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

Leonardo Cabral Ferreira Cardoso
Instituto Superior Técnico, Lisboa
leonardocardoso@tecnico.ulisboa.pt

***Abstract*— Autonomous vehicles scan their environment with a range of sensors (*e.g.* camera, LiDAR) to take the safest action on the road. Therefore, locating and predicting the motion of other road agents has motivated plenty of research on the computer vision tasks of object detection and object tracking. This thesis takes an existing deep learning pipeline for 3D object detection [1] and modifies it to make location and tracking predictions from 3D semantic point clouds only. These point clouds are obtained in a pre-processing step which exploits the fusion of dense 2D semantic segmentation results with 3D point clouds that naturally offer depth information. The tracking method from [2] is embedded within the network to predict objects' displacements between two consecutive frames, thus, predicting their velocity. An extra input channel for a heatmap containing the objects' location in the previous frame was also tested. Several ablation studies were conducted to test the model's performance using different types of heatmaps, and not using heatmaps in any way. Results showed that a heatmap absent model yielded overall better results. Our deep learning approach allows end-to-end learning for detection and tracking, and runs 38% faster (18 FPS) than the baseline model (13 FPS).**

I. Introduction

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

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

The contributions of this thesis are three-fold:

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

II. Related Work

CenterTrack, in [2], is a point-based framework for joint object detection and tracking. The input consists of two consecutive frames as well as a heatmap of objects in the previous frame. The output consists of 2D bounding boxes and respective offsets between frames that allow trajectory inference. Despite presenting an end-to-end deep learning model, CenterTrack does not take advantage of the point cloud data format, it works exclusively on 2D images.

Simon et al., in [3], proposed a 3D object detection and tracking framework for point clouds. The framework, named Complexer-YOLO, takes LiDAR point clouds and RGB images as input. It projects 2D semantic segmentation results onto a voxelized point cloud, which is fed into the 3D object detection pipeline, Complex-YOLO [1]. Although object detection is achieved via a deep neural network, the tracking method is based on a classic method.

Our method is based on the Complexer-YOLO idea. However, we do not perform point cloud voxelization since it is computationally expensive. Instead, we project the point cloud to the ground plane and feed it to CNN. Moreover, we propose a deep learning network, based on a combination between Complex-YOLO and CenterTrack, to achieve end-to-end joint detection and tracking, based on 3D semantic point clouds, in a faster way.

III. Methods

In this section, the methods on which this thesis is based are presented. Therefore, a full understanding of these methods is essential. Based on Complexer-YOLO [3], our pipeline feeds a 3D semantic point cloud to the Complex-YOLO [1] model, which is a 3D object detector. Since Complex-YOLO expands on YOLOv2 [4], the latter is detailed first in Sec. III-A and the former is presented in Sec. III-C. No official code was released for Complex-YOLO. However, several implementations can be found online. Some are true to the original paper, thus, using YOLOv2 as the base for Complex-YOLO, while others, more recent versions, are based on YOLOv3 and YOLOv4, which are newer versions of the YOLO model. The chosen and later adapted source code[1] was an implementation of Complex-YOLO based on YOLOv4 [5] since it was the most recent version of YOLO at

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

the time. The third and fourth versions of YOLO are consecutive improvements over YOLOv2. However, most aspects of the object detector remain the same. Key differences from YOLOv2 to YOLOv3 and YOLOv4 are detailed in Sec. III-D and Sec. III-E, respectively.

A. YOLOv2

YOLOv2 [4] is a real-time 2D object detector whose input is a single RGB image, $I \in \mathbb{R}^{H \times W \times 3}$. The network architecture comprises convolutional and pooling operations only. The final layer is a 1×1 convolutional layer with N filters. In the middle of the feature extraction, the input image I is downsampled throughout the network by a factor of 32, producing an output feature map $\hat{F} \in \mathbb{R}^{\frac{H}{32} \times \frac{W}{32} \times N}$. This feature map depicts a grid where each cell is responsible for predicting $B = 5$ bounding boxes. For each bounding box, the network predicts its coordinates and dimensions, t_x, t_y, t_w, t_h, an objectness/confidence score t_o, and C class probabilities, one for each class of the dataset. Therefore, the number of channels in the output feature map is given by

$$N = B \times (5 + C). \tag{1}$$

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

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

$$t_o = Pr(\text{Object}) \times IoU^{\text{ground truth}}_{\text{prediction}}, \tag{2}$$

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

B. YOLOv2 Loss Function

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

$$\begin{aligned}
L_{\text{YOLOv2}} &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(O_i - \hat{O}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(O_i - \hat{O}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{Classes}} p_{ij}(c) \log \left(\hat{p}_{ij}(c) \right)
\end{aligned} \tag{3}$$

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

In the YOLOv2 paradigm, the majority of predicted boxes are not responsible for detecting objects. This sets their objectness scores (confidence scores) to zero, thus, overwhelming the gradient from cells that contain objects. This leads to model instability, making the training diverge early on. To tackle this issue, a smaller weight is attributed to the confidence loss of non-responsible boxes, by multiplying its term with $\lambda_{\text{noobj}} = 0.5$. Similarly, the localization loss is multiplied by $\lambda_{\text{coord}} = 5$ to assign more weight to the localization error in the backward propagation of errors to assure convergence in the early stages of training. At test time, the C conditional class probabilities are multiplied by the box's objectness score (2):

$$\begin{aligned}
P(\text{Class}_c|\text{Object}) \times P(\text{Object}) \times IoU^{\text{ground truth}}_{\text{prediction}} = \\
P(Class_c) \times IoU^{\text{ground truth}}_{\text{prediction}}, c \in C.
\end{aligned} \tag{4}$$

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

C. Complex-YOLO

Complex-YOLO [1] is a real-time 3D object detector whose input is LiDAR-based only. It expands on YOLOv2, by predicting the orientation of the regressed bounding boxes. The input point cloud is pre-processed into a BEV RGB image where each channel encodes handcrafted features. These channels encode the points' height, intensity, and density, respectively. The resulting BEV RGB map is fed into the Complex-YOLO network whose novel E-RPN (Euler Region Proposal Network) is responsible for estimating the heading of the object by adding a real and an imaginary fraction, t_{re} and t_{im}, to the regression network. With the oriented bounding boxes obtained on the BEV perspective, a pre-defined height for each class is attributed to the objects to output 3D bounding boxes.

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

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

$$b_\phi = arg(|z|e^{ib_\phi}) = \text{atan2}(t_{im}, t_{re}) \tag{5}$$

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

$$N = B \times (7 + C). \tag{6}$$

Complex-YOLO loss function extends YOLOv2's loss function (3) by adding an Euler regression part:

$$L_{\text{Complex-YOLO}} = L_{\text{YOLOv2}} + L_{\text{Euler}}. \tag{7}$$

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

$$\begin{aligned} L_{\text{Euler}} &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left| \left(e^{ib_\phi} - e^{i\hat{b}_\phi} \right)^2 \right| \\ &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(t_{im} - \hat{t}_{im} \right)^2 + \left(t_{re} - \hat{t}_{re} \right)^2 \right] \end{aligned} \tag{8}$$

Similarly to (3), the Euler loss term is also scaled by the factor λ_{coord} to guarantee convergence in early training stages. Moreover, the prediction of t_{im} and t_{re} is only penalized for responsible bounding boxes, i.e., $\mathbb{1}_{ij}^{\text{obj}}$ equals to one only if the j^{th} bounding box predictor in cell i has the highest IoU between itself and the ground truth box.

D. YOLOv3

YOLOv3 [6] is the follow-up work of YOLOv2. Similarly, it takes as input a single RGB image, $I \in \mathbb{R}^{H \times W \times 3}$, and performs bounding box regression in the same way from an output feature map. However, YOLOv3 innovates by making predictions at three different scales, i.e., an output feature map, $\hat{F} \in \mathbb{R}^{\frac{H}{R} \times \frac{W}{R} \times N}$, is obtained for each scale. R denotes the downscaling factor of the feature maps and N denotes the number of channels, which, similarly to YOLOv2, is given by (1) with $B = 3$, i.e., it predicts $B = 3$ bounding boxes per grid cell of the output feature map.

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

As stated in Sec. III-B, YOLOv2 computes its loss function in (3) concerning the output feature map. The same idea is applied in YOLOv3, thus, the following loss function is

computed for each scale of the output feature maps:

$$
\begin{aligned}
L_{\text{YOLOv3\_Scale\_s}} &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \text{BCE} \left(\hat{O}_i, O_i \right) \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \text{BCE} \left(\hat{O}_i, O_i \right) \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{Classes}} \text{BCE} \left(\hat{p}_{ij}(c), p_{ij}(c) \right), \qquad (9)
\end{aligned}
$$

with

$$
\text{BCE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}). \qquad (10)
$$

It is identical to (3), however, the objectness (confidence) and classification terms (the last three terms) are now optimized by the Binary Cross-Entropy (BCE) loss. YOLOv3 aims to optimize the sum of the three loss functions corresponding to each output scale. Consequently, its loss function is given by

$$
L_{\text{YOLOv3}} = \sum_{s=1}^{3} L_{\text{YOLOv3\_Scale\_s}}. \qquad (11)
$$

E. YOLOv4

YOLOv4 [5] has an identical pipeline to its predecessor. However, it presented some modifications to increase the model performance. These innovations are two-fold: (i) changes that only increase the training cost and (ii) changes that slightly increase the inference cost but result in a significant accuracy improvement.

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

YOLOv4 significantly altered YOLOv3's feature extractor. These modifications significantly improve the accuracy of the model, with a slight increase in the inference cost. The feature extractor is redesigned to accommodate Cross-stage Partial (CSP) connections, a Spatial Pyramid Pooling (SPP) module, and a Path Aggregation Network (PANet). In short, these methods are used to (i) reduce the vanishing gradient problem (i.e., the difficulty of backpropagating loss signals in very deep networks), (ii) reinforce feature propagation, (iii) promote features reuse, and (iv) reduce the number of network parameters.

Despite the significant changes promoted to the architecture, YOLOv4 still predicts bounding boxes at three different scales. YOLOv4 changed from SSE to MSE (Mean Square Error) for the localization loss terms of each scale (the first two terms), and it is now given by:

$$
\begin{aligned}
L_{\text{YOLOv4\_Scale\_s}} &= \lambda_{\text{coord}} \frac{1}{N} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \frac{1}{N} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \text{BCE} \left(\hat{O}_i, O_i \right) \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \text{BCE} \left(\hat{O}_i, O_i \right) \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{Classes}} \text{BCE} \left(\hat{p}_{ij}(c), p_{ij}(c) \right), \qquad (12)
\end{aligned}
$$

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

$$
L_{\text{YOLOv4}} = \sum_{s=1}^{3} L_{\text{YOLOv4\_Scale\_s}}. \qquad (13)
$$

IV. Proposed Pipeline

Our pipeline performs semantic segmentation on a 2D image and fuses this information with a 3D point cloud to obtain a 3D semantic point cloud. Then, it performs 3D object detection and tracking on this rich environment representation that offers both semantic and depth information. In this chapter, the developed pipeline is described, and the choices behind the design are justified. The dataset of choice, the Semantic Segmentation module, the 3D Object Detection, and the 3D Object Detection and Tracking module are described in detail in Sec. IV-A, Sec. IV-B, Sec. IV-C, and Sec. IV-D, respectively.

A. KITTI dataset

The KITTI dataset [7] was chosen due to its popularity and widespread use among research methods that apply deep learning techniques focused on autonomous driving scenarios. Two branches of the dataset were used, the 3D Object Detection and the 3D Multi-Object Tracking datasets. The former is a set of unordered frames and was used to train and evaluate the object detector. The latter is a set of sequences of frames and was used to train and evaluate the proposed joint object detector and tracker.

The 3D Object Detection and 3D Multi-Object Tracking datasets consist of frames from the KITTI vehicle's environment. Each frame corresponds to a full spin of the LiDAR scanner, with a sampling frame rate equal to 10 Hz. KITTI's ego-vehicle is equipped with two grayscale cameras (indexed 0 and 1), two color cameras (indexed 2 and 3), a LiDAR sensor, and an IMU/GPS module. The monocular RGB images used are captured by Camera 2. Every time the Velodyne laser scanner (LiDAR) rotates to the vehicle's forward position, the cameras are triggered to capture one image, guaranteeing the synchronization between the camera and LiDAR sensors.

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

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

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

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

$$\mathbf{y} \sim \mathbf{P}_{\text{rect}}^{(i)} \mathbf{R}_{\text{rect}}^{(0)} \mathbf{T}_{\text{cam}}^{\text{velo}} \mathbf{x} \tag{14}$$

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

$$\mathbf{P}_{\text{rect}}^{(i)} = \begin{bmatrix} f_u^{(i)} & 0 & c_u^{(i)} & -f_u^{(i)} b_x^{(i)} \\ 0 & f_v^{(i)} & c_v^{(i)} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{15}$$

This matrix describes the mapping of 3D points in the world, through a pinhole camera model [8], to 2D points in the i^{th} camera image plane.

$\mathbf{R}_{\text{rect}}^{(0)} \in \mathbb{R}^{3\times3}$ is the rectifying rotation matrix of the reference camera, i.e., camera number 0. Rectification consists of applying a rotation to make images of distinct cameras lie on the same plane [9]. Finally, $\mathbf{T}_{\text{velo}}^{\text{cam}} \in \mathbb{R}^{4\times4}$ denotes the 3D rigid-body transformation that takes points from the Velodyne coordinate system to the camera coordinate system. This transformation matrix is given by

$$\mathbf{T}_{\text{velo}}^{\text{cam}} = \begin{bmatrix} \mathbf{R}_{\text{velo}}^{\text{cam}} & \mathbf{t}_{\text{velo}}^{\text{cam}} \\ 0 & 1 \end{bmatrix} \tag{16}$$

where $\mathbf{R}_{\text{velo}}^{\text{cam}} \in \mathbb{R}^{3\times3}$ denotes the rotation matrix from the Velodyne coordinate system to the camera coordinate system and $\mathbf{t}_{\text{velo}}^{\text{cam}} \in \mathbb{R}^{1\times3}$ denotes the translation vector from the Velodyne coordinate system to the camera coordinate system.

B. Semantic Segmentation Module

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

Following the SOTA methods for semantic segmentation [10]–[16], which are all based on deep learning techniques, using a deep learning network to perform semantic segmentation was the obvious choice. Since the semantic segmentation module will be fed with images from the KITTI domain, the semantic segmentation network should be trained on RGB images from the Semantic Segmentation branch of the KITTI dataset. However, the KITTI Semantic Segmentation dataset is quite limited, as it only provides 200 labeled RGB images. Training a deep learning network on such a low number of samples would lead to poor results in the task at hand. A solution to this problem is presented by the MSeg dataset [17]. The authors introduce a composite dataset that combines several existing semantic segmentation datasets, including the KITTI dataset, into a unified domain. Their composite dataset enables training a single semantic segmentation network, which is effective even when tested on individual datasets. Besides solving the scarce dataset problem, the authors in [17] also offer a pre-trained model[2], similarly named MSeg, ready to use with the KITTI dataset. Performing semantic segmentation on the RGB images, captured by the ego-vehicle's Camera 2, implies feeding the images to the pre-trained MSeg model. The network outputs pixel-wise class labels, predicted from 193 different class labels.

With the semantic segmentation results obtained for the RGB image, the final step of this module consists of mapping 3D points in the world to pixels in the image plane, in order to obtain a 3D semantic point cloud. Initially, to simplify computation, every point in the point cloud is converted to homogeneous coordinates, i.e., a point $\mathbf{x} = [x, y, z]^T$ now has the coordinates $\mathbf{x} = [x, y, z, 1]^T$. Next, (14) is applied to project the whole point cloud onto the 2nd image plane (RGB is captured by camera number 2). Then, the whole point cloud is normalized by the z-coordinate to convert to pixel coordinates. Given the whole point cloud in pixel coordinates of the 2nd image plane, the points that live outside the camera FoV are removed. Finally, having every 3D point inside the camera FoV associated with a pixel from camera 2, the correspondent pixel-wise class label is copied onto the respective 3D point.

C. 3D Object Detection Module

The 3D object detection module is based on the Complex-YOLO pipeline [1]. The selected Complex-YOLO pipeline consists of an implementation based on YOLOv4. Henceforth, this implementation will be called Complex-YOLOv4.

The Complex-YOLOv4 pipeline was altered to accommodate the semantic information obtained in the Semantic Segmentation module (Sec. IV-B). The main difference to the original pipeline, to achieve 3D object detection only, lies in the network input. Further changes to embed the tracking task to this pipeline are later explained in Sec. IV-D. For now, only the changes made to achieve object detection on semantic point clouds are described. Originally, the network takes as input a BEV RGB map where each channel consists

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

of handcrafted features, such as the points' height, intensity, and density, respectively. Now, the Complex-YOLOv4 network takes as input a BEV RGB map where the RGB channels encode the points' semantic class labels.

Another change promoted to the Complex-YOLOv4 pipeline is related to its network architecture, in order to predict and detect an extra class, the Truck class. The KITTI dataset labels seven different classes of objects (see Table I). The Complex-YOLOv4 implementation follows the original paper and predicts only three classes: Car, Pedestrian, and Cyclist. It does so by grouping Car and Van into the same Car class, Pedestrian and Person Sitting into the unique Pedestrian class, and Cyclist needs no further grouping. Therefore, the classes Truck and Tram were ignored by the model. Since Trams are rarer than Trucks on the dataset (see Table I), only objects whose class was Truck were made to be predicted/detected by the network. The change consists of incrementing the number of classes, denoted by C, in (6).

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

In Velodyne coordinates, the BEV perspective of the semantic point cloud is obtained by discarding the z-coordinate and mapping the xy plane (ground plane) onto an image plane. The labelled points whose coordinates fall into the square delimited by $x \in [0, 50]m$ and $y \in [-25, 25]m$ are mapped onto a square image with 608×608 resolution, thus preserving the aspect ratio. The mapping is simply done by converting the points' coordinates in meters to pixel coordinates, where each pixel corresponds to approximately $0.08m$. Finally, the pixels are assigned a color according to the respective point's class label previously obtained in the Semantic Segmentation module. The class-color pair association is given by a dictionary that reduces the 193 different classes output by MSeg to nine classes and consequently nine different colors. Bright and contrasting colors are attributed to the four classes to be detected (Car, Pedestrian, Cyclist, Truck), whereas dim and similar colors are attributed to non-detectable classes.

D. 3D Object Detection and Tracking Module

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

Based on CenterTrack [2], the goal is to predict object-wise bounding boxes, their offsets relative to the previous frame, and a heatmap depicting the objects' locations in the current frame. The input consists of (i) the frame whose objects will be detected, (ii) the previous frame, and (iii) the heatmap with the objects' location on the previous frame. In our case, the current and previous frames consist of Semantic BEV RGB maps. At test time, it comprises a feedback loop that feeds the pipeline on instant *t* with the heatmap predicted on the previous instant, *t-1*. During training, the input heatmap consisted of the ground truth. Consequently, a heatmap dataset was produced to train the network.

The goal of the heatmaps is to feed the network the location in the previous frame of the objects to be detected in the current frame. A dataset of ground truth heatmaps was generated so the network could compare its output with the ground truth, therefore, learning how to predict heatmaps. Ground truth heatmaps are also used during training as the input heatmap from the previous frame. In other words, the feedback loop does not exist during training. Thus, it is performed in an offline manner.

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

Two different heatmap styles were generated and tested. Initially, centroids were depicted as ellipses to better resemble the rectangular bounding boxes. Results showed that this idea didn't work so well, as detailed in Sec. V-B. Consequently, the second type of heatmaps was tested where the centroids depict circular shapes adaptive to the object's dimensions, similarly to CenterTrack's method.

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

$$g(x, y) = A \cdot \exp\left(-\left(a(x - x_0)^2 + 2b(x - x_0)(y - y_0) + c(y - y_0)^2\right)\right) \tag{17}$$

where,

$$\begin{aligned} a &= \frac{\cos^2\theta}{2\sigma_X^2} + \frac{\sin^2\theta}{2\sigma_Y^2} \\ b &= -\frac{\sin 2\theta}{4\sigma_X^2} + \frac{\sin 2\theta}{4\sigma_Y^2} \\ c &= \frac{\sin^2\theta}{2\sigma_X^2} + \frac{\cos^2\theta}{2\sigma_X^2}. \end{aligned} \tag{18}$$

In (17), the pair (x_0, y_0) denotes the mean value of the normal distribution and it equals to the coordinates of the splattered object's centroid. The amplitude value $A = 1$ in order to get normalized values, i.e., within the range $[0, 1]$. In (18), the standard deviation parameter is what gives the

centroids' gaussian blobs different shapes. For the elliptical centroids, $\sigma_X = w/2$ and $\sigma_Y = l/2$, where w and l depict the object's width and length, respectively. For circular centroids, $\sigma_X = \sigma_Y$ and their value is a function of the object size [18]. Finally, the angle $\theta = r_y$, where r_y depicts the object rotation over the y axis in camera coordinates.

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

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

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

$$L = L_{\text{YOLOv4\_Scale\_3}} + L_{\text{Offsets}} + L_{\text{Heatmap}} \tag{19}$$

where $L_{\text{YOLOv4\_Scale\_3}}$ is given by (12),

$$L_{\text{Offsets}} = \frac{1}{N} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(dx_i - \hat{dx}_i)^2 + (dy_i - \hat{dy}_i)^2 \right] \tag{20}$$

is similarly computed to the localization loss in (12). L_{Offsets} consists of the MSE loss for every responsible bounding box (as detailed in Sec. III-B), and

$$L_{\text{Heatmap}} = -\lambda_{hm} \frac{1}{M} \cdot \sum_{c=1}^{C} \sum_{p=0}^{P^2} \begin{cases} (1 - \hat{H}_{xyc})^{\alpha} \log(\hat{H}_{xyc}) & \text{if } H_{xyc} = 1 \\ (1 - H_{xyc})^{\beta} (\hat{H}_{xyc})^{\alpha} \log(1 - \hat{H}_{xyc}) & \text{otherwise} \end{cases} \tag{21}$$

depicts the loss function for the predicted heatmap. It consists of an adapted focal loss function [18], with parameters $\alpha = 2$ and $\beta = 4$. It iterates over each cell of every channel in the predicted heatmap, with $C = 4$ classes and $P = 152$ pixels/cells. M denotes the number of objects in the current frame, and $\lambda_{hm} = 0.5$ is responsible for decreasing the weight of the heatmap loss value. The minus sign, in the beginning, is to invert the logarithmic growth. A high penalty is given when a ground truth peak ($H = 1$) is wrongly predicted and a low ground truth value is predicted as a peak or a neighboring point to the peak. However, any predicted values near the peaks ($0.7 < \hat{H} < 0.9$) are barely and equally penalized for either a right or wrong prediction.

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

$$\vec{v}_{\text{object\_ego}} = \begin{bmatrix} \frac{dx}{T} \\ \frac{dy}{T} \end{bmatrix}. \tag{22}$$

Applying the relative velocity equation

$$\vec{v}_{\text{object\_ego}} = \vec{v}_{\text{object}} - \vec{v}_{\text{ego}}, \tag{23}$$

and using the ego-vehicle velocity (given by the IMU)

$$\vec{v}_{\text{ego}} = \begin{bmatrix} v_{\text{ego\_right}} \\ v_{\text{ego\_forward}} \end{bmatrix}, \tag{24}$$

we can compute the velocity of an object

$$\vec{v}_{\text{object}} = \vec{v}_{\text{object\_ego}} + \vec{v}_{\text{ego}} = \begin{bmatrix} \frac{dx}{T} \\ \frac{dy}{T} \end{bmatrix} + \begin{bmatrix} v_{\text{ego\_right}} \\ v_{\text{ego\_forward}} \end{bmatrix}. \tag{25}$$

As detailed in Sec. IV-C, a pixel in the BEV RGB map corresponds to $0.08m$. Given this ratio, the predicted velocities of the detected objects can be drawn onto the BEV RGB map.

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

V. Experimental Results

In this section, the detection results obtained for the 3D Object Detection module are presented. Then, the results obtained for the 3D Object Detection and Tracking module are detailed along with empirical results that justify design choices and an ablation study for the final model architecture. Lastly, a comparison between our model and the baseline model, Complexer-YOLO, is performed.

All models were trained for 300 epochs on an NVIDIA GeForce GTX 1070, with a batch size equal to 2, using the Adam optimizer. A cosine learning rate decay is applied and its initial value is 0.001. Other hyperparameters include the weights associated to the loss functions (12) and (21).

A. 3D Object Detection

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

Initially, the 3D object detection Complex-YOLOv4 was only altered to accept Semantic BEV RGB maps instead of the handcrafted features encoded in the RGB map of stock Complex-YOLOv4. Then, we added the Truck class as an

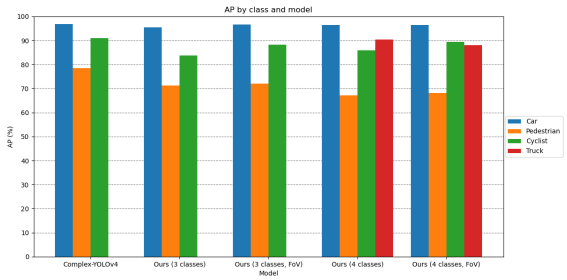

Fig. 1: AP per class per 3D Object Detection Model.

extra class to be detected. Finally, the FoV filtering was applied to filter out ground truth objects whose locations were outside the camera FoV. If the objects live outside the FoV they do not have semantic features, therefore, do not appear in the BEV RGB map. Before the FoV filtering, the network was penalized for not predicting objects not visible in the BEV RGB map, i.e., feature-less objects. This change was promoted to both the 3-class and 4-class models, and in both cases, it increased the mAP.

Average Precision (AP) values for each class and model are depicted in Fig. 1. It is visible that the overall better-detected class is the Car class, for any of the tested object detection modules. This occurs due to the imbalanced class distribution (see Table I) present in the 3D Object Detection of the KITTI dataset. The Car class (Car and Van together) make up for over 80% of the available ground truth objects, while Pedestrians, Cyclists, and Trucks comprise 11%, 4%, and 2% of the labeled objects, respectively. Eventhough there are more Pedestrian than Cyclist and Truck instances combined in the dataset, the tested 3D object detection models yielded better results for Cyclist and Truck detection than for Pedestrian detection. This happens because the size of the objects also plays a role in the model's performance at detecting them. In the point clouds, the Pedestrian object instances comprise the least amount of points per instance.

Although the stock Complex-YOLOv4 model achieves the highest mAP (88.77%) out of all the tested models, unlike Complex-YOLOv4, our method does not use handcrafted features to build the BEV RGB maps. Instead, it exploits semantic information to obtain the BEV RGB maps and achieves an on-par mAP result (85.52%).

B. 3D Object Detection and Tracking

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

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

C. Heatmap Loss Function and Tuning

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

To try to fix this issue, a higher weight value was attributed to the heatmap loss by multiplying its value by a factor λ_{hm}. Several values were tested, and through empirical observation, it was noted that changing this value would result in good early convergence for the heatmap loss curve. However, it only delayed the divergent course for the heatmap MSE loss curve. Identical results were obtained by changing the loss function to the L1 loss. Finally, the adapted focal loss proposed in [18] was implemented, tested, and resulted in convergence for the heatmap loss. Contrary to the MSE loss, which penalized errors equally, the focal loss penalized predictions more heavily when the predicted and ground truth values were far apart, as stated in Sec. IV-D. Although converging, the loss values for the heatmap loss were still too high compared to the other losses. Ideally, the loss curves should converge at about the same rate. Therefore, the factor λ_{hm} was set to 0.5. Thus, decreasing the weight given to the heatmap loss function.

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

D. Heatmap Upsampling

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

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

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

the training stage is crucial because the adapted focal loss applied to the heatmap loss needs to have these peaks where the target heatmap is 1, to correctly compute the loss and, consequently, correctly predict the heatmaps.

The output feature map of scale 3 has its spatial dimensions upsampled and the number of channels downsampled to reach the desired output heatmap shape of $4 \times 608 \times 608$. Applying the coarser nearest-neighbor interpolation to the output feature maps would render the loss of features, whereas the application of the finer bilinear interpolation to the output feature maps leads to maintaining the important features needed to mimic the gaussian blobs of the target heatmaps. Through empirical observation, bilinear interpolation leads to better heatmap prediction.

E. Train-Test Split

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

The train-test split was performed following the rule of thumb that 80% of the samples should be used for training and the remaining 20% for testing. This results in 17 sequences used for training and 4 sequences used for testing purposes. Besides the 80/20 split, each of the four sequences selected for the testing stage belongs to the four environment categories stated above. The resulting class distribution for the test split is detailed in Table II.

F. Ablation Study

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

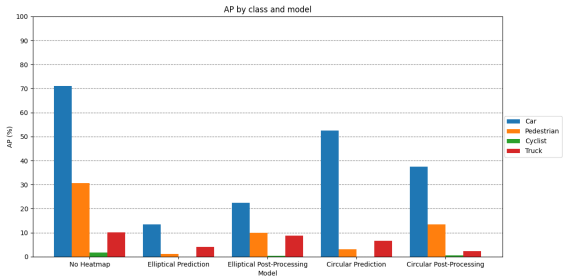

Fig. 2: AP per class per 3D Object Detection and Tracking Model.

The next logical step was taken, a dataset of circular style heatmaps was generated. The circular centroids within these heatmaps have a reduced standard deviation value which decreases the spread of the gaussian blobs. The network was trained using this heatmap style. Qualitative results showed that (i) the predicted heatmaps, which go through the feedback loop, were quite similar to the ground truth (target) heatmaps and, consequently, (ii) more objects are detected, using this type of heatmaps. Another experiment was conducted, in which the ground truth circular heatmap was fed to the network.

A final heatmap experiment was conducted in which the input heatmap at instant t is obtained via a post-processing of the heatmap predicted at $t-1$. Instead of directly injecting the predicted heatmap in the following instant, it injected a new heatmap generated from the predicted centroids in the heatmap and bounding box dimensions. It increased the mAP value for the elliptical style heatmap models, since it feeds the network a true elliptical style heatmap. However, many centroids are still missing in the predicted heatmap, thus, they are not present on the post-processing heatmap, consequently, degrading the object detection performance of the model. In the case of the post-processed heatmap for the circular heatmap model, the mAP yielded a similarly poor result to the direct feeding of heatmaps without post-processing.

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

Average Precision (AP) values for each model and class are depicted in Fig. 2. The domination of the Car class in the dataset makes every model perform significantly better for this class. The poor detection of the other classes is explained by the class distribution as well (see Table I). This lowers the mAP value significantly since it averages the AP value across all classes.

Feeding the heatmap models with the ground truth heatmap leads to a significant improvement of the recall value, i.e., more and more occluded objects are being detected correctly. This demonstrates that the features from the heatmap influence the detection and offset prediction. Moreover, the elliptical model fares better than the circular

model when both take the ground truth heatmap as input ($mAP = 67.60\%$ vs. $mAP = 46.07\%$). This happens because the elliptical model comprises heatmaps whose centroids mimic the orientation and shape of the bounding boxes, thus, making the bounding box regression task easier.

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

G. Complexer-YOLO vs. Ours

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

Complexer-YOLO runs the real-time ENet for semantic segmentation at 90 FPS. We applied the MSeg model since it was the one readily available to deploy. Complexer-YOLO performs object detection (predicting on voxelized input) at 15 FPS and runs the tracking method at 100 FPS. Consequently, their bottleneck relies on the detection task, which added to the tracking task amounts to 13 FPS. Our method runs the proposed joint detection and tracking network at 18 FPS. Therefore, our method improves the bottleneck computational time of the Complexer-YOLO pipeline, while performing detection and tracking with an end-to-end trainable deep network.

VI. Conclusion

This work exploits the fusion of dense 2D semantic information with 3D geometric data to achieve robust object detection in the three-dimensional space. The tracking paradigm from CenterTrack was embedded within the Complex-YOLO architecture to achieve a joint 3D object detection and tracking pipeline. Results demonstrated that input heatmaps, containing objects' locations in the previous frame, help the detection task because they explicitly offer a feature map that helps to detect more occluded objects. However, the achieved heatmap prediction presented poor results, which degraded the model's performance. Therefore, the heatmap absent model yielded overall better results.

Comparing with the baseline method of Complexer-YOLO, our method improved their bottleneck computational time from 13 FPS to 18 FPS, by performing joint object detection and tracking in an end-to-end fashion using a deep neural network.

Future work can be done towards experimenting with other heatmap loss functions to improve the heatmap prediction and using data augmentation techniques to counter the bad class distribution in the dataset, thus, improving the model generalization.

References

[1] M. Simon, S. Milzy, K. Amendey, and H.-M. Gross, "Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds," in *Proceedings of the European Conf. Computer Vision (ECCV) Workshops*, 2018. 1, 3, 5

[2] X. Zhou, V. Koltun, and P. Krähenbühl, "Tracking objects as points," in *European Conf. Computer Vision (ECCV)*, 2020, pp. 474–490. 1, 6

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

[4] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7263–7271. 1, 2

[5] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020. 1, 4

[6] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018. 3

[7] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3354–3361. 4

[8] P. Sturm, *Pinhole Camera Model*. Springer US, 2014, pp. 610–613. 5

[9] O. Faugeras, *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993. 5

[10] F. J. Lawin, M. Danelljan, P. Tosteberg, G. Bhat, F. S. Khan, and M. Felsberg, "Deep projective 3d semantic segmentation," in *International Conference on Computer Analysis of Images and Patterns*. Springer, 2017, pp. 95–107. 5

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440. 5

[12] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou, "Tangent convolutions for dense prediction in 3d," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 3887–3896. 5

[13] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, July 2017. 5

[14] F. Engelmann, T. Kontogianni, J. Schult, and B. Leibe, "Know what your neighbors do: 3d semantic segmentation of point clouds," in *European Conf. Computer Vision (ECCV)*, 2018, pp. 0–0. 5

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

[16] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4558–4567. 5

[17] J. Lambert, Z. Liu, O. Sener, J. Hays, and V. Koltun, "MSeg: A composite dataset for multi-domain semantic segmentation," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020. 5

[18] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *European Conf. Computer Vision (ECCV)*, 2018, pp. 734–750. 7, 8