thresholds or a hard decision boundary. Being a neural network *CLOCs* adapts dynamically during fusion to give flexible weights to the relevance of 2D and 3D detection candidates. The interested reader can find more descriptions of qualitative results and edge-cases in the Appendix.

Figure 5.3.: A significant amount of cyclists and pedestrians FPs are removed from the scene after fusion

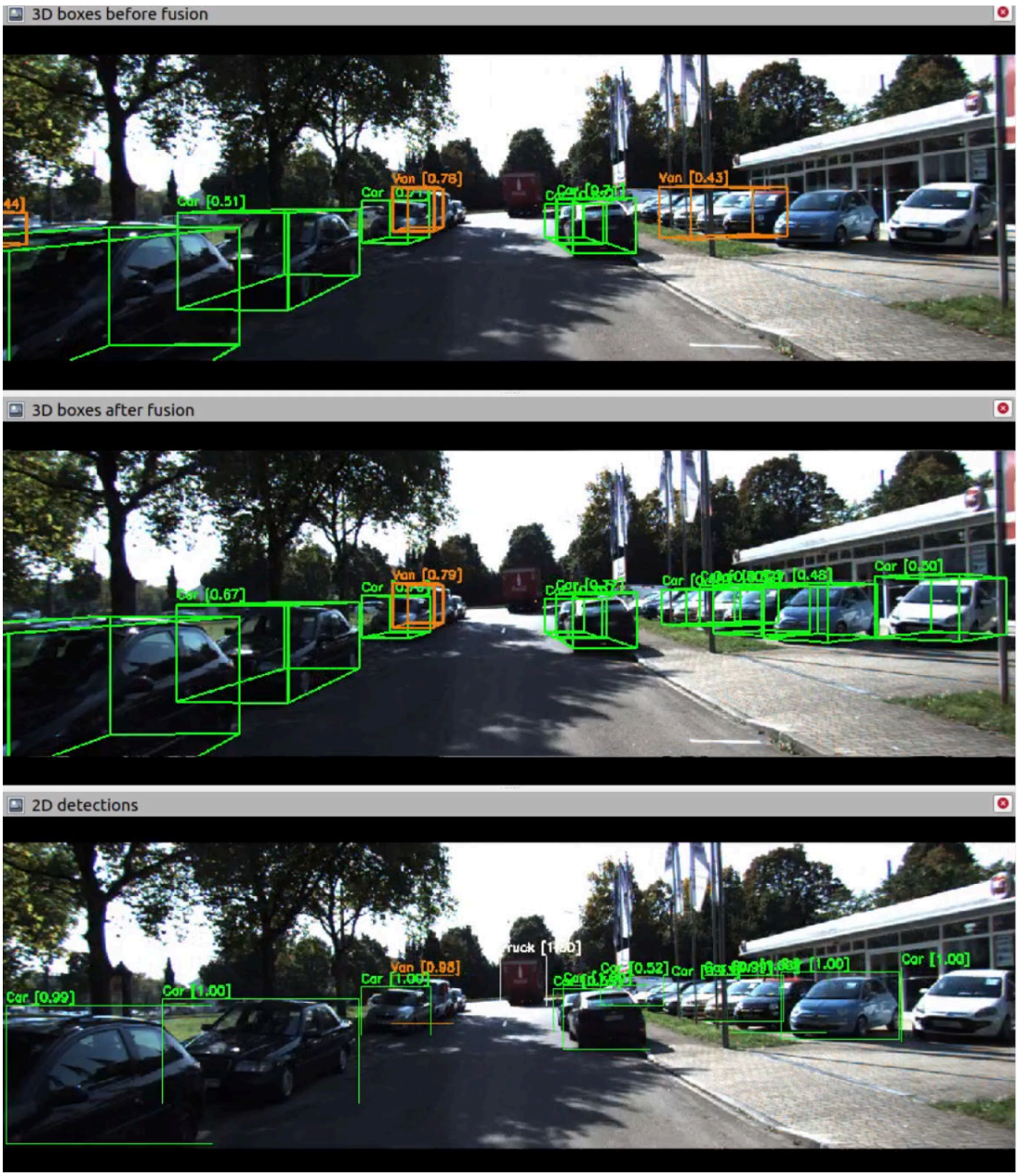


Figure 5.4.: The vehicles on the right that were not detected or misclassified are recovered after fusion

Figure 5.5.: Fusion enhances the confidence scores of the predicted objects, *CenterPoint* alone estimates 'Car' [0.43], 'Van' [0.58]. *CLOCs* boosted the scores to be 'Car' [0.55], 'Van' [0.70] and deleted the false pedestrians on the right confused with plants before fusion

Figure 5.6.: The 3D box of the TP van in the middle predicted by *CenterPoint*, was suppresed after fusion because *YOLOv4* failed to detect it

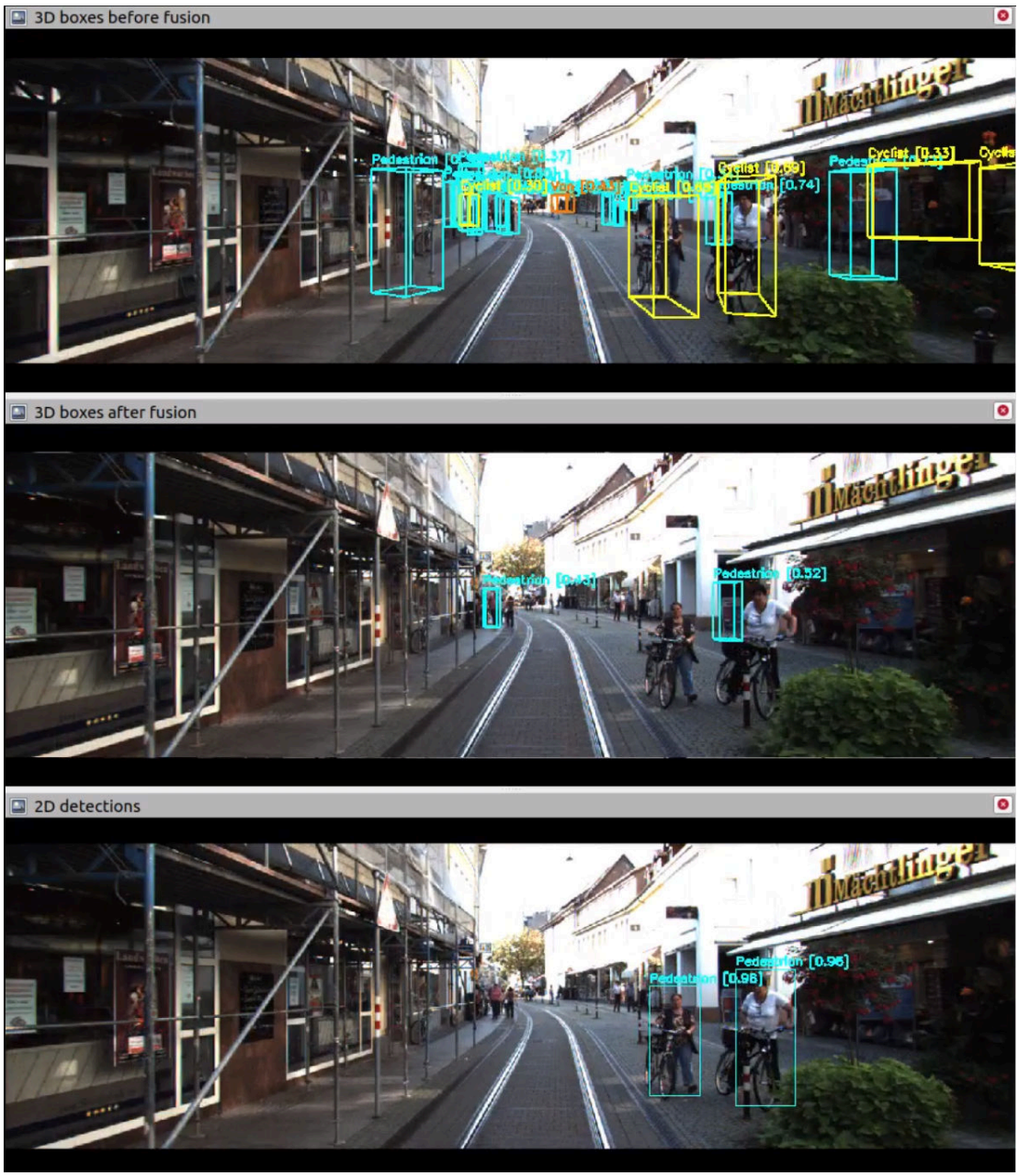


Figure 5.7.: FPs are removed on the sides, but the two cyclists are not identified after fusion because *YOLOv4* classified these as pedestrians. *CLOCs* then considered the TPs cyclists as FPs given the mismatch in the 3D and 2D predicted labels. Though it is questionable if a person not riding, but just holding a bike should be classified as either 'Pedestrian' or 'Cyclist'

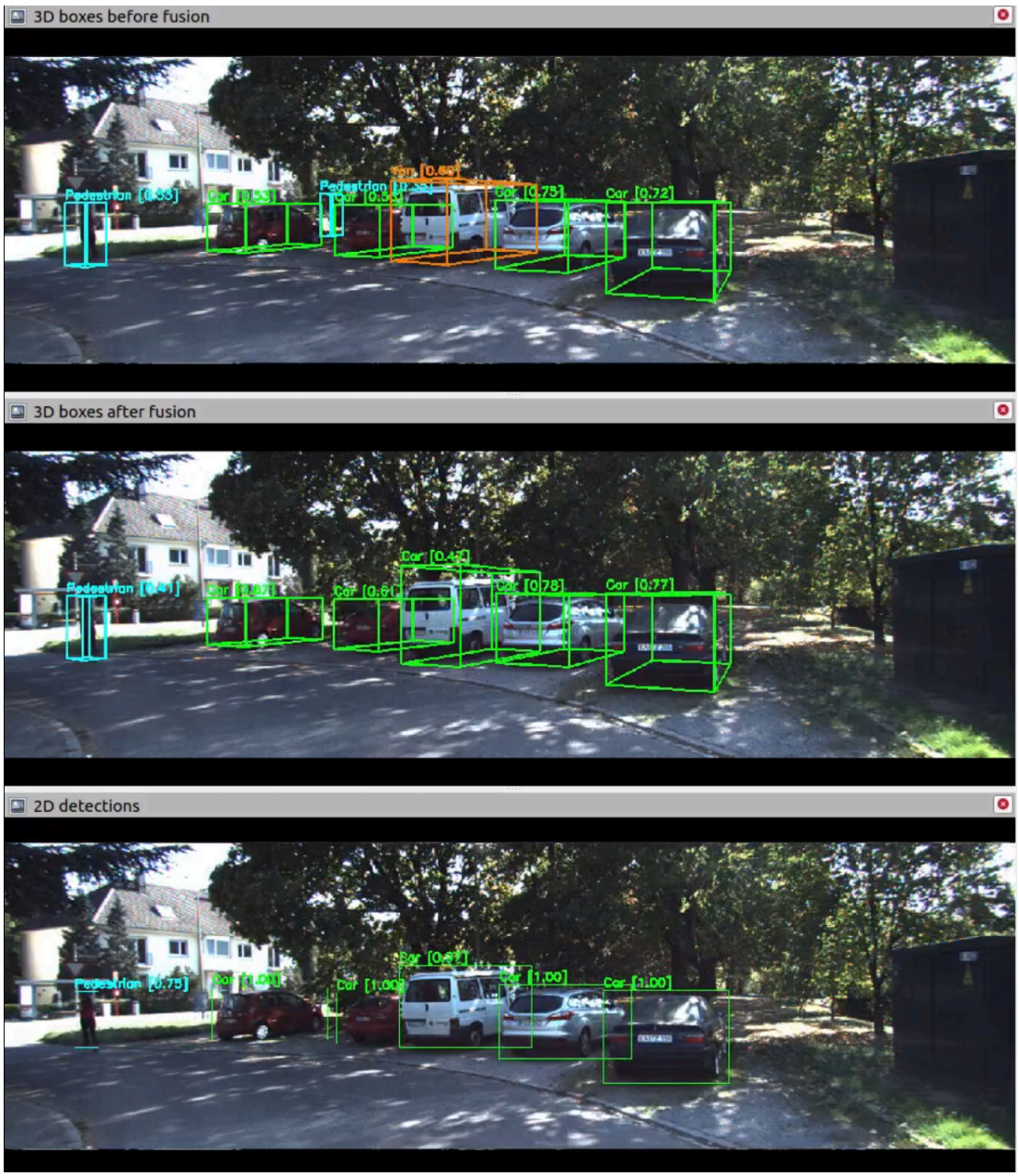


Figure 5.8.: Originally *CenterPoint* classified correctly as 'Van' and predicted better the size of the white large vehicle in the middle. Since *YOLOv4* labeled this vehicle as 'Car' instead, after fusion the category of its 3D box was wrongly changed to 'Car' and its size was a little overestimated.

5.3. Trials and errors

As mentioned in Chapter 1 the context frame of this thesis was the *Providentia++* Intelligent Infrastructure System and in principle the goal was to conduct experiments on data collected from the *Providentia++* LiDAR and camera sensors. A dataset consisting of 300 point cloud frames recorded with a LiDAR Ouster OS1-64 was labeled in the annotation tool 3D BAT [39] with classes 'Car', 'Van', 'Pedestrian' and 'Truck'. In Fig. 5.9 the annotation of one example training frame in the *3D BAT* environment is shown. A *CenterPoint* toy model was trained on this very small dataset with satisfactory results qualitatively speaking. Fig. 5.10 displays the 3D boxes predicted by *CenterPoint* on a testing point cloud. Rather than writing an API from scratch to train a new dataset, it is advisable to modify the existent APIs written for one of the most popular datasets, practically all open-source networks for 3D object detection have worked with either *KITTI*, *nuScenes* or *Waymo*. However, the tiny *Providentia++* dataset we labeled would make any 3D detector model fall into overfitting and is not sufficient to train a high-quality 3D detector for deployment on the edge-computing unit at the highway. We experienced that annotating ourselves point cloud frames for 3D object detection is a quite time-consuming procedure that takes more effort, concentration and longer elaboration than 2D box image annotation. It is subject to mistakes too, e.g. because the point cloud generated by the Ouster LiDAR was of low density and so sparse to distinguish objects perfectly in some frames, we often had to approximately guess the orientation, location and dimensions of the ground-truth box for a vehicle, so in some sense the ground-truth annotations were already injecting a bias and false assumed information. The point cloud frames were recorded with an Ouster LiDAR mounted on a tripod pointing to the highway A9, yet the recording was done without a strict technical verification protocol of the exact relative position of the LiDAR sensor with respect to the camera, which is a first step for the extrinsic calibration of the sensors. Although perhaps even just measuring the relative position of camera and LiDAR would have been not enough because manual measurements are prone to severe estimation errors. Conversely, training a *YOLOv4* model on images from cameras of *Providentia++* was rather easy, see Fig. 5.11 that shows the 2D boxes predicted by *YOLOv4* on one image taken from the highway A9.

Later on when we became aware that camera-LiDAR calibration was a fundamental required step towards multi-modal data annotation and fusion, all LiDAR devices were occupied in their installation to be fixed in the *Providentia++* stations, a process that also went through unforeseen delays. Only one LiDAR device of the brand Valeo and a spare Basler camera were left and available to the students for indoor testing. The idea was to perform an extrinsic calibration of these sensors in the laboratory area mounting them

on a tripod, then translating the same tripod configuration and sensors relative position to the highway to record some new extrinsic calibrated data. A novel target-based method that optimizes the selection of samples for robust LiDAR-camera calibration [40] was employed with the Valeo and Basler sensors. This method required necessarily the ring-channel of the point cloud that is published by other LiDARs like the Ouster ones or the higher-quality Velodyne devices. The Valeo LiDAR does not provide this feature by default, thus the ring-information for each point was calculated and added manually to each point cloud scan done with the Valeo LiDAR. Unfortunately we obtained quite bad imprecise calibration results as the reprojection errors in pixels and millimeters of the point cloud into the images were too high to be any helpful for further data labelling experiments. The errors during the collection of image-point-cloud sample pairs for calibration were mostly due to the high noise of the Valeo LiDAR not being able to adjust and capture the chessboard target with fair planar shape, i.e. the points had too much spatial deviation from the actual surface making the plane look jagged and discontinuous in the point cloud. Since camera-LiDAR calibration is not the main scope of this thesis and given the time constraints and deadline for its development, we decided to present experiments on the *KITTI* dataset. Surely new students and the staff team of *Providentia++* will be able to solve the camera-LiDAR calibration challenge of the already installed sensors in the near future.

Another issue one has to consider is the transmission bandwith limit of the ports used to connect both the camera and LiDAR simultaneously because we experienced packet loss from the LiDAR if the camera was connected to the same Ethernet port of a conventional laptop. The packet loss causes the point cloud broadcasting to look frozen for a while and the refresh rate of the LiDAR is so reduced to be impractical to scan moving objects. A workaround to reduce this packet loss could be to utilize Ethernet-to-USB adapters to connect the devices to different ports. Equally important is the time synchronization of the camera and LiDAR, e.g. we struggled to realize that, while recording, a parameter needs to be correctly set up for the Valeo LiDAR to provide point clouds timestamped according to the *ROS* clock and not the internal hardware timestamp. On the contrary the Basler camera uses the *ROS* clock by default. Without synced timestamps a multi-modal dataset cannot be created, moreover the time difference between an image and the closest point cloud frame should be as minimal as possible to avoid data misalignment even if the sensors operate or the training sequences are collected at similar sampling frequencies.

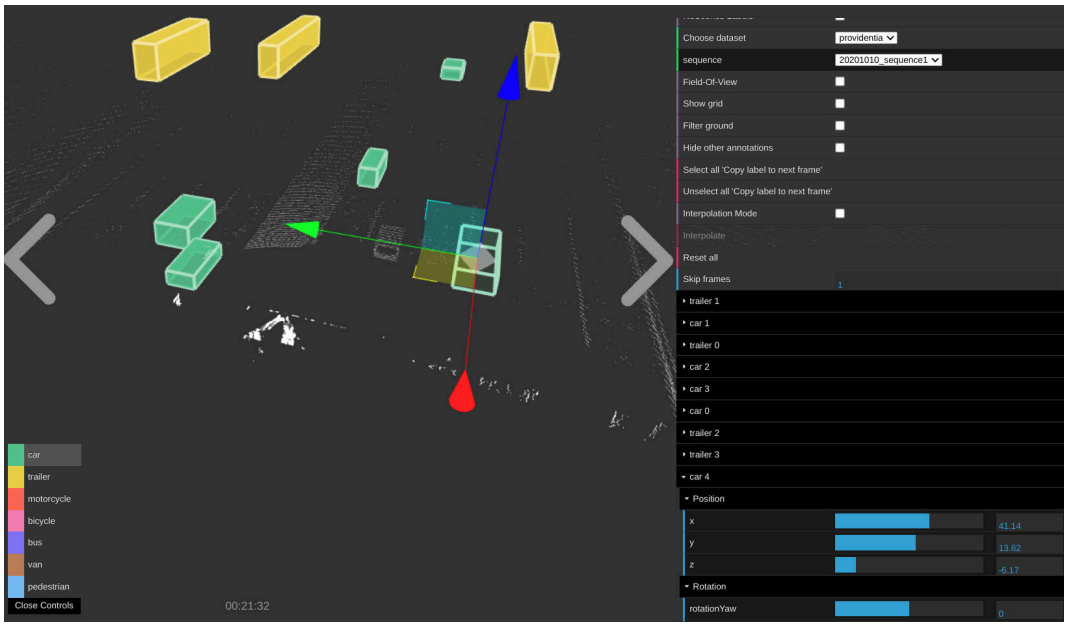


Figure 5.9.: *3D BAT* annotation-tool environment where in this example from the highway A9 in *Providentia++*, trucks are labelled with yellow bounding boxes and cars with green boxes

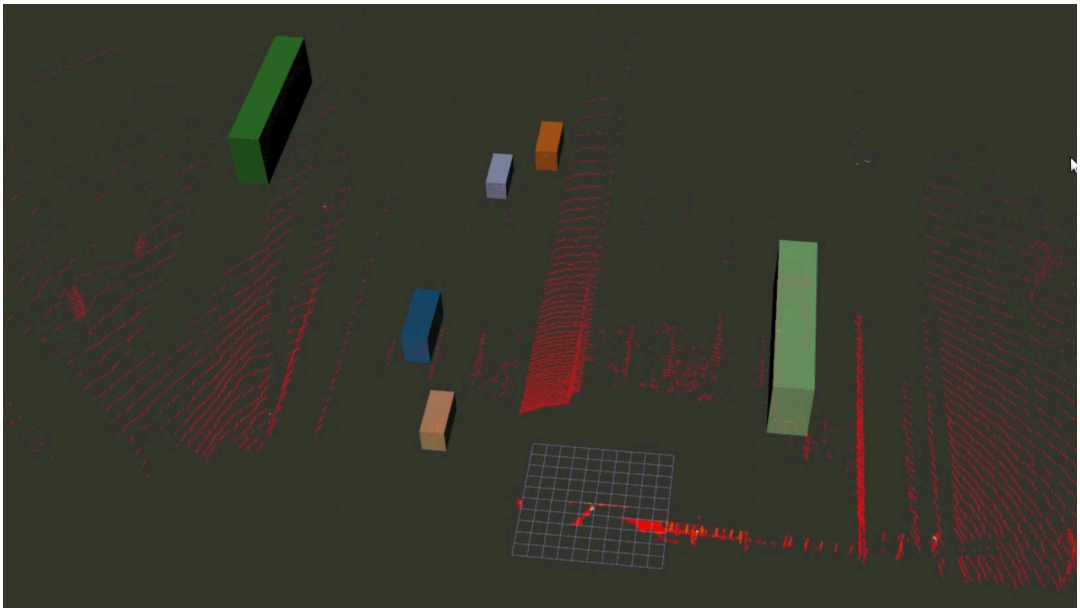


Figure 5.10.: 3D boxes of traffic participants predicted by a *CenterPoint* toy model on a test point cloud recorded from the highway A9 that is a portion of the testbed in *Providentia++*. Small boxes are cars and large boxes are trucks detected. The red point cloud is the scan of the highway.

Figure 5.11.: 2D boxes predicted by *YOLOv4* on an image from the highway A9 that is a portion of the testbed in *Providentia++*, where the classes 'Car', 'Van', 'Truck' and 'Trailer' are detected

6. Conclusions

This thesis has addressed the implementation of a real-time multi-modal 3D object detection pipeline based on the state-of-the-art neural networks *CenterPoint*, *YOLOv4* and *CLOCs*, which are specialized in the subtasks of LiDAR-based 3D detection, camera-based 2D detection and camera-LiDAR sensor late-fusion respectively. The *CLOCs* method extended here for handling fusion of multiple classes at once from a single *CenterPoint* model has proven to drastically enhance the accuracy of *CenterPoint* according to the evaluation results on the *KITTI* validation set. Surprisingly, the *PointPillars*-version of *CenterPoint* trained on *KITTI* achieves a worse accuracy than the baseline *PointPillars* also trained on *KITTI*. After consultation with Tianwey Yin, the main creator of *CenterPoint*, he is aware of this issue and suspects that because in *KITTI* most objects to be detected are axis-aligned with the ego-vehicle frame, so it could be that an anchor-free detector like *CenterPoint* does not have an advantage over anchor-based networks. Moreover, the Average Precision (AP) metric in *KITTI* focus exclusively on the 3D IoU, where as in *nuScenes* - the first target dataset for which *CenterPoint* was designed - this AP metric is based on the 2D center distance on the ground plane, thus this more relaxed 2D-based metric could lead to better evaluation results. Even with the modest accuracy of *CenterPoint* on *KITTI*, this 3D detector has helped us to analyze what to expect from *CLOCs* for real-time multi-class fusion. Besides, Pang Su who is the main author of *CLOCs*, confirmed lately that his network does a very good job at improving the performance of *CenterPoint* in *nuScenes* as well. Although there have been some interesting works in this direction, [41], the fine-tuning of networks for 3D object detection across different datasets remains still today a challenge. The so-called transfer-learning technique in Deep Learning cannot be applied in 3D object detection as straightforward as in the 2D case. The 3D neural networks are influenced heavily by the inherent features of the training dataset, which could be very unique depending on the point cloud range and channels, characteristics of the LiDAR device used to generate the point cloud frames, different perspectives and city locations of the sensor recordings, etc. Therefore, if possible, it is always more convenient to collect a new big enough dataset and train a 3D detector from scratch for a custom application.

In terms of speed, the experiments showed that for 3 or 4 classes - different settings between these two tests apart from the class number - the proposed solution is able

to run at 23.3 or 16.7 FPS respectively, on a GPU RTX 3090. The experiment with 4 classes involved more than twice the amount of raw 3D boxes candidates for fusion, so it could also be considered as a quasi-experiment with 6 classes with the same configuration of the 3-classes experiment. Although there does not exist a fair comparison of other approaches, most multi-modal networks found in the literature run on average between 10 and 15 Hz or slower for one class only, thus, the proposed solution has an acceptable runtime considering the multi-class extension case. However, the inference speed depends on many factors like the point cloud range, voxel size, output stride of the 3D network backbone, number of classes, and of course the GPU hardware to name a few. Speaking of hardware, the proposed solution requires one very powerful GPU or two moderate GPUs to run the 2D and 3D detector in parallel for real-time inference, this can be perhaps a major constrain for low-budget scenarios. Qualitative results on *KITTI* have shown the main repercussion of camera-LiDAR fusion with *CLOCs* to be a great reduction of False Positives and improvement of the 3D boxes confidence scores, but *CLOCs* is not exempt from removing True Positive detections when the 2D detector fails to identify an object or when it predicts the wrong class label.

The *Providentia++* team can afford to employ the proposed solution in this thesis since the Data-Fusion Units distributed along the testbed count with several modern fast GPUs. Assuming the 2D detector performance is excellent, the proposed solution in this thesis could boost particularly the detection of pedestrians, cyclists and vehicles in the far range. There could be a risk of missing detections after fusion on the highway when there is a traffic jam or the vehicles appear very occluded or cluttered in the camera image. Therefore, it is suggested the proposed solution would be more suitable for the Garching intersection area, where the Bird-Eye-View from above is clearer and pedestrians and cyclists transit more frequently. If the inference time cost is too much of a burden, the *CLOCs* method could be used with only a subset of classes, the less common ones, since these might have the most potential better gains from fusion. As for camera-LiDAR calibration, it would help to first augment the resolution, i.e. point density, and apply a denoising algorithm on the point cloud scans done by the Ouster and Valeo LiDARs. A manual target-based procedure seems impractical to be done in the middle of the road, thus, carrying out a target-less online automatic camera-LiDAR calibration such as the recent [42] or [43] would be extremely beneficial to compose a high-quality synchronized multi-modal *Providentia++* dataset.

7. Future Work

- **Speed-related improvements**:
  If less precision is required or certain portions of the scene are not within the region of interest for detection, a *CenterPoint* model with narrower point cloud range, bigger voxel size and smaller output stride of the backbone neck can be eonfigured to achieve higher speed because this will create much less raw 3D boxes predictions. In this regard, the amount of 3D boxes preprocessed and passed to the Fusion Layer could be reduced to increase the speed, e.g. take only the 3D boxes for which their score is bigger than a 0.1 threshold since most raw 3D boxes have actually very low scores close to zero. But this would require an adaptation of the heatmap loss function used for training the *CLOCs* instances as the number of 3D boxes predicted by a detection head in *CenterPoint* will fluctuate for every frame. Furthermore, the inference time may decrease if the *CenterPoint* model is exported to ONNX (Open Neural Network Exchange) format and accelerated with TensorRT as developed in [44].

- **Accuracy-related improvements**:
  The accuracy of the 2D detector has a crucial impact in the final predictions after fusion. Being *CLOCs* a late-fusion method, the positive side is that *YOLOv4* can be replaced by a more accurate and optimized detector like the recent *PP-YOLOv2* with little effort. In fact, it would be enlightening to conduct ablation studies after fusion with more advanced 2D detectors and bigger image resolutions other than 512x512 employed in this thesis. As for *CenterPoint*, it would be worth investigating what was exposed in the *MoCa* paper that switching the backbone *PointPillars* with *RegNetX* [45] brings great improvements. The 3D detector could be upgraded to a new version named *CenterPoint++* which ranked on a high position in the real-time 3D detection challenge of Waymo. This *CenterPoint++* runs with *VoxelNet* backbone, so it is expected to be more accurate but slower than *PointPillars*. Another modification could be replacing *VoxelNet* in *CenterPoint* with the backbone and neck of *SECOND* that should be likewise accurate but faster. In addition, novel universal plug-and-play second-stage detection heads like *CT3D* [46] and *LiDAR R-CNN* [47] have been proposed to increase the accuracy of any 3D detector, thus it would be worth testing these in combination with *CenterPoint*.

A. Appendix

In the following pages supplementary qualitative results are provided. Again in the next figures always the top image has the 3D boxes predicted by *CenterPoint* alone before fusion, the bottom image has the 2D detections of *YOLOv4* and the middle image has the 3D boxes predicted by *CenterPoint + CLOCs* after fusion with the 2D boxes from *YOLOv4*. The figures are properly captioned and without specific order they are illustrative examples of these post-fusion effects: successful elimination of False Positives, improvement of recall and scores of the predicted 3D boxes, as well as suppression of True Positives as a consequence of the inaccuracy of the *YOLOv4* detector. An exceptional rare edge-case is described where a high 2D TP in the image due to a reflective surface creates an absurd 3D FP in the point cloud domain. Apart from that, one scene remarks the advantage in perception of LiDAR-based over camera-based object detection for recognition of a challenging target when under adverse light conditions the detection in the image becomes indeed very difficult.

Figure A.1.: The predicted 3D box as 'Van' is corrected after fusion with *CLOCs* thanks to the right classification in 2D. *YOLOv4* additionally detects a car on the left in the far range that the 3D detector was not able to locate

n

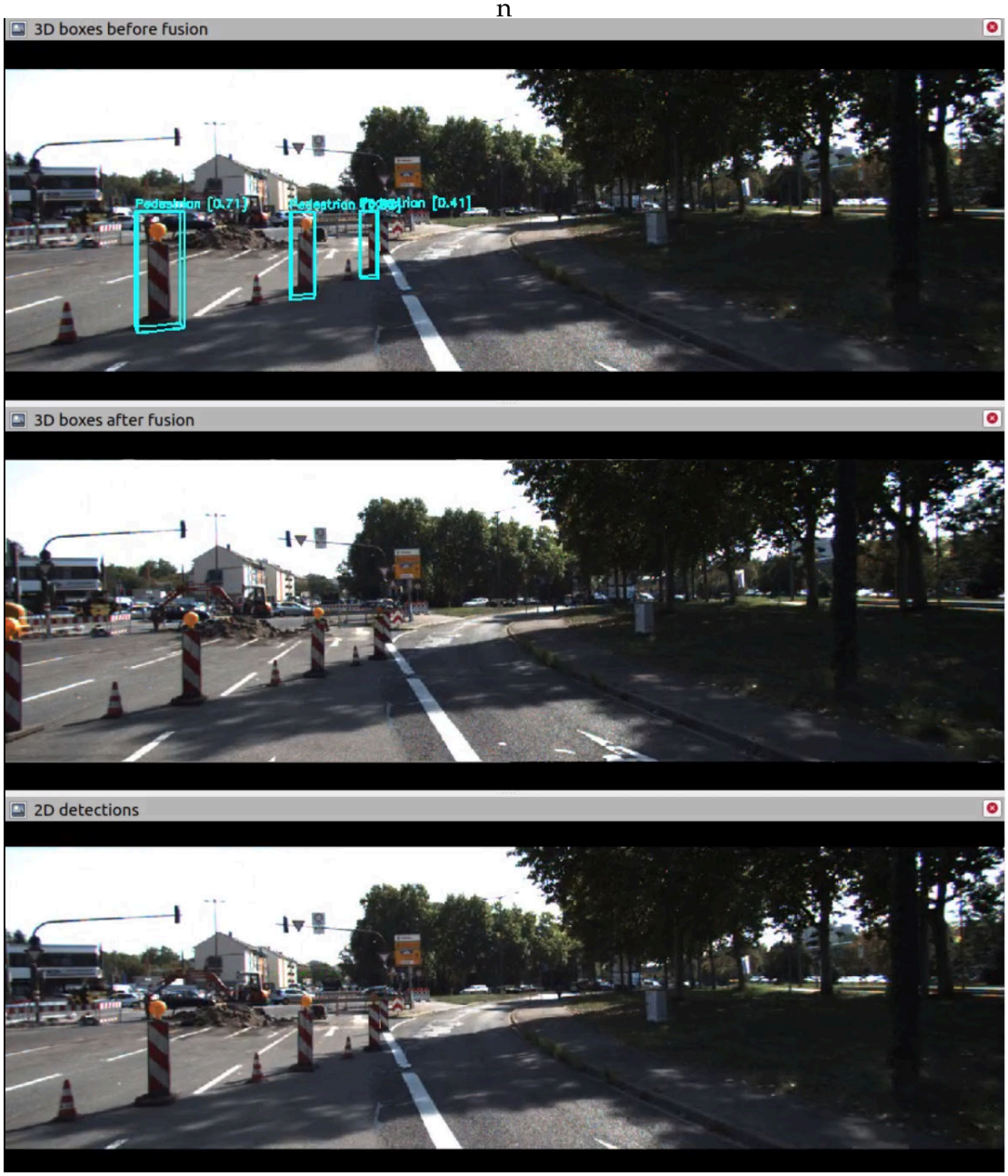


Figure A.2.: FPs pedestrians confused with poles of a construction site on the road are removed from the scene

Figure A.3.: The fusion method deletes FPs pedestrians confused with poles on the road boundaries

Figure A.4.: The detections in 2D helped to increase the recall to locate two cars on the left that were missed previously

Figure A.5.: The 2D detections from *YOLOv4* enabled the detection of more 3D car boxes, (only 2 before vs 6 after fusion) plus some other FPs were removed

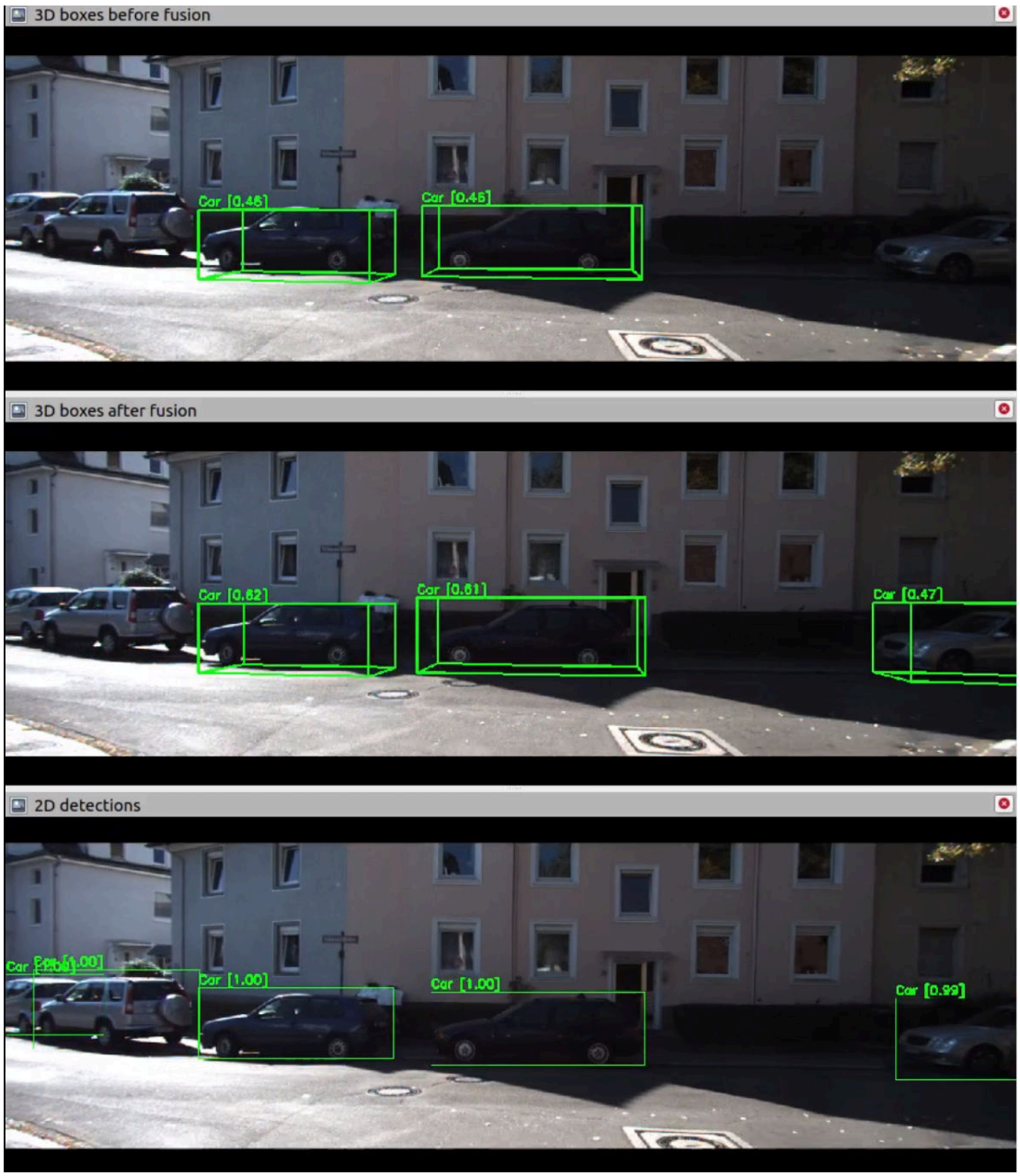


Figure A.6.: After fusion one car more on the right was detected and the confidence scores of the two cars in the middle were enhanced.
CenterPoint alone [0.48, 0.46] vs [0.62, 0.61] *CenterPoint + CLOCs*

Figure A.7.: A FP cyclist confused with the highway barrier was removed after fusion on the right side of the scene

Figure A.8.: A FP cyclist confused with a pole on the highway was removed after fusion on the right side of the scene

Figure A.9.: The white vehicle on the right was correctly detected as 'Van' by *CenterPoint*, but after fusion the TP was removed since a detection in 2D from *YOLOv4* was missing, so *CLOCs* considered the initial TP as a FP

Figure A.10.: The gray big vehicle on the right was well detected by both *CenterPoint* and *YOLOv4*, but there is a strong disagreement between the predicted labels that causes the 3D box 'Van' to be removed after fusion

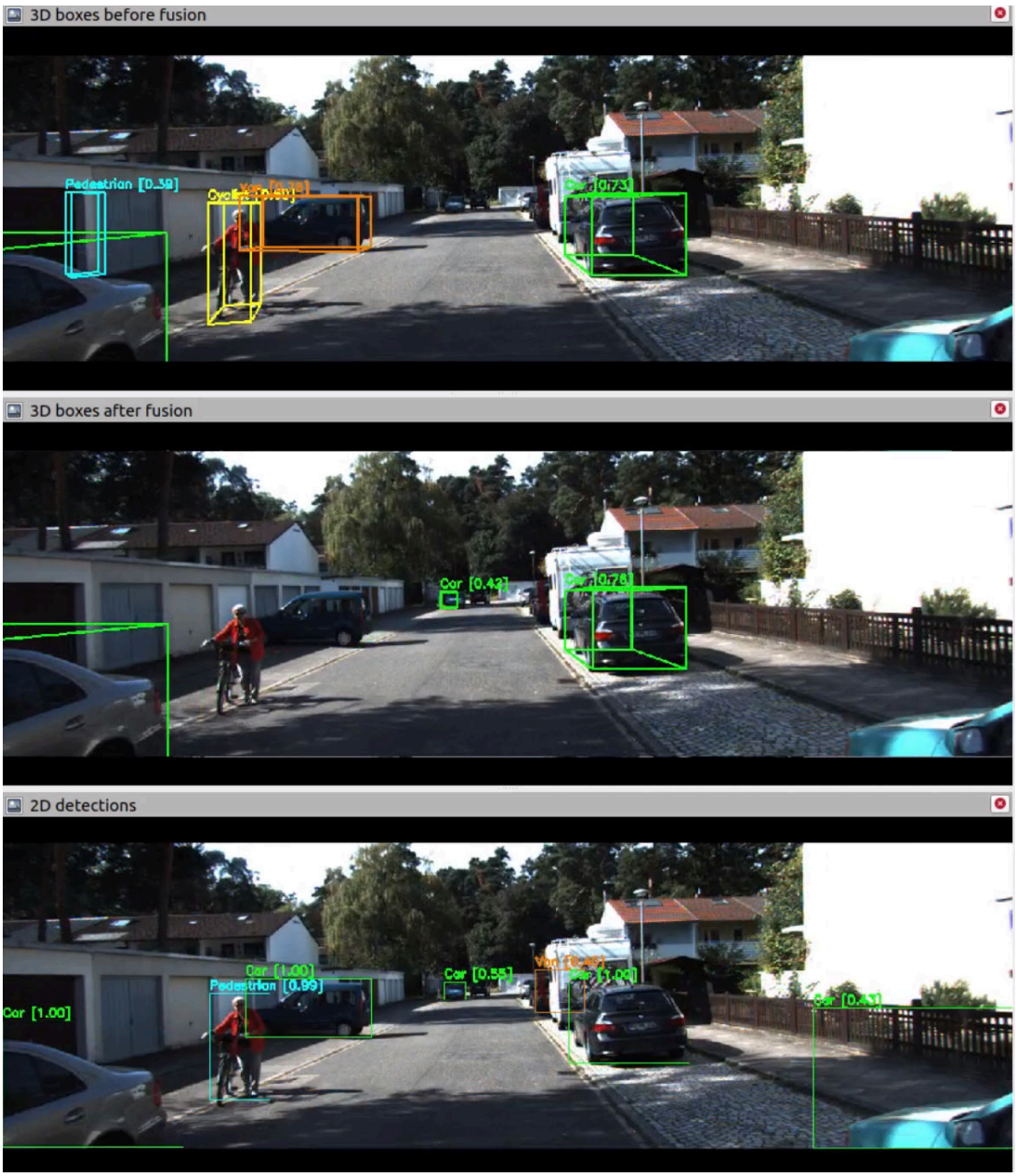


Figure A.11.: A FP pedestrian was deleted on the left side of the street after fusion, but also the TPs 'Cyclist' and 'Van' were suppressed because *YOLOv4* detected these with different labels 'Pedestrian' and 'Car' respectively. Also after fusion a car in the middle of the scene in the far range got detected in 3D thanks to a corresponding 2D 'Car' match from *YOLOv4*

Figure A.12.: A FP pedestrian confused with a tree trunk was removed and more cars on the right side of the street got detected after fusion, but also the TP van was deleted since *YOLOv4* predicted this van to be a car instead

Figure A.13.: This is almost the same scene as in the previous figure, but just a bit later in time so the objects appear closer. In this case the van that was missed in the previous figure after fusion, now it does get detected after fusion because *YOLOv4* agrees with *CenterPoint* that this vehicle is of the same class, then the labels 'Van' match both in 2D and 3D

Figure A.14.: Lots of FPs were removed after fusion, but also a TP pedestrian standing on left side next to the wall was suppressed because *YOLOv4* did not detect anything in that area

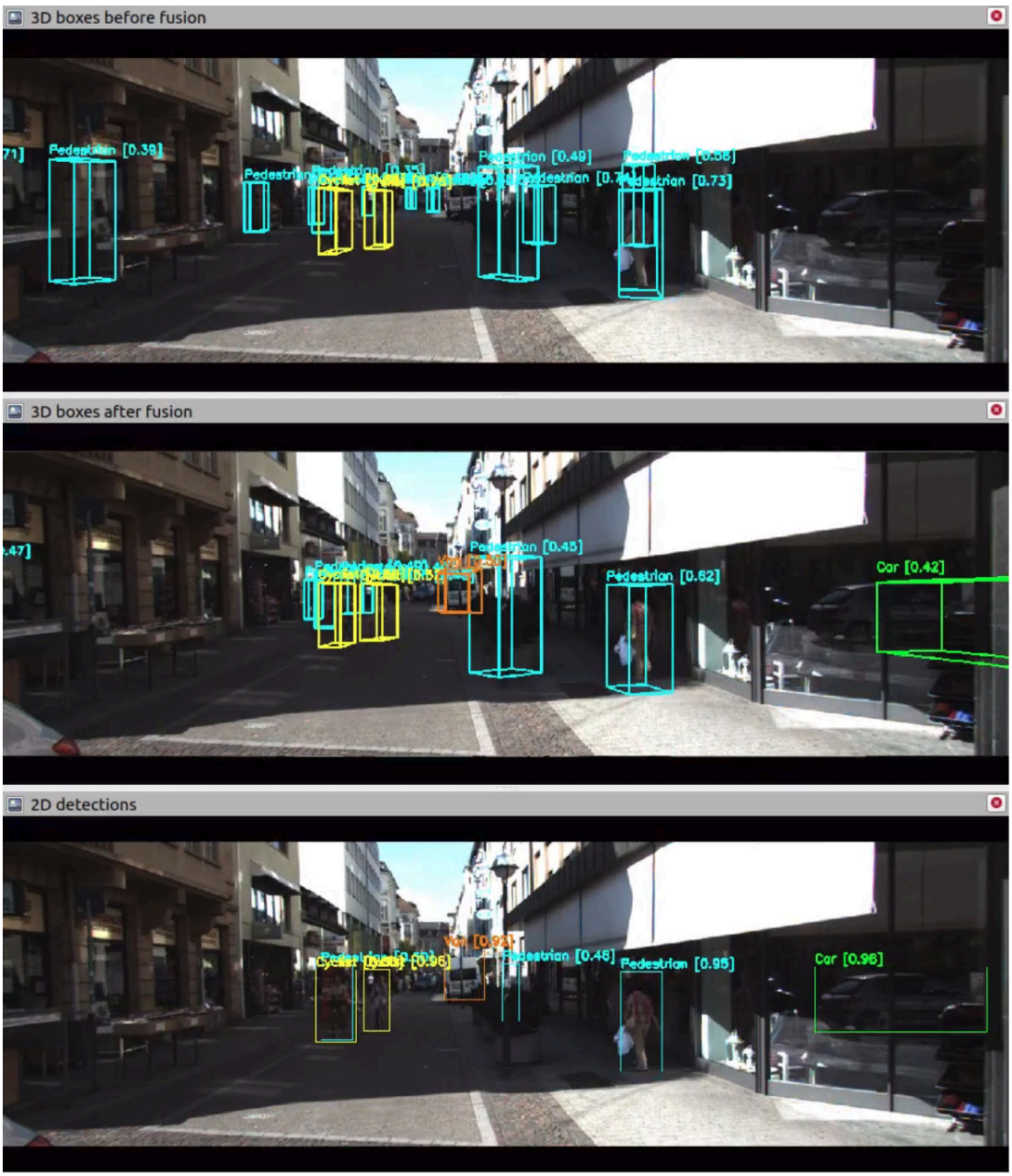


Figure A.15.: This a very peculiar edge-case where a car object (a tiny bit of its rear side is visible on the left bottom corner) is reflected on the shop window on the right of the scene, thus *YOLOv4* detects the reflected car image with high confidence , then after fusion *CLOCs* generated a ridiculous 3D car box FP inside the shop building. Look at the next figure that represents the same scene in the point cloud

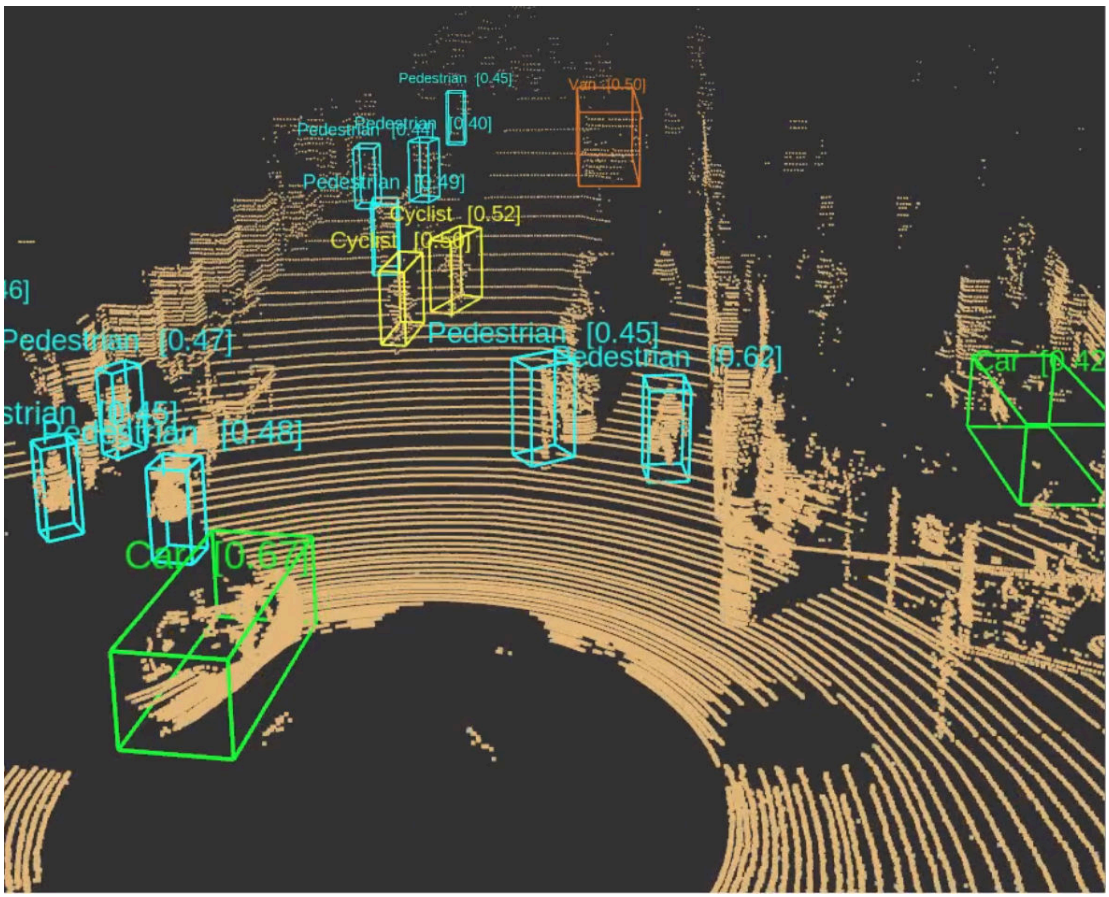


Figure A.16.: This is the same scene after fusion as in the previous figure in the point cloud domain. The FP predicted car on the right lies inside the walls of the shop building. The origin of this artifact is the reflected image on the shop window of the actual car passing by on the left. This fooled *CLOCs* into mistakenly predicting the FP car in an absurd location

Figure A.17.: This an example of a situation where the LiDAR-based detection is of advantage when bad light conditions are against the 2D detection. *YOLOv4* misses to identify the pedestrian in the shadow standing in the middle of the street, but *CenterPoint* both before and after fusion does detect this pedestrian very well

Bibliography

[1] A. Krämmer, C. Schöller, D. Gulati, V. Lakshminarasimhan, F. Kurz, D. Rosenbaum, C. Lenz, and A. Knoll. *Providentia - A Large-Scale Sensor System for the Assistance of Autonomous Vehicles and Its Evaluation*. 2020. arXiv: 1906.06789 [cs.RO].

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. "ROS: an open-source Robot Operating System." In: vol. 3. Jan. 2009.

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. *CARLA: An Open Urban Driving Simulator*. 2017. arXiv: 1711.03938 [cs.LG].

[4] A. Geiger, P. Lenz, and R. Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[5] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. *nuScenes: A multimodal dataset for autonomous driving*. 2020. arXiv: 1903.11027 [cs.LG].

[6] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov. *Large Scale Interactive Motion Forecasting for Autonomous Driving : The Waymo Open Motion Dataset*. 2021. arXiv: 2104.10133 [cs.CV].

[7] S. A. Bello, S. Yu, C. Wang, J. M. Adam, and J. Li. "Review: Deep Learning on 3D Point Clouds." In: *Remote Sensing* 12.11 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12111729.

[8] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: 1612.00593 [cs.CV].

[9] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. *Deep Learning for 3D Point Clouds: A Survey*. 2020. arXiv: 1912.12033 [cs.CV].

[10] S. Shi, L. Jiang, J. Deng, Z. Wang, C. Guo, J. Shi, X. Wang, and H. Li. *PV-RCNN++: Point-Voxel Feature Set Abstraction With Local Vector Representation for 3D Object Detection*. 2021. arXiv: 2102.00463 [cs.CV].

[11] Y. Zhou and O. Tuzel. *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. 2017. arXiv: 1711.06396 [cs.CV].

[12] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. *PointPillars: Fast Encoders for Object Detection from Point Clouds*. 2019. arXiv: 1812.05784 [cs.LG].

[13] Z. Yang, Y. Sun, S. Liu, and J. Jia. *3DSSD: Point-based 3D Single Stage Object Detector*. 2020. arXiv: 2002.10187 [cs.CV].

[14] S. Shi, X. Wang, and H. Li. *PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud*. 2019. arXiv: 1812.04244 [cs.CV].

[15] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington. *LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving*. 2019. arXiv: 1903.08701 [cs.CV].

[16] W. Shi, Ragunathan, and Rajkumar. *Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud*. 2020. arXiv: 2003.01251 [cs.CV].

[17] R. Nabati and H. Qi. "CenterFusion: Center-based Radar and Camera Fusion for 3D Object Detection." In: *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)* (Jan. 2021). DOI: 10.1109/wacv48630.2021.00157.

[18] B. Yang, R. Guo, M. Liang, S. Casas, and R. Urtasun. *RadarNet: Exploiting Radar for Robust Perception of Dynamic Objects*. 2020. arXiv: 2007.14366 [cs.CV].

[19] Y. Wang, Q. Mao, H. Zhu, Y. Zhang, J. Ji, and Y. Zhang. *Multi-Modal 3D Object Detection in Autonomous Driving: a Survey*. 2021. arXiv: 2106.12735 [cs.CV].

[20] W. Zhang, Z. Wang, and C. C. Loy. *Exploring Data Augmentation for Multi-Modality 3D Object Detection*. 2021. arXiv: 2012.12741 [cs.CV].

[21] S. Vora, A. H. Lang, B. Helou, and O. Beijbom. *PointPainting: Sequential Fusion for 3D Object Detection*. 2020. arXiv: 1911.10150 [cs.CV].

[22] T. Huang, Z. Liu, X. Chen, and X. Bai. *EPNet: Enhancing Point Features with Image Semantics for 3D Object Detection*. 2020. arXiv: 2007.08856 [cs.CV].

[23] J. H. Yoo, Y. Kim, J. Kim, and J. W. Choi. "3D-CVF: Generating Joint Camera and LiDAR Features Using Cross-view Spatial Feature Fusion for 3D Object Detection." In: *Lecture Notes in Computer Science* (2020), pp. 720–736. ISSN: 1611-3349. DOI: 10.1007/978-3-030-58583-9_43.

[24] Z. Zhang, M. Zhang, Z. Liang, X. Zhao, M. Yang, W. Tan, and S. Pu. *MAFF-Net: Filter False Positive for 3D Vehicle Detection with Multi-modal Adaptive Feature Fusion*. 2020. arXiv: 2009.10945 [cs.CV].

[25] S. Xu, D. Zhou, J. Fang, J. Yin, Z. Bin, and L. Zhang. *FusionPainting: Multimodal Fusion with Adaptive Attention for 3D Object Detection*. 2021. arXiv: 2106.12449 [cs.CV].

[26] Y. Yan, Y. Mao, and B. Li. "Second: Sparsely embedded convolutional detection." In: *Sensors* (2018).

[27] T. Yin, X. Zhou, and P. Krähenbühl. *Center-based 3D Object Detection and Tracking*. 2021. arXiv: 2006.11275 [cs.CV].

[28] X. Zhou, D. Wang, and P. Krähenbühl. "Objects as Points." In: *arXiv preprint arXiv:1904.07850*. 2019.

[29] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[30] J. Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016.

[31] M. Tan, R. Pang, and Q. V. Le. *EfficientDet: Scalable and Efficient Object Detection*. 2020. arXiv: 1911.09070 [cs.CV].

[32] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. *Detectron*. https://github.com/facebookresearch/detectron. 2018.

[33] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[34] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. *Scaled-YOLOv4: Scaling Cross Stage Partial Network*. 2021. arXiv: 2011.08036 [cs.CV].

[35] X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, and S. Wen. *PP-YOLO: An Effective and Efficient Implementation of Object Detector*. 2020. arXiv: 2007.12099 [cs.CV].

[36] X. Huang, X. Wang, W. Lv, X. Bai, X. Long, K. Deng, Q. Dang, S. Han, Q. Liu, X. Hu, D. Yu, Y. Ma, and O. Yoshie. *PP-YOLOv2: A Practical Object Detector*. 2021. arXiv: 2104.10419 [cs.CV].

[37] S. Pang, D. Morris, and H. Radha. *CLOCs: Camera-LiDAR Object Candidates Fusion for 3D Object Detection*. 2020. arXiv: 2009.00784 [cs.CV].

[38] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. "Vision meets Robotics: The KITTI Dataset." In: *International Journal of Robotics Research (IJRR)* (2013).

[39] W. Zimmer, A. Rangesh, and M. Trivedi. *3D BAT: A Semi-Automatic, Web-based 3D Annotation Toolbox for Full-Surround, Multi-Modal Data Streams*. 2019. arXiv: 1905.00525 [cs.CV].

[40] D. Tsai, S. Worrall, M. Shan, A. Lohr, and E. Nebot. *Optimising the selection of samples for robust lidar camera calibration*. 2021. arXiv: 2103.12287 [cs.CV].

[41] Y. Wang, X. Chen, Y. You, L. Erran, B. Hariharan, M. Campbell, K. Q. Weinberger, and W.-L. Chao. *Train in Germany, Test in The USA: Making 3D Object Detectors Generalize*. 2020. arXiv: 2005.08139 [cs.CV].

[42] C. Yuan, X. Liu, X. Hong, and F. Zhang. *Pixel-level Extrinsic Self Calibration of High Resolution LiDAR and Camera in Targetless Environments*. 2021. arXiv: 2103.01627 [cs.RO].

[43] X. Lv, B. Wang, D. Ye, and S. Wang. *LCCNet: LiDAR and Camera Self-Calibration using Cost Volume Network*. 2021. arXiv: 2012.13901 [cs.CV].

[44] *CenterPoint-PonintPillars Pytorch model convert to ONNX and TensorRT*. https://github.com/CarkusL/CenterPoint. Accessed: 2021-10-01.

[45] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár. *Designing Network Design Spaces*. 2020. arXiv: 2003.13678 [cs.CV].

[46] H. Sheng, S. Cai, Y. Liu, B. Deng, J. Huang, X.-S. Hua, and M.-J. Zhao. *Improving 3D Object Detection with Channel-wise Transformer*. 2021. arXiv: 2108.10723 [cs.CV].

[47] Z. Li, F. Wang, and N. Wang. *LiDAR R-CNN: An Efficient and Universal 3D Object Detector*. 2021. arXiv: 2103.15297 [cs.CV].

List of Figures

List of Tables