Project4

Mapping and perception for an autonomous robot/ 0510-7591

By: Tomer Shimshi

Tel Aviv University

August 2023

Abstract:

In this project we dive deep into the world of 3D object detection and Multi Object Tracking.

In Part A we explain the mechanism of the feature encoder of the Point pillars model, and analyzed 3 detection results of 3 different scenarios.

In part B we analyze 3 models for 3D detection based on point cloud and RGB image, namely the Frustum Convnet, Frustum PointNet, Frustum-PointPillars. We briefly explain the architecture of each model, we mention 2 advantages of the Frustum-PointPillars architecture compare to PointPillar approach. And give further explanation of the mechanasimim used to distinguish between forground and background clutter.

In part C We first explore the working mechanism of 4 known trackers by their academic articles and describe the benefit of each article compared to the one that came before it. After that we run the BoT-SORT racker on the five given subsets from MOT17 and attached the results to this report, on top of that we gave a high-level explanation of the results we got. In the finale part of the report, we analyzed more deeply each of the resulting videos and gave example of good and bad tracking from the outputted videos themsels.

Contents

Abstract:....2

Solutions: 4

Part A Detection analysis based on PointPillars (PP) results on KITTI dataset. 4

1. 4

2. 4

Example1:....4

Example 2....7

Example 3....9

Part B....11

1. Describe shortly each of the Multi-stage methods based on the paper: Frustum Convnet,Frustum PointNet and Frustum-PointPillars architecture.....11

2. Mention at least 2 advantages of Frustum-PointPillars architecture compare to PointPillar approuch.....21

3. 21
A.What is the mechanism described in the paper to isoltaed foreground and background clutter?....21
B. In any case foreground and background clutter are not well Isolated?....21
Part C: Multi Object Tracking....21
1....21
A: Describe the main mechanism of the SORT algorithm. 21
B. Describe the main benefit of the DeepSort algorithm to the SORT process. Elaborate on RE-ID in your explanation. 22
C. Describe at least two good additions to ByteTrack for DeepSort.....22
D. Describe at least three good additions to BoT-SORT compared to ByteTrack. 23
2....24
A. Run the BoT-SORT tracker on the five given subsets from MOT17 . Go step by step to get an evaluation report and animation results. Attach the final performance report and animation to the report.....24
B. Compare in high level between them focusing on CLEARMOT, HOTA, IDS , Identity and Detetcion perfomance 24
3. Analysis of results 25
A. In some of the given subsets, the camera is in movement and not static. What might be the challenge in tracking in this case, and how do you suggest handling it? 25
B. ID switch 25
D. Tracking on occlusion/truncated object....28
E. Detection performance....33
Summary....35

Solutions:

Part A Detection analysis based on PointPillars (PP) results on KITTI dataset.

1.

During the backbone, the Feature Encoder (Pillar feature net) converts the point cloud into a sparse pseudo-image. How does the mechanism convert the input from a 3D point cloud into a grid of a specific size? What information does the pseudo-image include?

As can be seen in the image bellow, the pillar feature net, first stores all the input point from the point cloud into stacked pillars where each pillar have D= 9 features which are the regular X,Y,Z,R of the point cloud along with Xc,Yc,Zc which are the distance to the arithmetic mean of all points in the pillar and Xp, Yp which are the offset from the pillar x, y center. Where the point pillars are mostly emphty due to the sparsity of the point cloud, This sparsity is exploited by imposing a limit both on the number of non-empty pillars per sample (P) and on the number of points per pillar (N) to create a dense tensor of size (D, P, N). After this stage they applied a simplified PointNet where, for each point, a linear layer is applied followed by BatchNorm and ReLU to generate a (C, P, N) sized tensor. On top of that they applied a max operation to go from a tensor of size (C,P,N) to size (C,P) which performs as a backbone feature encoder (much like the usage of ResNet backbone in many of the image processing models) Once encoded, the features are scattered back to the original pillar locations to create a pseudo-image of size (C, H, W) where H and W indicate the height and width of the canvas. By applying the above they mange to go from a 3D into a 2D input by doing this they mange to save a lot of the computation since 2D convolutions are easier to compute when compared to a 3D convolutions.

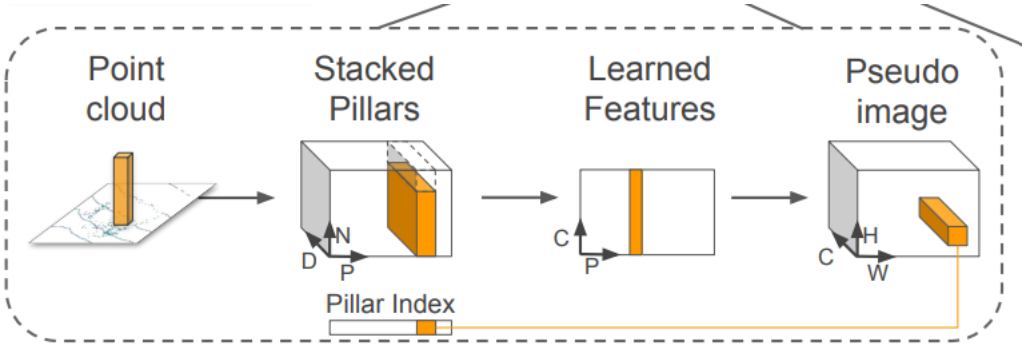

Figure 1 Pillar feature net

2.

Example1:

- High level explanation of the scene: in the scene we can see a car driving through a narrow street with a lot of parking cars as can be seen in Figure 2 which is the RGB image representation of the scene an on Figure 3 which is the PC representation of the scene.
  As we can see the tracker is only focused on the part of the road that the car is in front of the car, an evidence to that can be seen in Figure 4 where all the detections (marked as a 3D bounding box in the scene) are in the front of the car.
  As we can see The issue in this scenario is that there are many cars parked densely and occluding each other which makes it hard for the detector to detect them. As evidence to

that we can see that the cars closer to the sensors front have a high accuracy score and good detection as can be seen on Figure 4 while for a more occluded region of the image we get a lower accuracy score as can be seen on Figure 5 and some cares appear on the image but do not appear on the PC as can be seen when comparing Figure 6 and Figure 7.
On top of that we also received one bad detection where a tree was detected a person as can be seen on Figure 6.

- As explained on the previous section and can be seen on the images bellow what effects the detection confidence is the proximity to the sensor itself (i.e the LiDAR) which in turn effect the amounts of point that can be related to the detection itself. Meaning the closer objects that have a lot of points from the point cloud that describe them will have high accourecy while object that are occluded and have less points from the point cloud describing them have a lower confidence rating.
- For the failure detected (i.e. the failure to detected some of the parked cars) we can understand it as these cars were occluded by the some of the other cars and hence did not have points from the point cloud that pointed to them for example we can compare Figure 2 and Figure 3 and see that some of the cars that appear in the image do not appear in the point cloud view and hence they were not detected.

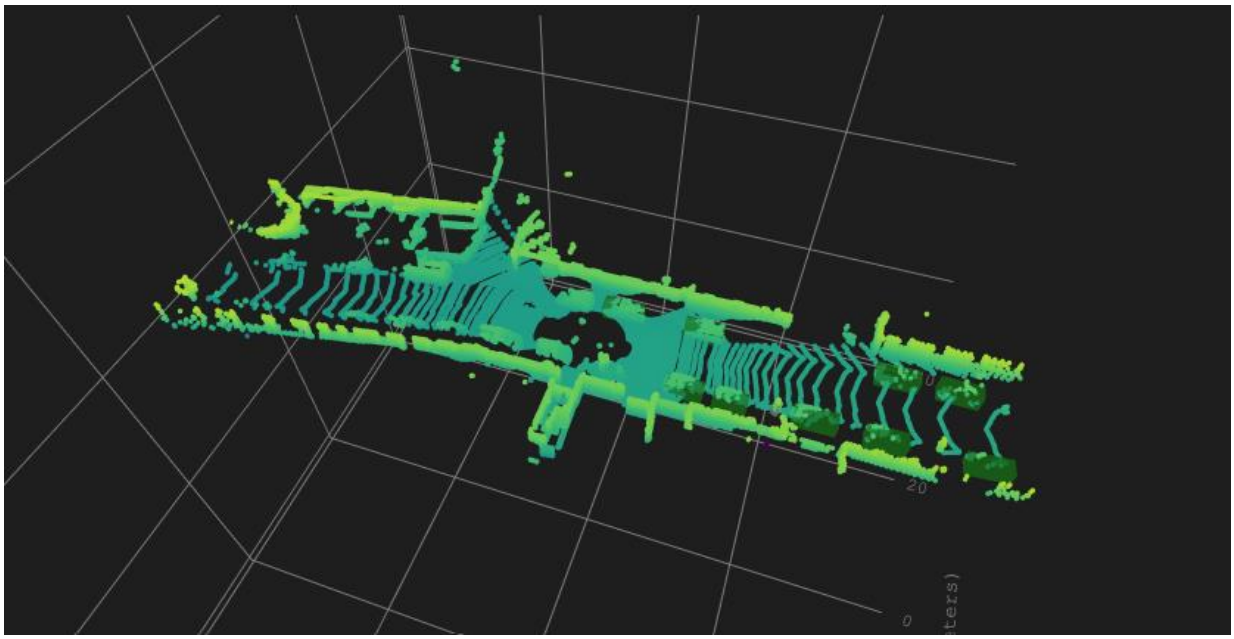

Figure 2 Example1 PC

Figure 3 Example1 image

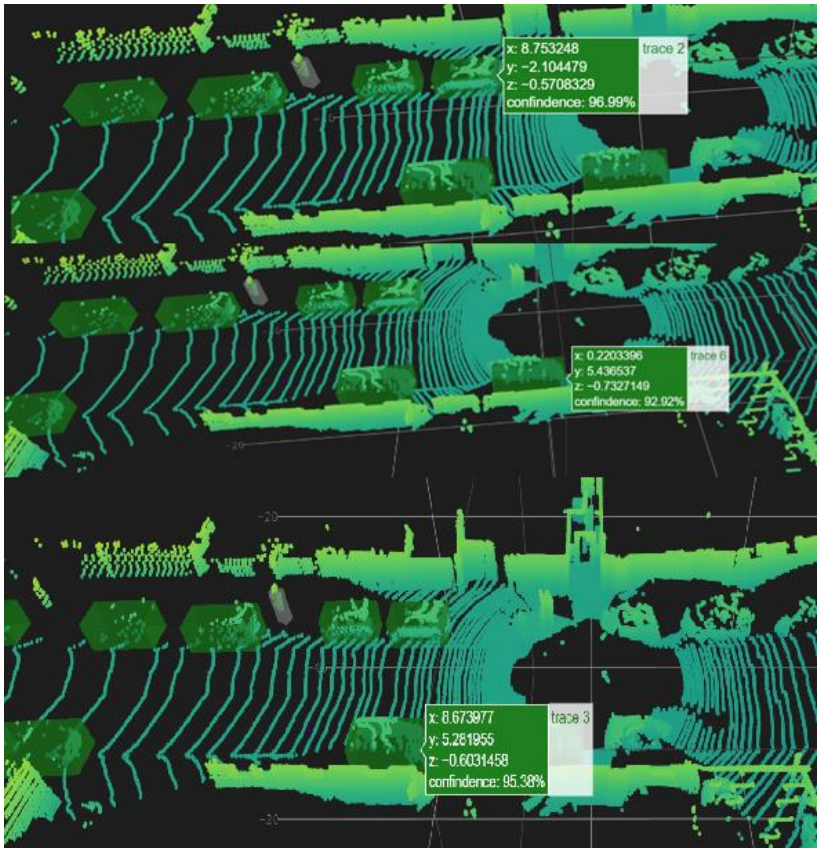

Figure 4 example 1 good detections

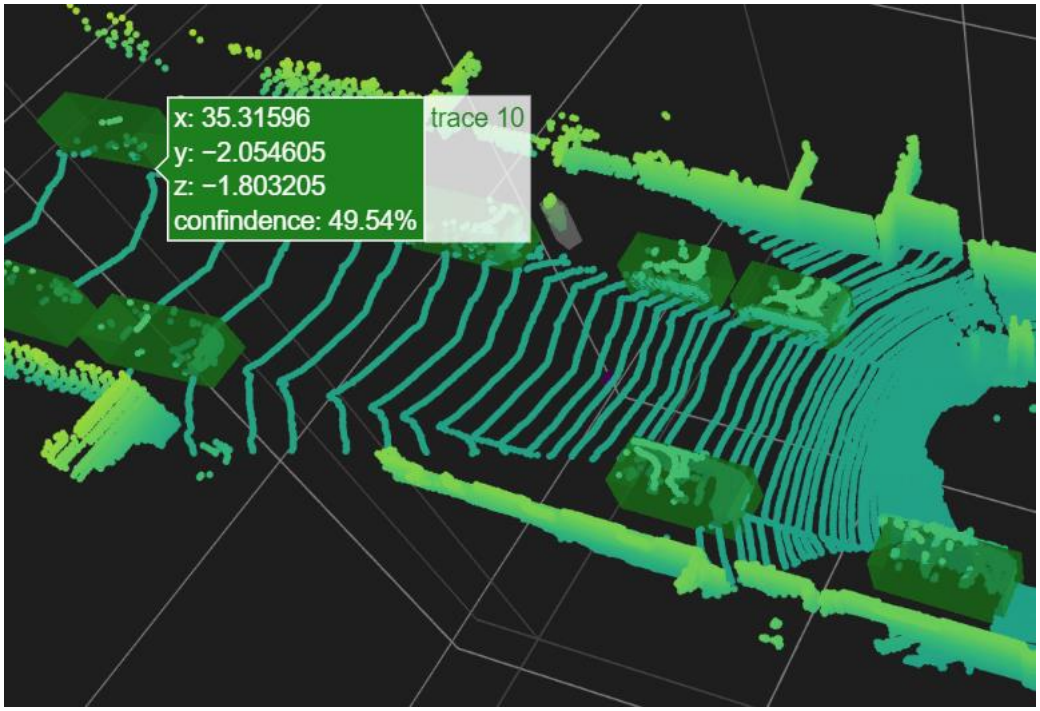

Figure 5 example1 good detection low accuracy

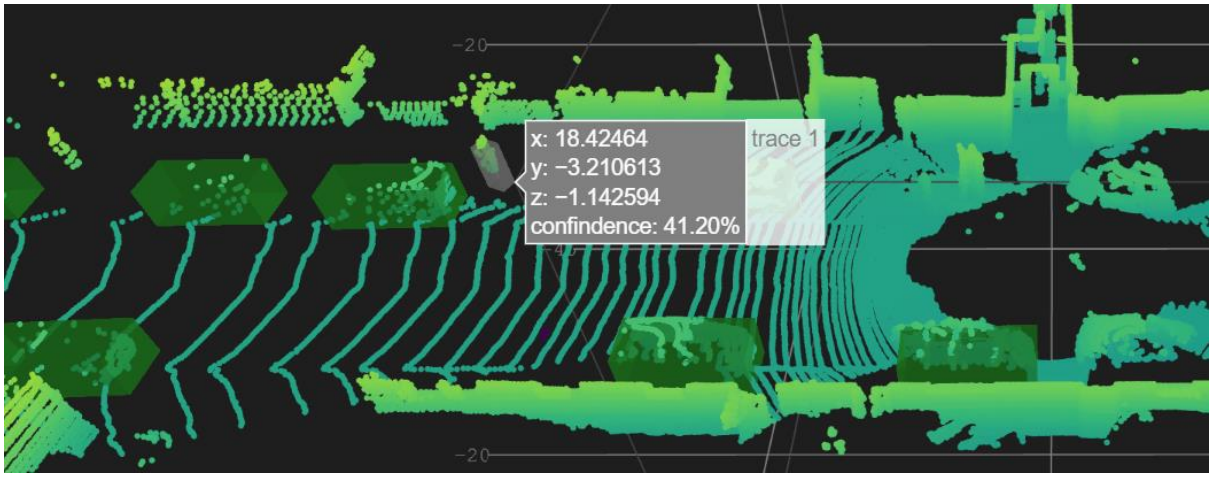

Figure 6 example 1 bad deduction of a pedestrian

Example 2

- As can be seen on Figures 7 and 8 we can see that the described scene is a drive through a main road in the city, we can see that we have good detection for the parked cars on the road and also we have good detection of the car that is in front of us as can be seen on Figure 9. And on top of that we can see that the detector did not detect the van in the image as a car as can be seen on Figure 7 (where the points to right of the van represents the van but there is no detection bounding box on them
- The three main updated that need to be done regarding the calibration of the PointPillars are as follows:
    1. The obvious will be to add a detection class for vans and retrain the model accordingly.

2. If we do not want to retrain the model for vans and leave their class as car, then we will need to increase the detectors anchor size to catch larger cars such as the van.
3. On top of the above 2 option the third and most important thing to do is to validate that in the training dataset there labeled examples of vans and if not retrain the model with a dataset thar has labeled examples of vans (doesn't matter if they are addressed as car or vans as long as they are labeled and model will gain loss by not classifying them well)

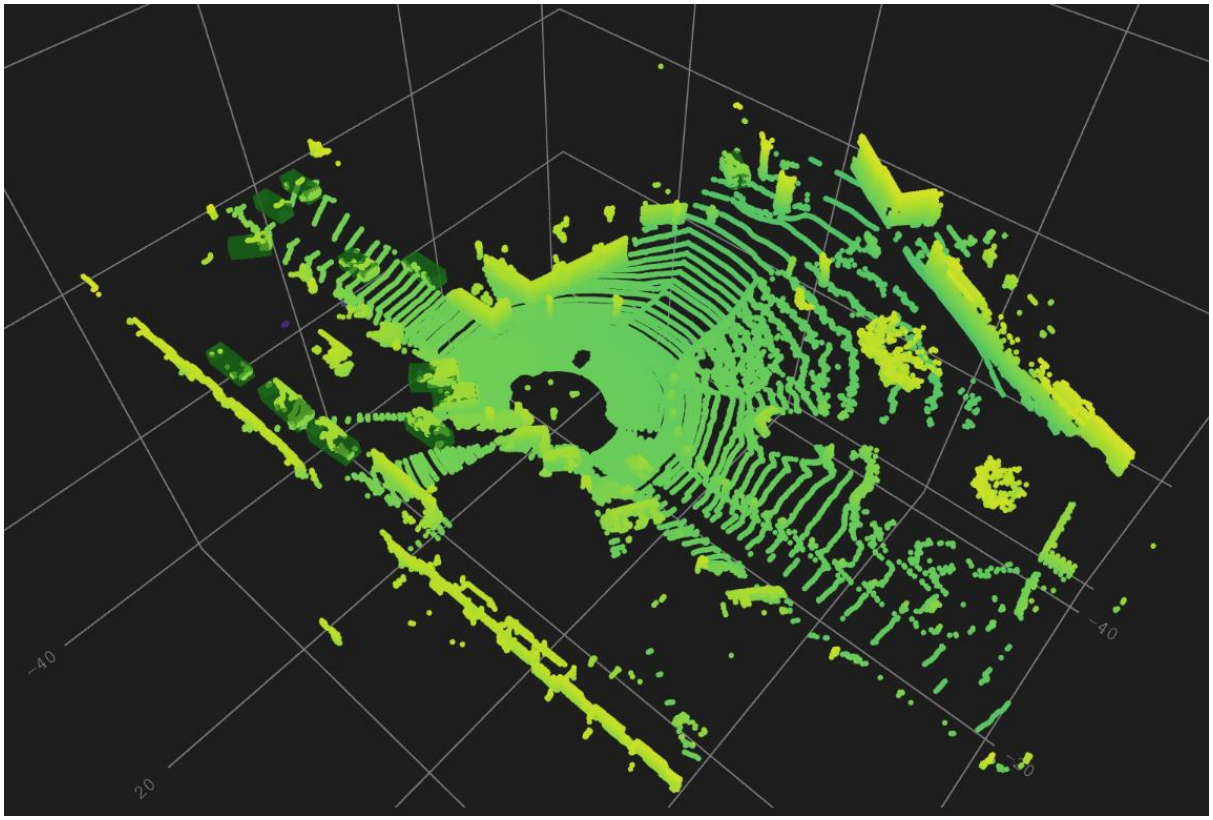

Figure 7 Example 2 PC

Figure 8 Example 2 image

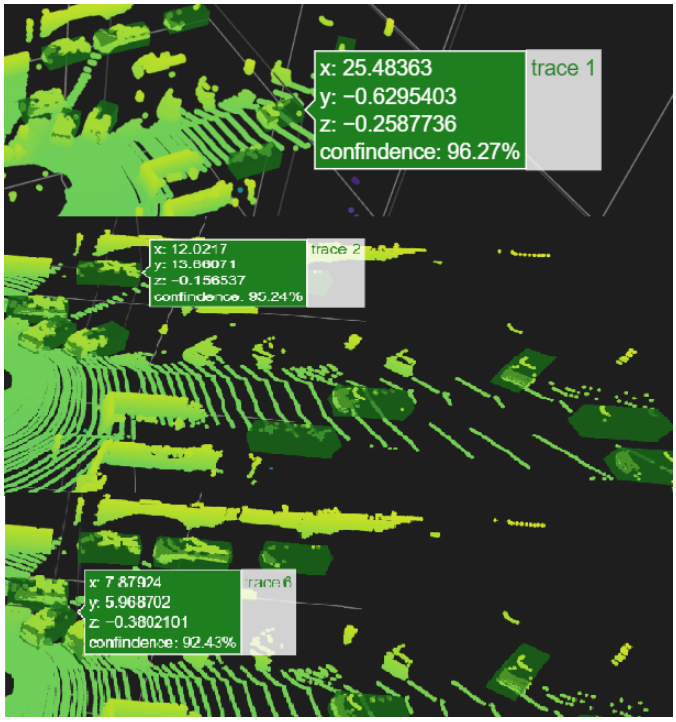

Figure 9 Example 2 good detection

Example 3

- In this scene we can see a crossway on a main road in the city, we can see on Figures 10 and 11 the point cloud of the front end of the car and the corresponding image, overall there seem to be a descent tracking in this seen, the model managed to spot the pedestrian and the cyclist with good accuracy and also the car in front of us on the road with a good accuracy as can be seen on Figure 12 but we also have false detection of a tree confused to be a pedestrian as can be seen on Figure 13 but it got a lower confidence score than the ones that the true positive received.
- The vehicle on the right that is also visible on Figures 10 and 12 is a false negative despite it matching the car dimensions, we can try and explain it by the fact that this car is a type of van (which have a squared back which is a bit different than the usual car back) and as we have seen in Example 2 van detection are not supported on this model hence it was not detected here either. And to be more precise this car is very similar to the Renault kangoo which is a sort of van- car hybrid and since the model does not recognize vans thus it can explain why the model failed to detect it here too.
- The false positive can be seen on Figure 13 as the False detection of a tree as a pedestrian the easiest way to fix this types of mistake would be to incorporate sensor fusion into the model, meaning using an image processing detector such as YOLO fused with the PointPillars model in order to increase the detection confidence and eliminate false positive such as this one.

  Another option to fix this would be perhaps incorporating the reflectance r of each point and try to add it to the model because currently there is no real usage in this added sensor data. An advanced LiDAR sensor (such as the ones Innovis make) has more details with its reflectance data that can be used to differentiate between a tree and a pedestrian.

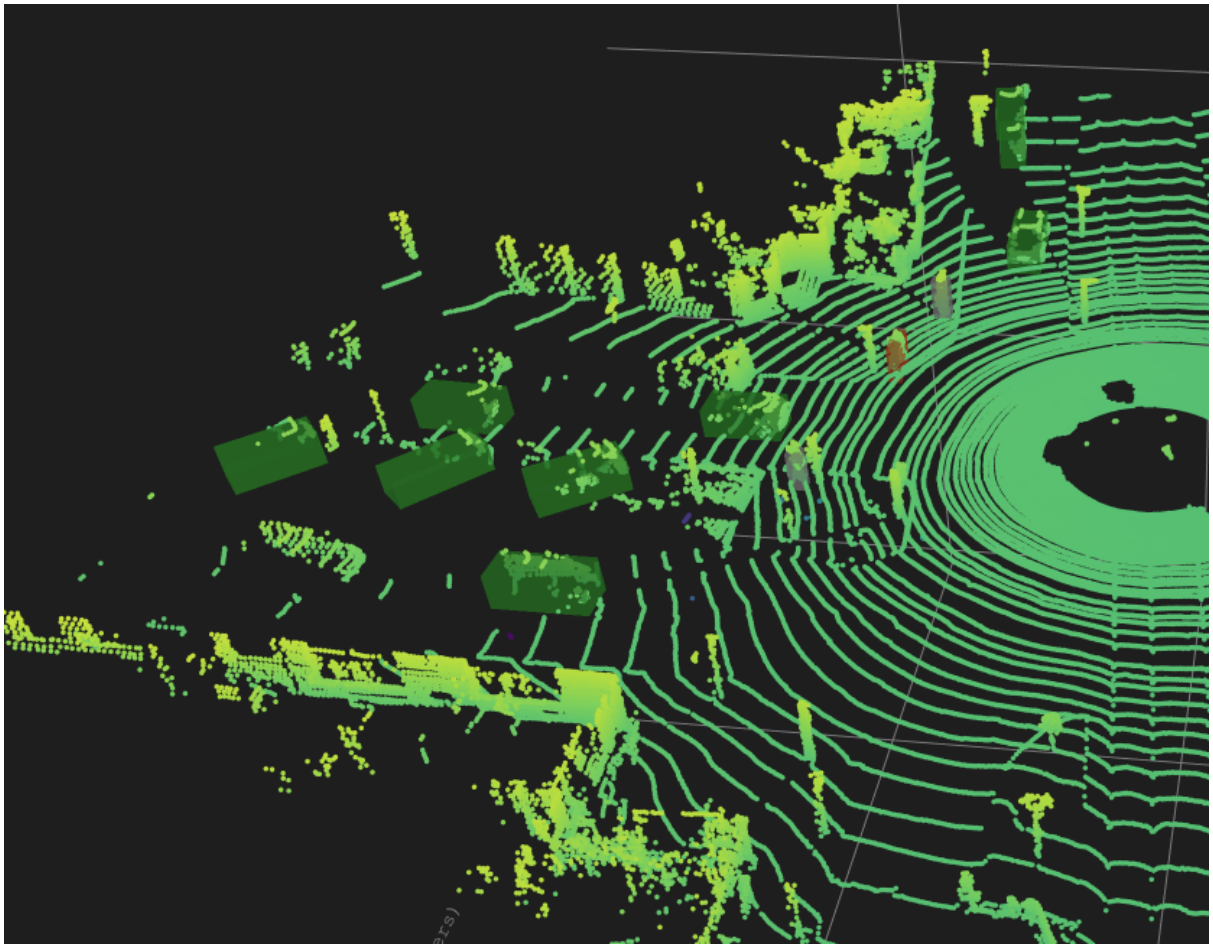

Figure 10 Example 3 PC

Figure 11 Example 3 image

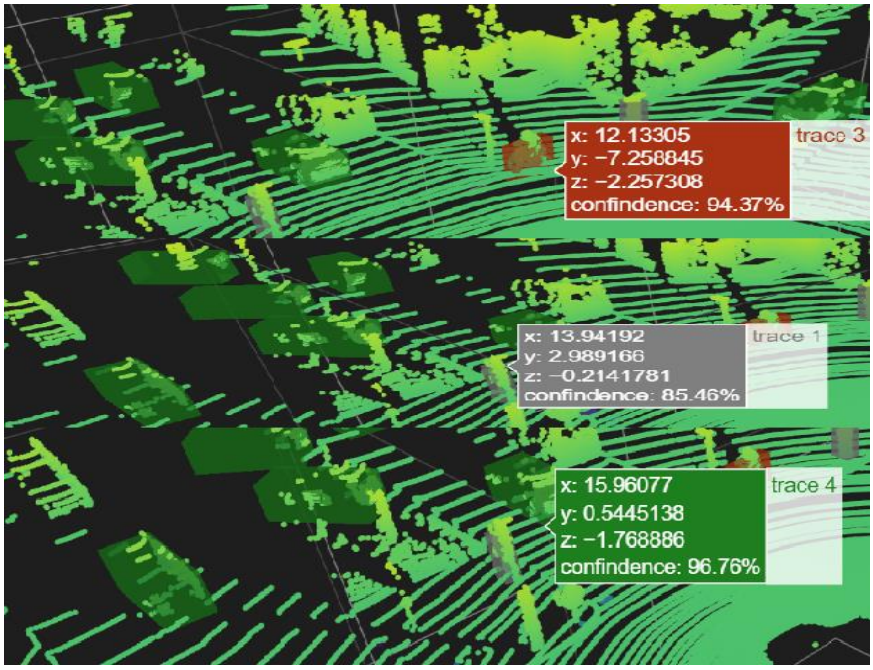

Figure 12 Example 3 good detection

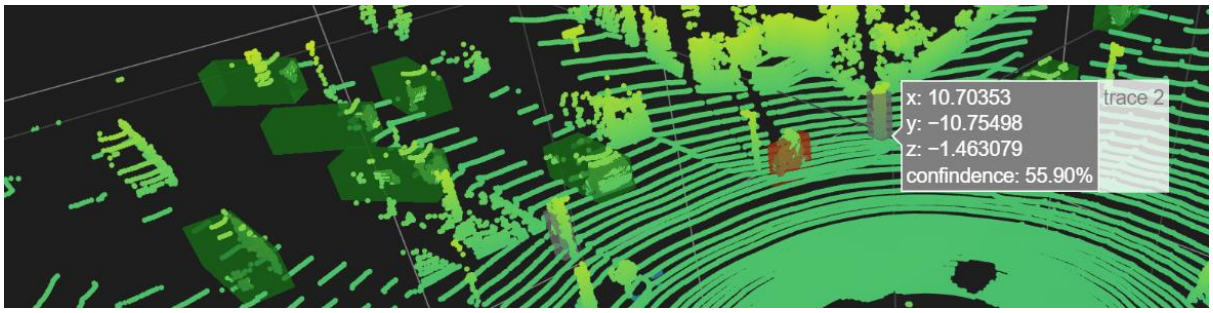

Figure 13 Example 3 false detection

Part B

1. Describe shortly each of the Multi-stage methods based on the paper: Frustum Convnet,Frustum PointNet and Frustum-PointPillars architecture.

Frustum PointNet:

PointNets can classify a whole point cloud or predicting a semantic class for each point in a point cloud, it is unclear how this architecture can be used for instance-level 3D object detection. Towards this goal, they had to address a key challenge: how to efficiently propose possible locations of 3D objects in a 3D space. As sliding window and 3d reginal proposals are computationally expansive. They purposed to take advantage of the mature 2D object detectors. They extract the 3D bounding frustum of an object by extruding 2D bounding boxes from 2D image detectors. Then, within the 3D space trimmed by each of the 3D frustums, they consecutively perform 3D object instance segmentation and amodal 3D bounding box regression using two variants of PointNet. The segmentation network predicts the 3D mask of the object of interest (i.e. instance segmentation), and the regression network estimates the amodal 3D bounding box (covering the entire object even if only part of it is visible).

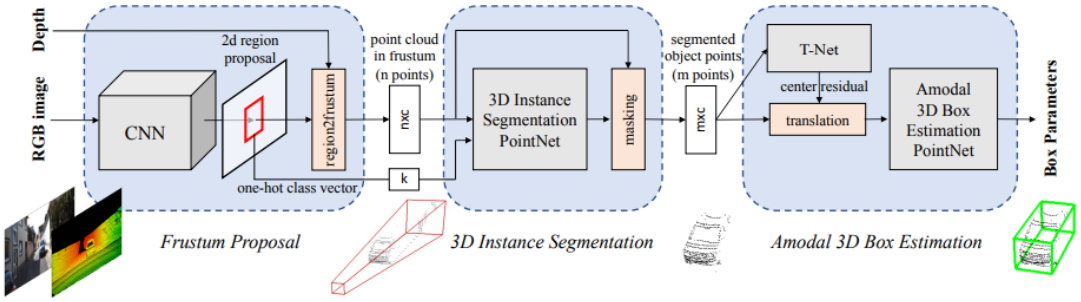

Figure 14 Furstom PointNets for 3D object detection

Frustom pointNet consists of three modules: frustum proposal, 3D instance segmentation, and 3D amodal bounding box estimation

- **Frustum Proposal** - With a known camera projection matrix, a 2D bounding box can be lifted to a frustum (with near and far planes specified by depth sensor range) that defines a 3D search space for the object. They then collect all points within the frustum to form a frustum point cloud. frustums may orient towards many different directions, which result in large variation in the placement of point clouds. We therefore normalize the frustums by rotating them toward a centre view such that the centre axis of the frustum is orthogonal to the image plane. This normalization helps improve the rotation-invariance of the algorithm. while their 3D detection framework is agnostic to the exact method for 2D region proposal, they adopted a FPN based model.
- **3D Instance Segmentation** - Because objects are naturally separated in physical space, segmentation in 3D point cloud is much more natural and easier than that in images where pixels from distant objects can be near-by to each other. Hence they perpose to segment instances in 3D point cloud instead of in 2D image or depth map. They realize 3D instance segmentation using a PointNet-based network on point clouds in frustums. Based on 3D instance segmentation, they are able to achieve residual based 3D localization. That is, rather than regressing the absolute 3D location of the object whose offset from the sensor may vary in large ranges they predict the 3D bounding box center in a local coordinate system – 3D mask coordinates as shown in the next figure(c):

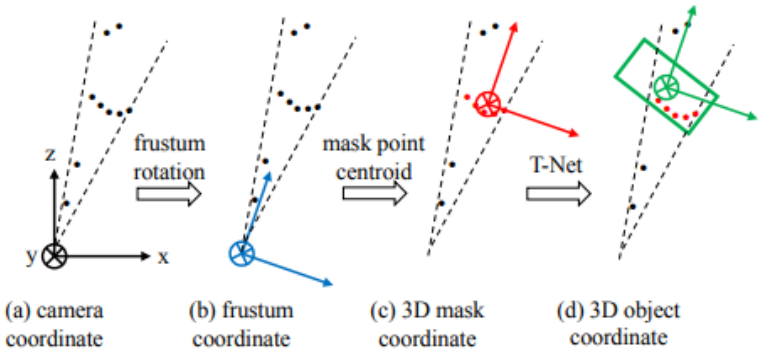

Figure 15Coordinate systems for point cloud

The network takes a point cloud in frustum and predicts a probability score for each point that indicates how likely the point belongs to the object of interest. Note that each frustum contains exactly one object of interest. Here those “other” points could

be points of non-relevant areas (such as ground, vegetation) or other instances that occlude or are behind the object of interest. Similar to the case in 2D instance segmentation, depending on the position of the frustum, object points in one frustum may become cluttered or occlude points in another. Therefore, their segmentation PointNet is learning the occlusion and clutter patterns as well as recognizing the geometry for the object of a certain category. In a multi-class detection case, they also leverage the semantics from a 2D detector for better instance segmentation. For example, if they know the object of interest is a pedestrian, then the segmentation network can use this prior to find geometries that look like a person. Specifically, in their architecture they encode the semantic category as a one-hot class vector (k dimensional for the pre-defined k categories) and concatenate the one-hot vector to the intermediate point cloud features. After 3D instance segmentation, points that are classified as the object of interest are extracted.(masking step) Having obtained these segmented object points, they further normalize its coordinates to boost the translational invariance of the algorithm, following the same rationale as in the frustum proposal step. In their implementation, they transform the point cloud into a local coordinate by subtracting XYZ values by its centroid.

- **Amodal 3D Box Estimation PointNet** - The box estimation network predicts amodal bounding boxes (for entire object even if part of it is unseen) for objects given an object point cloud in 3D object coordinate(d in above figure) The network architecture is similar to that for object classification, however the output is no longer object class scores but parameters for a 3D bounding box.
  They parameterize a 3D bounding box by its centre(c_x, c_y, c_z), size (h, w, l) and heading angle θ (along up-axis). They take a "residual" approach for box center estimation. The center residual predicted by the box estimation network is combined with the previous centre residual from the T-Net and the masked points' centroid to recover an absolute centre.

$$C_{pred} = C_{mask} + \Delta C_{t-net} + \Delta C_{box-net}$$

  For box size and heading angle, they use a hybrid of classification and regression formulations. Specifically, they pre-define NS size templates and NH equally split angle bins. Their model both classifies size/heading (NS scores for size, NH scores for heading) to those pre-defined categories as well as predict residual numbers for each category (3×NS residual dimensions for height, width, length, NH residual angles for heading). In the end the net outputs 3 + 4 × NS + 2 × NH numbers in total.

The Loss function:

$$L_{multi-task} = L_{seg} + \lambda(L_{c1-reg} + L_{c2-reg} + L_{h-cls} + L_{h-reg} + L_{s-cls} + L_{s-reg} + \gamma L_{corner})$$

simultaneously optimize the three nets involved (3D instance segmentation PointNet, T-Net and amodal box estimation PointNet) with multi-task losses

L_{c1-reg} is for T-Net and L_{c2-reg} is for center regression of box estimation net. L_{h-cls} and L_{h-reg} are losses for heading angle prediction while L_{s-cls} and L_{s-reg} are for box size. Softmax is used for all classification tasks and smooth-l_1 (huber) loss is used for all regression cases.

And L_{corner} defined as:

$$L_{corner} = \sum_{i=1}^{NS}\sum_{j=1}^{NH}\delta_{ij}\min\{\sum_{k=1}^{8}\|P_k^{ij} - P_k^{*}\|, \sum_{i=1}^{8}\|P_k^{ij} - P_k^{**}\|\}$$

they denote the anchor box corners as P_k^{ij}, where i, j, k are indices for the size class, heading class, and (predefined) corner order, respectively. In essence, the corner loss is the sum of the distances between the eight corners of a predicted box and a ground truth box. Since corner positions are jointly determined by centre, size and heading, the corner loss is able to regularize the multi-task training for those parameters.

Frustum Convnet:

Given 2D region proposals in an RGB image, their method first generates a sequence of frustums for each region proposal and uses the obtained frustums to group local points. F-ConvNet aggregates point-wise features as frustum level feature vectors and arrays these feature vectors as a feature map for use of its subsequent component of fully convolutional network (FCN), which spatially fuses frustum-level features and supports an end-to-end and continuous estimation of oriented boxes in the 3D space.

Their methode assumes the availability of 2D region proposals in RGB images, which can be easily obtained from off-the-shelf object detectors, and identifies 3D points corresponding to pixels inside each region proposal their method generates for each region proposal a sequence of (possibly overlapped) frustums by sliding along the frustum axis as can be seen in the next figure:

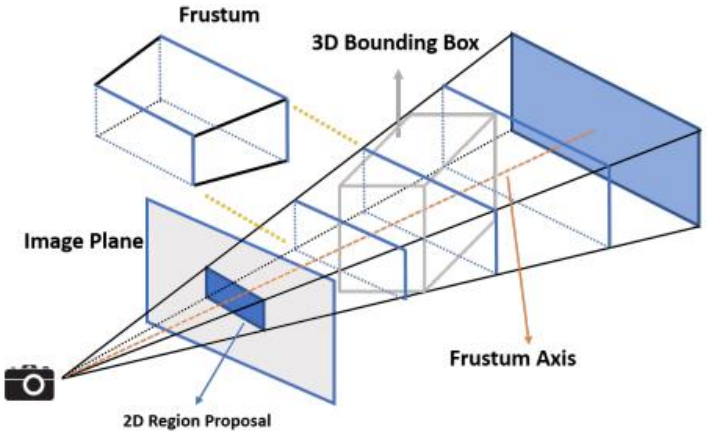

Figure 16 Illustration for how a sequence of frustums are generated for a region proposal in an RGB image.

These obtained frustums define groups of local points. Given the sequence of frustums and point association, the F-ConvNet starts with lower, parallel layer streams of PointNet style to aggregate point-wise features as a frustum-level feature vector; it then arrays at its early stage these feature vectors of individual frustums as 2D feature maps, and uses a subsequent fully convolutional network (FCN) to down-sample and up-sample frustums such that their features are fully fused across the frustum axis at a higher frustum resolution. Together with a final detection header their proposed F-ConvNet supports an end-to-end and continuous

estimation of oriented 3D boxes, where they also propose an FCN variant that extracts multi-resolution frustum features. Given an initial estimation of 3D box, a final refinement using the same F-ConvNet often improves the performance further.

Design of F-ConvNet centers on the notion of square frustum, and a sequence of frustums along the same frustum axis connect a cloud of discrete, unordered points with an FCN that enables oriented 3D box estimation in a continuous 3D space. By assuming the availability of 2D region proposals in RGB images, they first introduce a way of point association with sequences of (possibly overlapped) frustums that are obtained by sliding along frustum axes determined by 2D region proposals and compare with alternative ways of point association/grouping. They then present the architecture of F-ConvNet and specify how point-wise features inside individual frustums are aggregated and re-formed as 2D feature maps for a continuous frustum-level feature fusion and 3D box estimation.

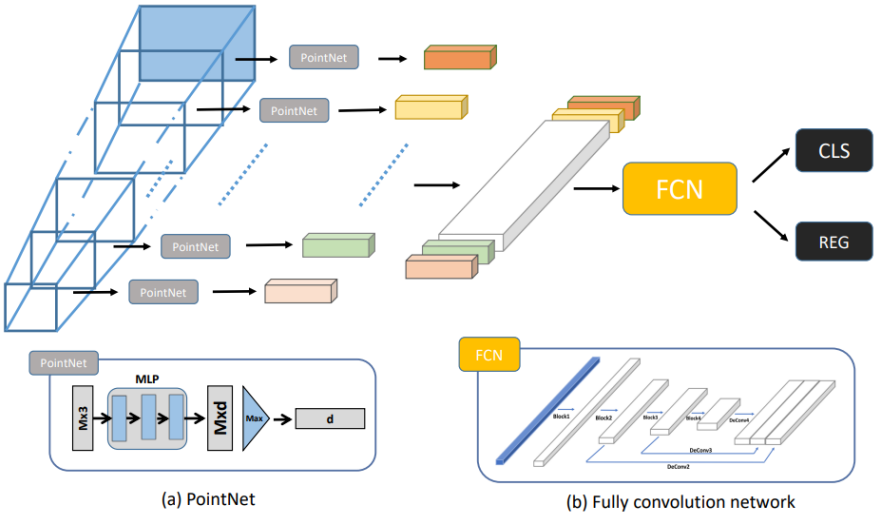

Figure 17 the whole frame work of F-convNet

Associating Point Clouds with Sliding Frustums:

A sequence of (possibly overlapped) frustums can be obtained by sliding a pair of parallel planes along the frustum axis with an equal stride, where the pair of planes are also perpendicular to the frustum axis. As can be seen in the next figure:

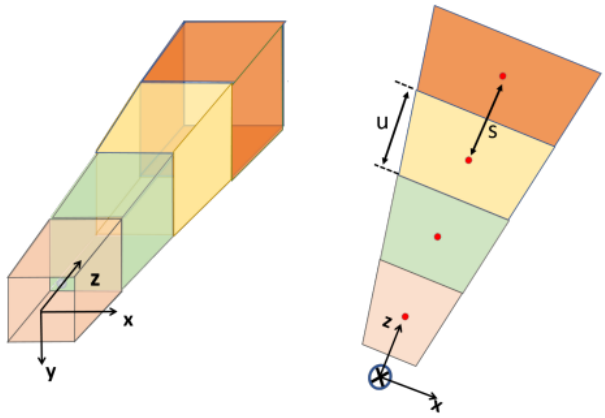

Figure 18 An illustration of frustums(non-overlapped)

They generate such a sequence of frustums for each 2D region proposal, and they use this obtained frustum sequences to group points, i.e., points falling inside the same frustums are grouped together. Assuming that 2D region proposals are accurate enough, their frustums mainly contain foreground points, and are aware of object boundaries. they note that for each 2D region proposal, a single frustum of larger size (defined by the image plane and the farthest plane) is generated and all points falling inside this frustum are grouped together; consequently, an initial stage of foreground point segmentation has to be performed before amodal 3D box estimation. In contrast, they generate for each region proposal a sequence of frustums whose feature vectors are arrayed as a feature map and used in a subsequent FCN for an end-to-end estimation of oriented boxes in the continuous 3D space.

The Architecture of Frustum ConvNet:

Given a sequence of frustums generated from a region proposal, the key design of an F-ConvNet is to aggregate at its early stage point-wise features inside each frustum as a frustum-level feature vector, and then array as a 2D feature map these feature vectors of individual frustums for use of a subsequent FCN, which, together with a detection header, supports an end-to-end and continuous estimation of oriented 3D boxes.

After generated the region perposals of the forstrums each one is inputted into Pointnet with shared weights to extract its global features instead of using the points directly as input they input the points shited by the forstrom center. These global features of size d are arrayed to form a 2D feature map Lxd (L is the number of feature vectors) which will be used as input of a subsequent FCN their FCN consists of blocks of conv layers, and de-conv layers corresponding to each block. Convolution in conv layers is applied across the frustum dimension by using kernels of the size 3 × d. The final layer of each of the conv blocks, except the first block, also down-samples (halves) the 2D feature map at the frustum dimension by using stride-2 convolution. Convolution and down-sampling fuse features across frustums and produce at different conv blocks virtual frustums of varying heights (along the frustum axis direction). Given output feature map of each conv block, a corresponding de-conv layer is used that up-samples at the frustum dimension the feature map to a specified (highet) resolution, outputs of all de-conv layers are then concatenated together along the feature dimension. Feature concatenation from virtual frustums of varying sizes provides a hierarchical granularity of frustum covering, which would be useful to estimate 3D boxes of object instances whose sizes are unknown and vary.

Detection Header and Training of Frustum ConvNet

On top of FCN is the detection header composed of two, parallel conv layers. They are respectively used as the classification and regression branches. The whole F-ConvNet is trained using a multi-task fashion, similar to those in 2D object detection.

Frustum-PointPillars:

F-PointPillars leverages 2D detections in RGB images, to reduce search space in 3D. 2D detections can be obtained from any of the available state-of-the-art object detectors. The 2D detections of the objects are extruded into 3D space creating a 3D bounding frustum of the object as shown in the next figure

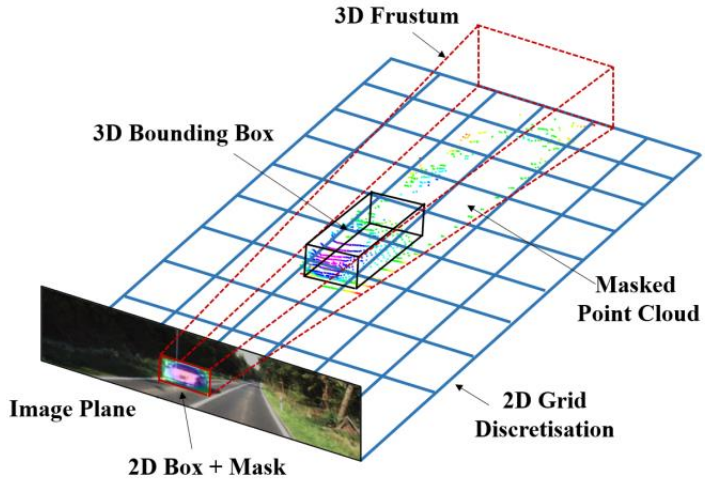

Figure 19 Overview of Frustum-PointPillars architecture

they then discretised the 3D frustums into a bird-eye-view (BEV) 2D grid to extract features at fine resolution This enables to accurately localize smaller objects within the 3D frustum

FRUSTUM POINTPILLARS ARCHITECTURE

Given the 2D detections of the objects in the scene and their corresponding frustum volume, F-PointPillars subdivides the 3D space within the frustum volume into a topview, finer 2D grid as shown in the figure above. Unlike F-ConvNet, which subdivides the 3D bounding frustum into a sequence of smaller frustums. They use a higher resolution for feature extraction. They use PointNet on each cell of the grid to extract pillar features and use 2D convolutions to extract spatial features. They then use a single-stage, anchor-based method to predict the 3D bounding box for all the objects in the scene simultaneously.

F-PointPillars consists of 4 main stages as can be seen in the next figure:

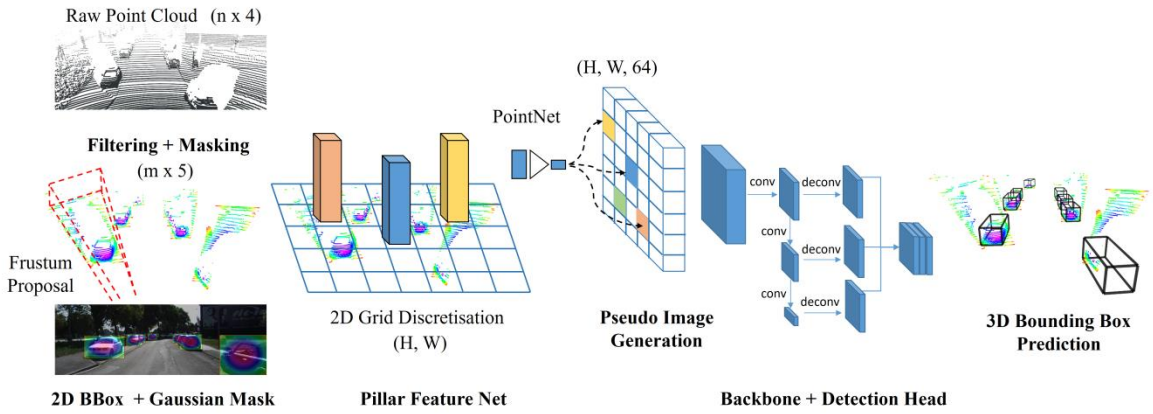

Figure 20 Frustum-PointPillars architecture

(1) Frustum proposal; (2) PointCloud masking; (3) Pillar feature encoding network that converts a point cloud to a sparse pseudo image; and (4) Backbone to process the pseudo-image and produce a high-level representation, and a detection head to regress position, orientation and size of 3D bounding boxes.

Frustum Proposal - With a known camera projection matrix, a 2D bounding box can be extruded to a frustum (with near and far planes specified by depth sensor range) that defines a 3D search space for the object. Another way of viewing this is that, given the camera and LiDAR calibration parameters, 3D LiDAR points can be projected onto the image plane. We then filter out all the points outside of the 2D bounding boxes, significantly reducing the number of points to be processed within a scene.

Point Cloud Masking - The 2D bounding boxes predicted by the 2D detector are designed to enclose the object. The points inside 2D boxes can belong to the object itself with some foreground and background clutter. Pedestrians are generally on the sidewalks, in groups, and close to other buildings and objects. In the case of pedestrians, foreground and background clutter are not well Isolated. To adress this issue they to use a probability mask. The region near the center of a 2d bounding box is more likely to be occupied by the object, the projected 3d points near the center region are also more likely to belong to the object instead of the background clutter. They define the likelihood of the points belonging to the object as a Gaussian function:

$$\mathcal{L}(\bar{x}, \bar{y}) = \exp\left(-\frac{(\bar{x} - \bar{x}_0)^2}{2w^2} - \frac{(\bar{y} - \bar{y}_0)^2}{2h^2}\right)$$

where $\bar{x}, \bar{y}$ are point cloud projection on image plane, $\bar{x}_0, \bar{y}_0$ are the center coordinates and w, h are the width and height of the 2D bounding box. $\alpha = w^2$, $\beta = h^2$ define the curvature of the likelihood function. They add the likelihood value to the point P as an additional feature vector. Input point feature C_{in} is now D = 5 dimensional (x, y, z, intensity, L). If a point is shared by multiple 2D bounding boxes, then the highest likelihood value is chosen.

Pillar Feature Encoding and Pseudo Image - After filtering out the points outside of frustum areas and masking the remaining points, they divide the 3D space into 2D grid of shape (H, W) and the cell resolution of r. The points in the non-empty cells are re-sampled to a fixed

number N. The points in each pillar are further augmented by adding extra features: x_c, y_c, z_c, x_p and y_p where the c subscript denotes distance to the arithmetic mean of all points in the pillar and the p subscript denotes the offset from the pillar center. Input point feature C_{in} is now D = 10 dimensional. A simplified version of PointNet with (N, C_{in}) as input and (1, C_{out}) as output is used to extract features per non-empty cell of the grid. These cell wise features or pillar features are tensors of size C_{out} creating a pseudo-image of size (H, W, C_{out}). Meaning each grid cell has a global feature retreved from the pointnet and of constant size C_{out}.

Backbone and Detection Head - To extract spatial features, they use a similar backbone as point pillars. The backbone consist of two sub-network: one topdown network that uses sequence of 2D convolution blocks to produce features at increasingly small spatial resolution. The second network performs upsampling using transposed 2D convolutions and concatenation of the top-down features. For the detection head they use a framework similar to the Single Shot Detector (SSD) setup. They match the prior anchor boxes to the ground truth using 2D Intersection over Union (IoU). Given a positive 2D match, bounding box parameters are regressed, the height and elevation become additional regression targets. As seen in the above figure.

Loss function: Similar to pointpillars they use combination of losses for the training of Frustum-PointPillars. 1) L_{loc}: SmoothL1 loss is used for regression of residuals between ground truth and anchors, 2) L_{dir}: a Softmax classification loss is used on the discretized directions to distinguished between flipped boxes and 3)L_{cls}: a Focal loss for object classification. The total loss is:

$$L_{total} = (\beta_{loc} L_{loc} + \beta_{cls} L_{cls} + \beta_{dir} L_{dir})$$

where, β_{loc}, β_{cls} and β_{dir} are the hyper parameters to give weightage for different losses.

PointPillar: As specified before a main issue is that the point cloud is not bonded which is a big issue hence in point pillar approach they take the unordered point cloud and organize it into a pseudo image (created a 3d image) now that we have an organized data it can be inserted to a 2D CNN backbone and perform detection (via detection head SSD) and get perdictions ad can be seen in the next figure:

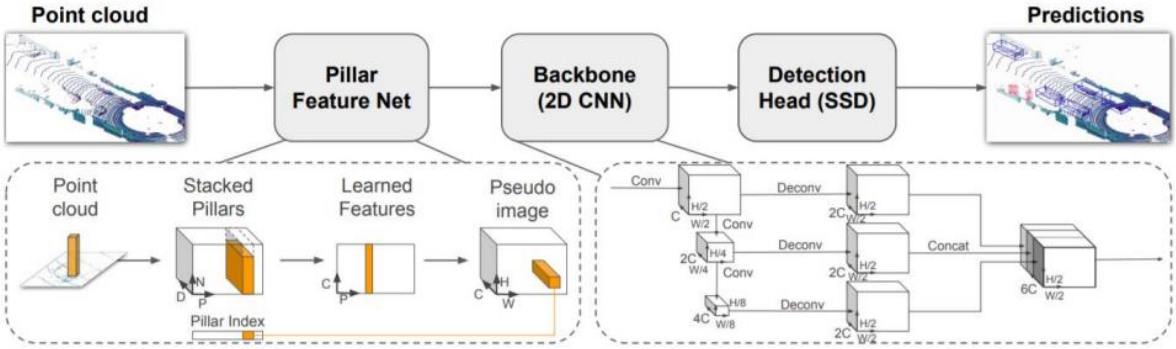

Figure 21 point pillar network overview

the left part, pillar feature net converts sparse point cloud(which has issues with sparsity data , object occlusion and sensor noise) to pillars. By deviding the space into a grid and learning vertical pillars of points (each pillar could have many point cloud points) they then

insert each point pillar points into a Pointnet which will give us global features of each pillar as can be seen in the next figure:

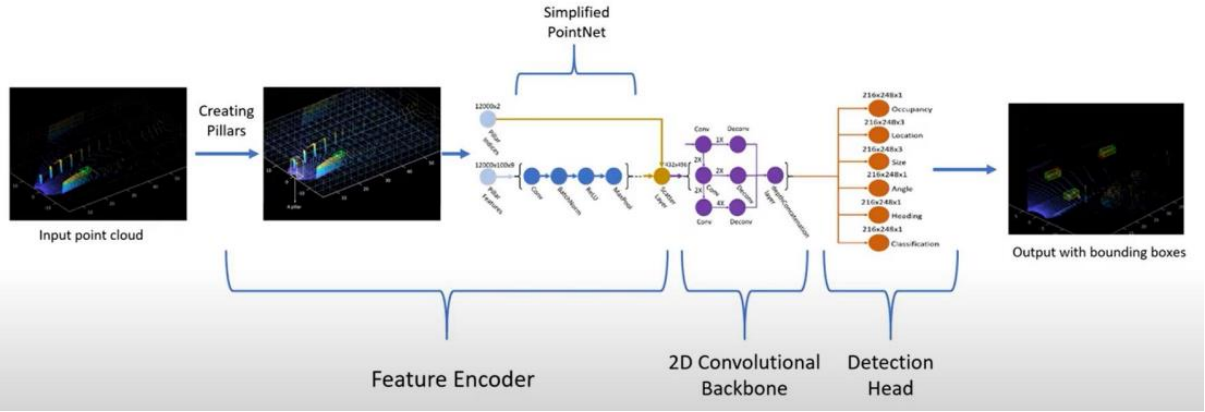

Figure 22 point pillar pipeline

Each point in the point cloud is represented with 9 dimesons: (x,y,z) coordinates of L from origin, (x_c,y_c,z_c) distance to arthmetic mean of all points in a pillar, (x_p,y_p) offset from pillar center as can be seen in the next figure:

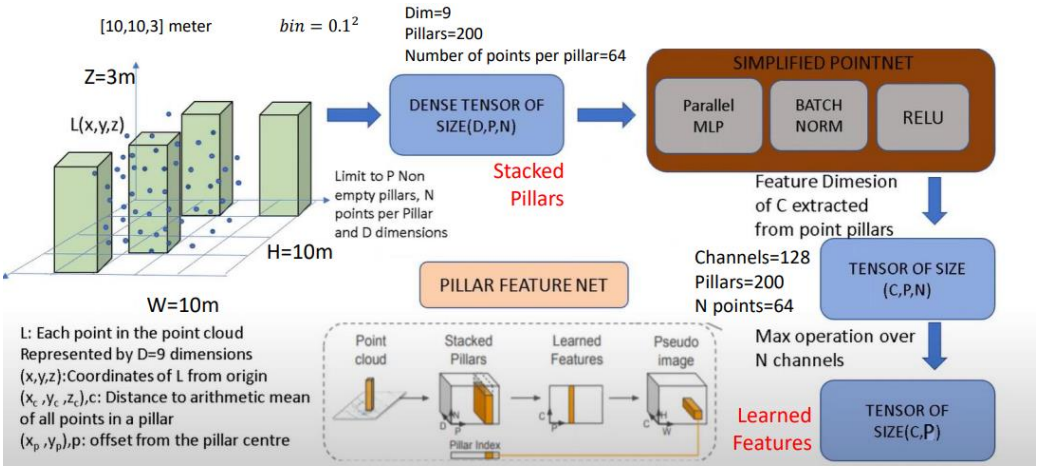

Figure 23 point pillar pipeline example

After using pointnet we get constant pillars which is now an organized pseudo image of learned features

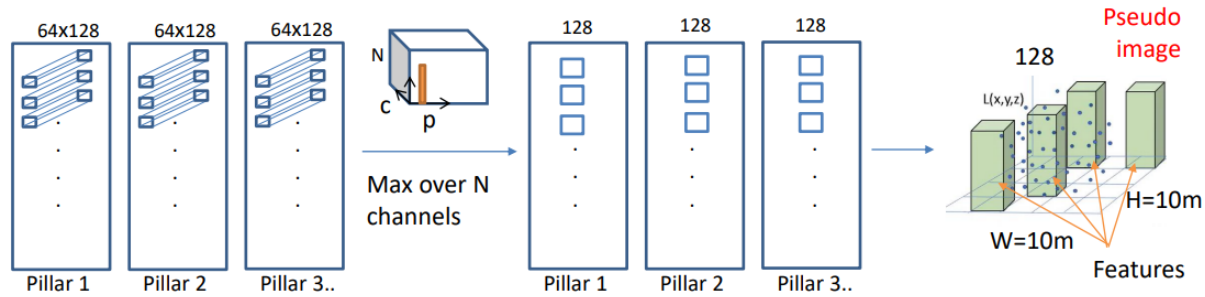

Figure 24 bounded pillars feature at output.

This pseudo image is now inserted to the 2d detector convolutional backbone and from there to the detector head.

2. Mention at least 2 advantages of Frustum-PointPillars architecture compare to PointPillar approuch.

As seen in theoretical backround:

1) F-point pillar utilizes the regions of interest using a 2D region proposal from an RGB image to create a frustum proposal and only after masking it is divided into pillars as in point pillars this enables to accurately localize smaller objects within the 3D frustum. This reduces the search space in 3D.

2) In addition to the 9-dimensional point input furstrom point pillars adds an additional feature of the gaussian likelihood of a point to belong to the background/ foreground making each point inputted to the point net a 10D vector allowing better separation between the wanted object and its surroundings.

3.

A.What is the mechanism described in the paper to isoltaed foreground and background clutter?

As mentioned in the description of the F-pointpillar architecture the paper purposed a likelihood to every point in the frustum adding an additional likelihood value to the point P as an additional feature vector. This likely hood is computed via a guassian function:

$$\mathcal{L}(\bar{x}, \bar{y}) = \exp\left(-\frac{(\bar{x} - \bar{x}_0)^2}{2w^2} - \frac{(\bar{y} - \bar{y}_0)^2}{2h^2}\right)$$

where $\bar{x}, \bar{y}$ are point cloud projection on image plane, $\bar{x}_0, \bar{y}_0$ are the center coordinates and w, h are the width and height of the 2D bounding box. $\alpha = w^2$, $\beta = h^2$ define the curvature of the likelihood function. Now the Input point feature C_{in} is now D = 5 dimensional (x, y, z, intensity, L). And If a point is shared by multiple 2D bounding boxes, then the highest likelihood value is chosen.

They assume the region near the center of a 2d bounding box is more likely to be occupied by the object, the projected 3d points near the center region are also more likely to belong to the object instead of the background clutter.

B. In any case foreground and background clutter are not well Isolated?

An alternative approach is to use mask RCNN as a region proposal for the forstrom instead of the bounding box perposal, to reduce the clutter. however, it can worsen the overall performance of 3D object detection due to the accumulated error from inaccurate object mask predictions. And low runtime.

Part C: Multi Object Tracking

1

A: Describe the main mechanism of the SORT algorithm.

The main mechanism of the SORT algorithm is to apply a Kalman filter on top of a detector (FrRCNN(VGG16)) they interduce the filter state vector as the following :

x = [u, v, s, r, u,˙ v,˙ s˙] T

where u and v represent the horizontal and vertical pixel location of the center of the target, while the scale s and r represent the scale (area) and the aspect ratio of the target's bounding box respectively. Note that the aspect ratio is constant. When a detection is associated to a target, the detected bounding box is used to update the target state where the velocity components are solved optimally via a Kalman filter framework. If no detection is associated to the target, its state is simply predicted without correction using the linear velocity model.
In order to assign detections, a cost matrix is then computed as the intersection-over-union (IOU) distance between each detection and all predicted bounding boxes from the existing targets. The assignment is solved optimally using the Hungarian algorithm.
They used the IOU distance of the bounding boxes since it implicitly handles short term occlusion caused by passing targets. Specifically, when a target is covered by an occluding object, only the occluder is detected, since the IOU distance appropriately favours detections with similar scale. This allows both the occluder target to be corrected with the detection while the covered target is unaffected as no assignment is made.
When objects enter and leave the image, unique identities need to be created or destroyed accordingly. For creating trackers, they consider any detection with an overlap less than IOUmin to signify the existence of an untracked object. The tracker is initialised using the geometry of the bounding box with the velocity set to zero. Since the velocity is unobserved at this point the covariance of the velocity component is initialised with large values, reflecting this uncertainty. Additionally, the new tracker then undergoes a probationary period where the target needs to be associated with detections to accumulate enough evidence to prevent tracking of false positives. Tracks are terminated if they are not detected for TLost frames. This prevents an unbounded growth in the number of trackers and localisation errors caused by predictions over long durations without corrections from the detector. In all experiments TLost is set to 1 for two reasons. Firstly, the constant velocity model is a poor predictor of the true dynamics and secondly in this article they were primarily concerned with frame-to-frame tracking where object re-identification is beyond the scope of this work. Should an object reappear, tracking will implicitly resume under a new identity.

B. Describe the main benefit of the DeepSort algorithm to the SORT process. Elaborate on RE-ID in your explanation.

The main benefit of the DeepSort to the SORT algorithm is by incorporating appearance information through a pre-trained association metric, with this extension they are able to track through longer periods of occlusion, making SORT a strong competitor to state-of-the-art online tracking algorithms. Yet, the algorithm remains simple to implement and runs in real time. By applying this appearance information along with the motion estimation received from the Kalman filter (i.e. computing for each tracked object its Mahalanobis distance).In summary they were able to track objects through multiple frames and handle occlusion for longer time periods which is a big improvement from the regular SORT algorithm, which as we explained above stop tracking each object that was occluded for more then 1 frame.

C. Describe at least two good additions to ByteTrack for DeepSort.

1. On ByteTrack they added the usage of low score detection boxes and tried to match all these low score detections with the remaining tracks that weren't matched with any of the high scores detection.
2. When matching the low scores detection with the remaining un matched tracks (i.e the tracks that were not matched with the high scores detections) they used the IoU alone as the Similarity#2 in the second association because the low score detection boxes usually contains severe occlusion or motion blur and appearance features are not reliable.

So it is a good addition to the algorithm since in DeepSort they used the same distance measure on all the detections.

D. Describe at least three good additions to BoT-SORT compared to ByteTrack.

1. Camera Motion Compensation (CMC): They applied a global motion compensation (GMC) technique used in the OpenCV implementation of the Video Stabilization module with affine transformation. In order to account for the case of a moving camera (like for example on autonomous robot or for the case that a static camera that is affected due to motion by vibrations or drifts caused by the wind. By applying the video stabilization they have a better overlap between frames which helps solve the problem that arise in a dynamic camera situation, where the bounding box location in the image plane can shift dramatically, which might result in increasing ID switches or false negatives.
2. IoU - Re-ID Fusion: They decided to abandon the common weighted sum between the appearance cost Aa and motion cost Am for calculating the cost matrix C
   C = λAa + (1 − λ)Am,
   Where the weight factor λ is usually set to 0.98. they instead developed a new method for combining the motion and the appearance information, i.e. the IoU distance matrix and the cosine distance matrix. First, low cosine similarity or far away candidates, in terms of IoU's score, are rejected. Then, they use the minimum in each element of the matrices as the final value of our cost matrix C. Theyre IoU-ReID fusion pipeline can be formulated as follows:

$$\hat{d}_{i,j}^{cos} = \begin{cases} 0.5 \cdot d_{i,j}^{cos}, (d_{i,j}^{cos} < \theta_{emb}) \wedge (d_{i,j}^{iou} < \theta_{iou}) \\ 1, \text{otherwise} \end{cases}$$

$$C_{i,j} = min\{d_{i,j}^{iou}, \hat{d}_{i,j}^{cos}\}$$

   Where Ci,j is the (i, j) element of cost matrix C. d iou i,j is the IoU distance between tracklet i-th predicted bounding box and the j-th detection bounding box, representing the motion cost. d cos i,j is the cosine distance between the average tracklet appearance descriptor i and the new detection descriptor j. $\hat{d}_{i,j}^{cos}$ is theyre new appearance cost. θ_{iou} is a proximity threshold, set to 0.5, used to reject unlikely pairs of tracklets and detections. θ_{emb} is the appearance threshold, which is used to separate positive association of tracklet appearance states and detections embedding vectors from the negative's ones.
3. Kalman Filter: They suggested a new way to apply the Kalman filter, instead of using the state vector as in SORT which is defined as $x = [x_c, y_c, s, a, \dot{x_c}, \dot{y_c}, \dot{s}]^T$, They found found through experiments, that estimating the width and height of the bounding box directly, results in better performance. Hence, we choose to define the KF's state vector as $x = [x_c, y_c, w, h, \dot{x_c}, \dot{y_c}, \dot{w}, \dot{h}]^T$ and they also changed the Q and R as follows:

$$\boldsymbol{x}_k = [x_c(k), y_c(k), w(k), h(k), \dot{x_c}(k), \dot{y_c}(k), \dot{w}(k), \dot{h}(k)]^\top$$

$$\boldsymbol{z}_k = [z_{x_c}(k), z_{y_c}(k), z_w(k), z_h(k)]^\top$$

$$\boldsymbol{Q}_k = diag\big((\sigma_p \hat{w}_{k-1|k-1})^2, (\sigma_p \hat{h}_{k-1|k-1})^2, (\sigma_p \hat{w}_{k-1|k-1})^2, (\sigma_p \hat{h}_{k-1|k-1})^2, (\sigma_v \hat{w}_{k-1|k-1})^2, (\sigma_v \hat{h}_{k-1|k-1})^2, (\sigma_v \hat{w}_{k-1|k-1})^2, (\sigma_v \hat{h}_{k-1|k-1})^2\big)$$

$$\boldsymbol{R}_k = diag\big((\sigma_m \hat{w}_{k|k-1})^2, (\sigma_m \hat{h}_{k|k-1})^2, (\sigma_m \hat{w}_{k|k-1})^2, (\sigma_m \hat{h}_{k|k-1})^2\big)$$

which they later showed experimentally that those changes leads to higher HOTA.

2

A. Run the BoT-SORT tracker on the five given subsets from MOT17 . Go step by step to get an evaluation report and animation results. Attach the final performance report and animation to the report.

We have attached the animation and the performance report to the submission of this report.

B. Compare in high level between them focusing on CLEARMOT, HOTA, IDS , Identity and Detetcion perfomance

We will address each one in high level, but first we will define each criterion:

HOTA: Higher order Tracking Accuracy. Geometric mean of detection accuracy and association accuracy. Averaged across localization threshold. (the higher the better)

MOTA: Multi-Object Tracking Accuracy. This measure combines three error sources: false positive, missed targets and identity switch. (the higher the better)

IDF1: ID F1 Score. The ratio of correctly identified detections over the average number of ground truth and computed detections. (the higher the better)

DetA: Detection accuracy. Detection Jaccard index averaged over all matching detections and then average over localization thresholds

First we will review each video at high level:

02 – A static camera at eye level that is observing a very crowded urban open area

04 – A static camera that is set on a high ground (possibly security pole) observing a very crowded urban open area

09 – A static camera staged a bit bellow eye level observing the entrance to a store on a crowded street

10 – A moving camera at eye level walking pass a very crowded street

13 – A moving camera stated on top of a vehicle passing through a city street

The recived results is attached to the report in the Results Analysis.txt file.
At a high level we can see that over all the stats for the most part the BotSORT received very nice score (lowest is combined is 79.1 for the HOTA) on all the different metrics. we can see that the model receives the overall best scores on the 04 recording which makes sense, since this scenario is static and have the best viewpoint from all the recordings.
And we can also see that model receives the lowest overall scores for the 02 recording which is a bit of surprise for us since in the beginning we thought that a moving camera such as the one in recording 10 will be the hardest since we need to account for the pedestrian changes between frames and also the camera changes between frames, but since the recording in 02 is A crowded area full of people hence full of occlusions, we can understand why the algorithm had the hardest time with it
On top of that we can see that the BotSORT excels on the MOTA (which is defined above) score, since it has a combined score of 91.16.
And the algorithm struggles the most with the HOTA score ,which as we can see in the attached Results Analysis.txt file, except for the 04 recording all the other recordings scored less than 80.

3. Analysis of results

A. In some of the given subsets, the camera is in movement and not static. What might be the challenge in tracking in this case, and how do you suggest handling it?

The challenge with a moving camera is the fact that Tracking-by-detection trackers rely heavily on the overlap between the predicted tracklets bounding boxes and the detected ones. In a dynamic camera situation, the bounding box location in the image plane can shift dramatically, which might result in increasing ID switches or false negatives. We suggest to handle it by image registration between two adjacent frames which is a good approximation to the projection of the rigid motion of the camera onto the image plane.

B. ID switch

In the images bellow (figures 25-28, marked by a bold black rectangle) we can see 2 examples of good crossing objects that keep the same ID. We think that tracking is successful because that in both this cases the crossing is between not similar pedestrians (so their encoding is different) and also in both examples the objects were previously seen in the frame so the Kalman filter had enough time to learn their motion profile.
the 3 core function I the mechanism of the tracker that can help the tracker in this complicated cases are:

1. The Kalman filter which helps predict each pedestrians motion even when they are occluded (like what happens when 2 pedestrians cross each other)
2. Deep Appearance Extractor, with the usage of the embedding of each tracked pedestrian appearance the Bot-SORT mange's to avoid associating of two different pedestrians id due to crossing
3. Second Association based IoU, we know that in the case of a crossing pedestrian we will get that one pedestrian will occluded the other. Hence when applying the second association based IoU of all the low scores detections with all the unmatched IDs they managed to associate even the occluded pedestrian with their ID and hence not lose track of them and not create a new ID for the occluded objects which will cause a reset of their Kalman filter and might cause an ID switch.

We can see an example of bad crossing object that do not keeps the same ID in figures 29-30 marked by a bold rectangle, we can see that the women with ID 8 has her ID change after crossing a group of people. We believe that the reason for that is since she walked as part of a group moving together at roughly the same speed and direction and sometimes crossing one another inside the group made it hard for the tracker to track their IDs on top of that since the group is crowded and the camera filed of view is at eye level it makes it hard for the detector to give good detections.

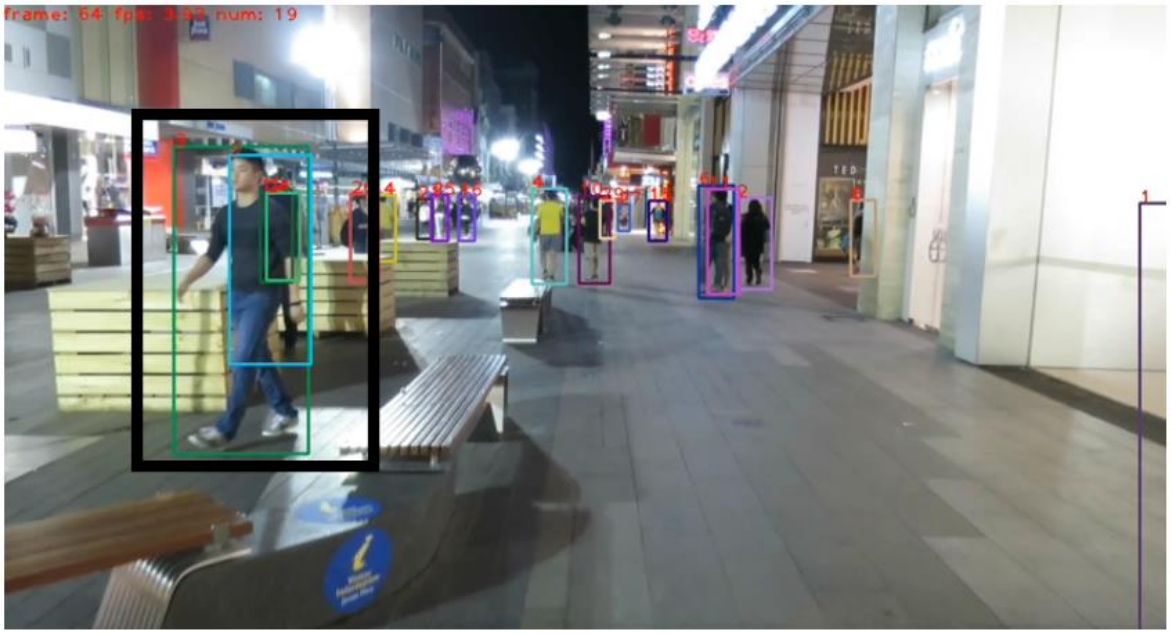

Figure 25 first example of good ID tracking part1 MOT 10 (ID 3 and 9)

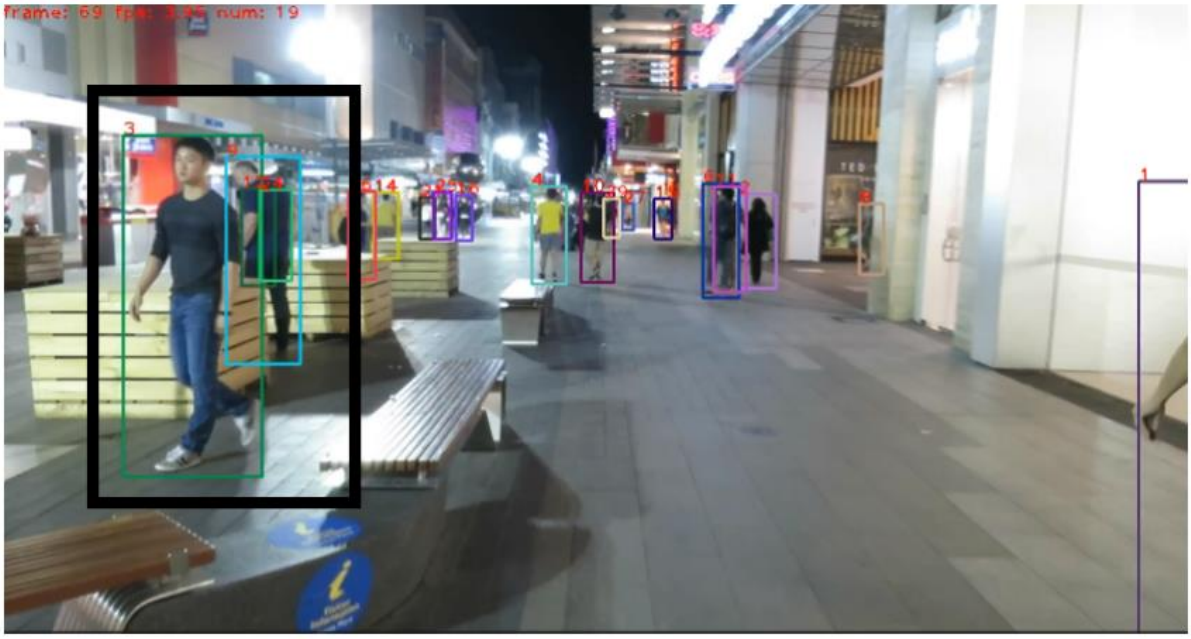

Figure 26 first example of good ID tracking part 2 MOT 10 (ID 3 and 9)

Figure 27 second example of good tracking part 1 MOT 09 (ID 14 and 18)

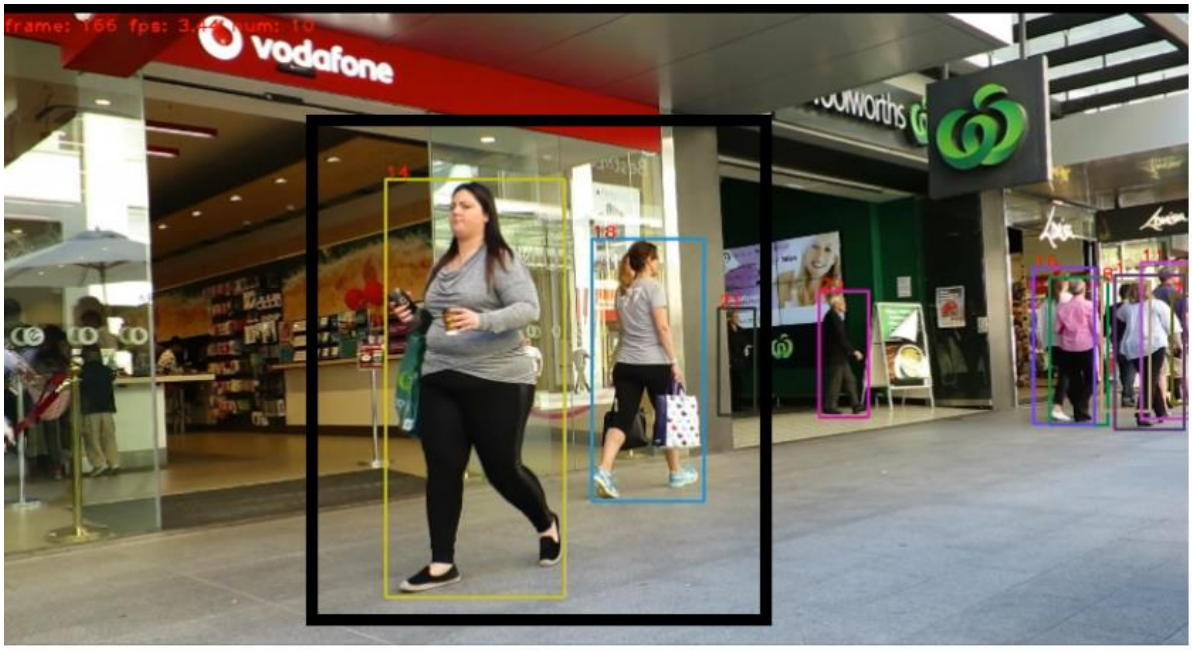

Figure 28 second example of good tracking part 2 MOT 09 (ID 14 and 18)

Figure 29 example of ID switch part1 MOT 09 (ID 8)

Figure 30 example of ID switch part2 MOT 09 (ID 8 that is now ID 11 hence the ID swithch)

D. Tracking on occlusion/truncated object

On figures 31 – 34 (the relevant part is marked by a bold black rectangle) we can see examples of good cases of tracking during occlusion, We think that it is successful due to the usage of the Kalman filter that managed to track them , along with the usage of the Second Association based IoU (which was explained previously), by applying the, both the tracker managed to track the objects even

during occlusion. But the key component for this 2 mechanism to be able to track the object even with occlusion is the fact that in bot cases we had already a good track of the object for a few frames before it was occluded, if it weren't for that than the object detection would probably below the detection threshold and it would not be tracked at all.

On figures 35 -38 (marked by a black bold rectangle) we have examples of 2 bad cases of tracking during occlusion. For the second occluded tracker (figures 37 38) we think that the failure reason was that the object entered the scene occluded hence the detector gave it a low score and no tracker was initialized for it hence it was not tracked (as we predicted will happen in the previous section). For the bad tracking on figures 35-36 we think that the reason that the tracker did not tracked the women with ID 14 was because she was fully occluded in the next frames hence the detector did not detected her at all so even with applying the Second Association based IoU the tracker weren't able to track her.

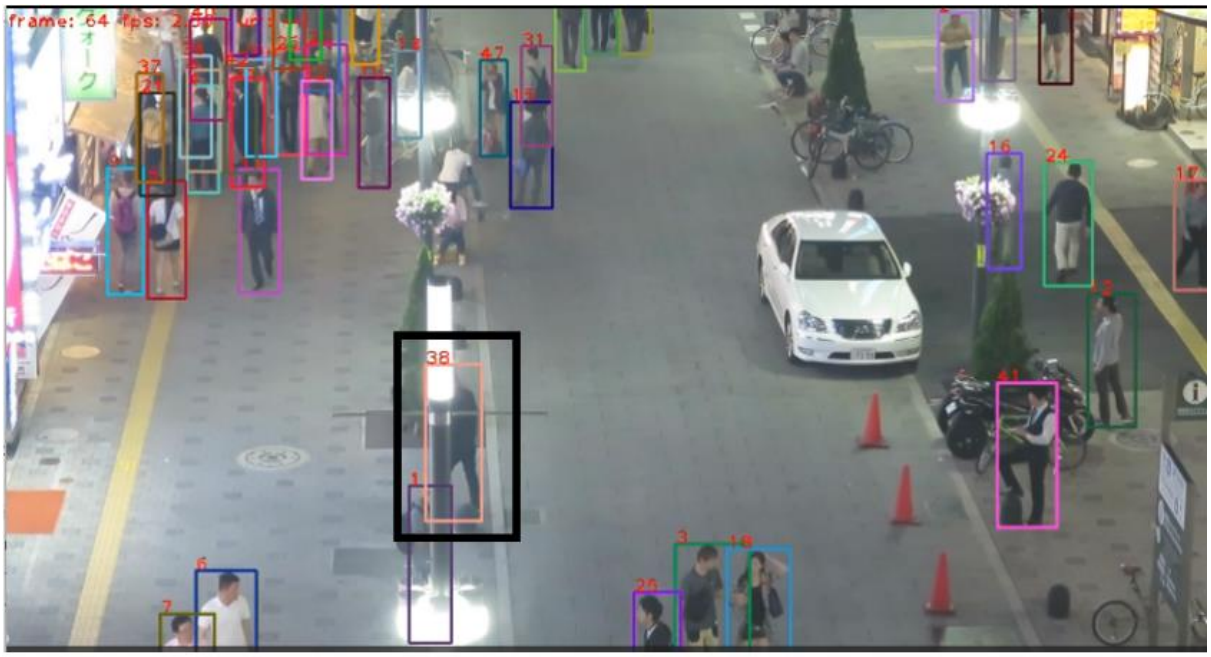

Figure 31 first example of tracking during occlusion part 1 MOT 04 (ID 38)

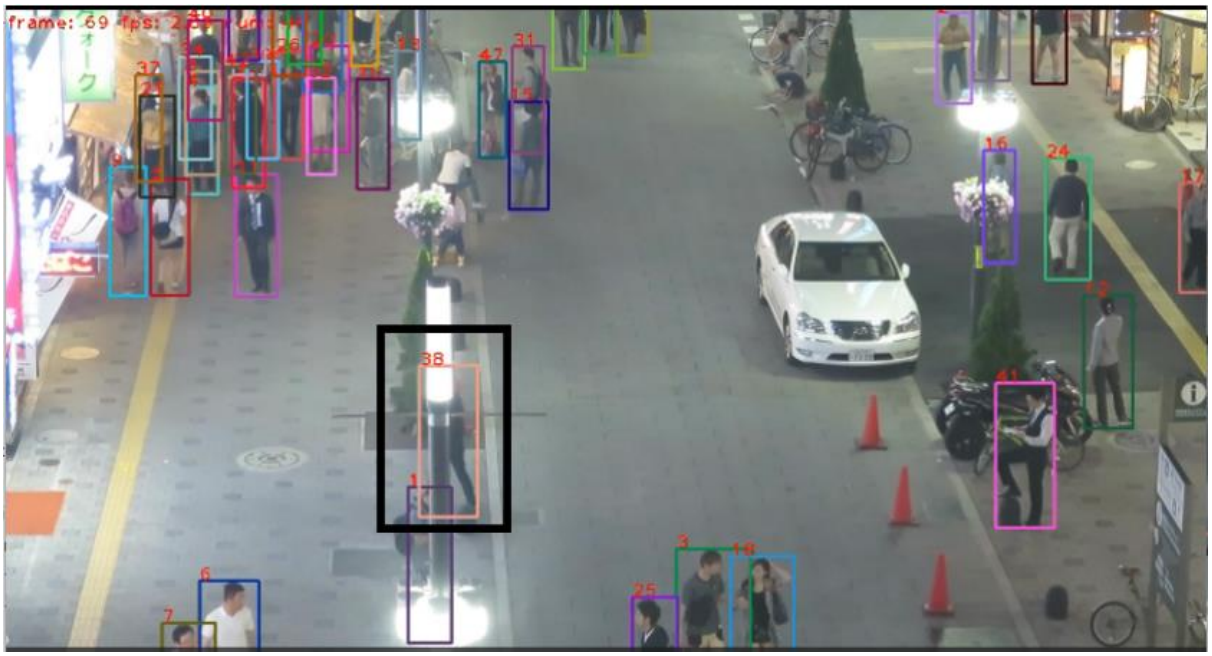

Figure 32 first example of tracking during occlusion part 2 MOT 04 (ID 38)

Figure 33 second example of tracking during occlusion part 1 MOT 04 (ID 16)

Figure 34 second example of tracking during occlusion part 2 MOT 04 (ID 16)

Figure 35 first example of bad tracking during occlusion MOT 09 (ID 14) part 1

Figure 36 first example of bad tracking during occlusion MOT 09 (ID 14) part 2

Figure 37 second example of bad tracking during occlusion MOT 13 (ID 36 not tracked since its occluded) part 1

Figure 38 second example of bad tracking during occlusion MOT 13 (ID 36) part 2

E. Detection performance

On figures 39 – 42 (marked by a clack bold rectangle) we can see three bad examples of detection that affect on tracking. We will explain for each how does it affect the tracking

1. For the first example: we can see in figures 39-40 that due to bad detection (i.e. false negative) the tracker did not received a detection for object 36 hence it was not tracked in the following frame which harmed the tracker ability to track the object.
2. For the second example we can see (on figure 41) that the detector did not detected (i.e false negative) the person sitting in the frames (marked by a bold black rectangle) and since that there was no detection for him during the entire video he was never tracked hence he did not receive a track ID.
3. For the third example (shown on figure 42) we can see a classic case of false positive we can see that the tree in the image was mistakenly detected as a pedestrian and hence the tracker started to track it and gave a track ID even though it is not a pedestrian but it is a tree.

Figure 39 first bad examples of detection MOT 13 (ID 36) part 1

Figure 40 first bad examples of detection MOT 13 (ID 36) part 2

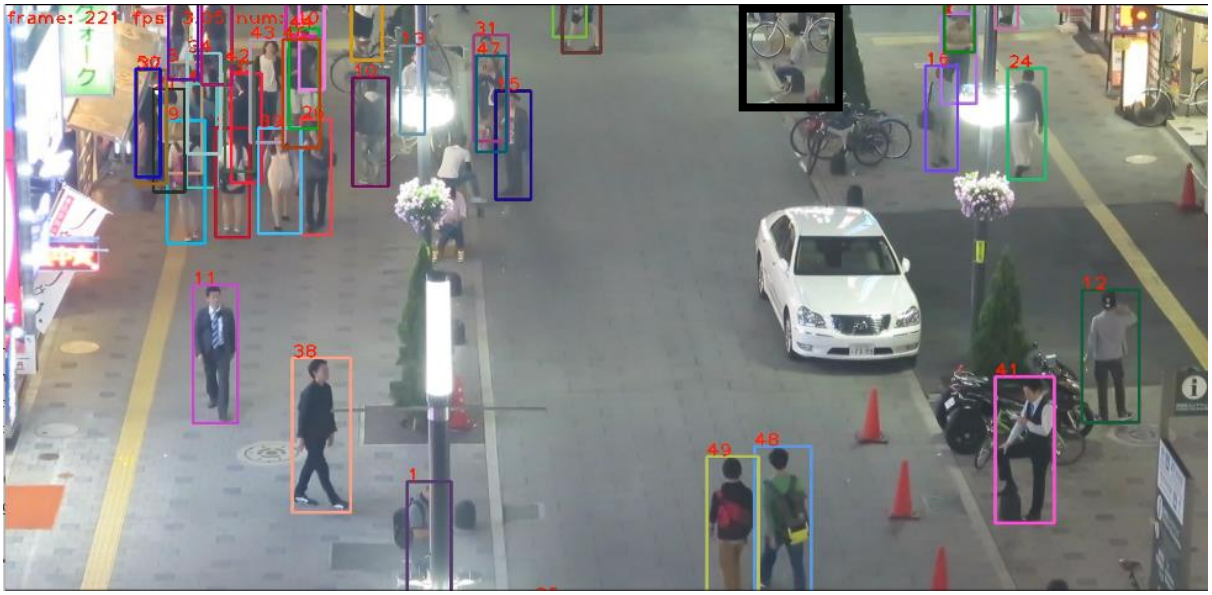

Figure 41 second bad examples of detection MOT 04 (no ID since no detection)

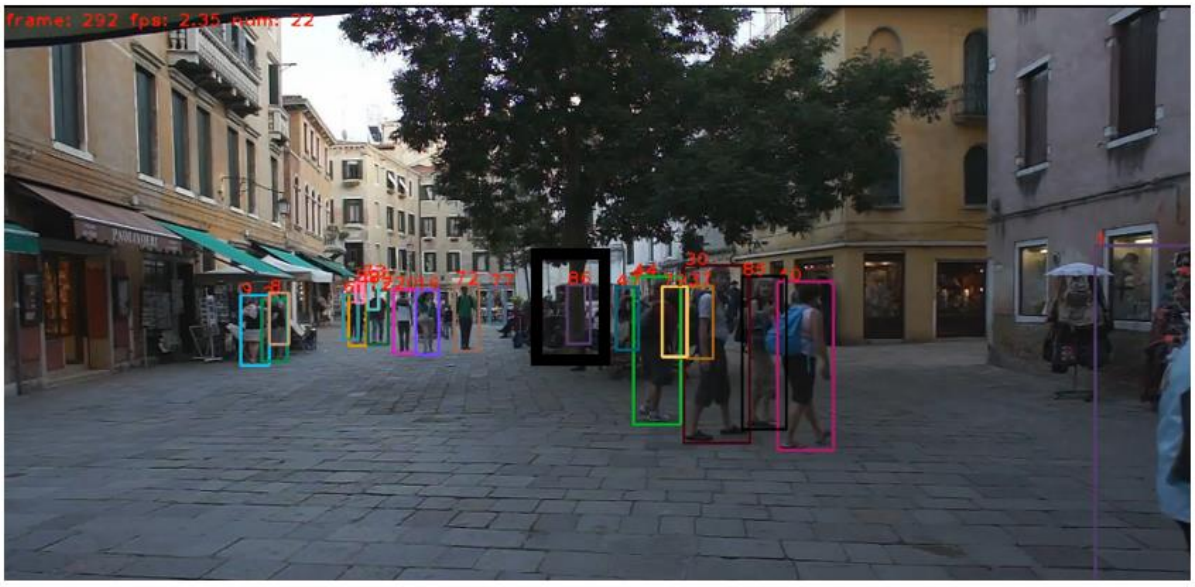

Figure 42 third bad examples of detection MOT 02 (ID 86)

Summary

In this project we focused on two fundamental aspects of computer vision: 3D object detection and multi-object tracking. The primary goal was to analyze and evaluate existing techniques and propose improvements to enhance the performance of these computer vision systems.

In the first part, we extensively studied the PointPillars technique, a deep learning-based approach for 3D object detection. Using an implementation trained on the KITTI dataset, we investigated how the Feature Encoder converts 3D point clouds into sparse pseudo-images. The analysis of detection results in different scenarios allowed us to identify key issues affecting the system's performance. We explored factors influencing confidence values, reasons for classification failures, and challenges related to false negatives and false positives.

Next, we explored the Frustum Convnet,Frustum PointNet and Frustum-PointPillars architecture that leverage both point clouds and RGB images for 3D object detection. We highlighted the advantages of this hybrid approach over the traditional PointPillar technique, showcasing its potential to push the boundaries of 3D object detection research.

In the multi-object tracking aspect, we dove into the theoretical background of popular trackers, including SORT, DeepSort, ByteTrack, and BoT-SORT. This allowed us to understand the core mechanisms behind these trackers. We emphasized the benefits of incorporating RE-ID into DeepSort, which significantly improved tracking performance. Moreover, we found for each newer method the good additions being made compared to the previous ones.

To assess the performance of the BoT-SORT tracker, we conducted evaluations on the MOT17 dataset, using various performance metrics. We analyzed challenging scenarios, such as tracking in dynamic camera settings, dealing with ID switches, occlusions, and the impact of detection performance on overall tracking accuracy. The analysis provided valuable insights into the strengths and limitations of BoT-SORT, and we presented exemplary cases of successful and failed tracking, along with theoretical solutions where applicable.

Overall, this project significantly contributed to the advancement of 3D object detection and multi-object tracking research. The comprehensive analyses, proposed enhancements, and evaluation results offered valuable knowledge to the computer vision community. Our findings could serve as a foundation for further research and development in these critical areas of computer vision, ultimately leading to more accurate and robust object detection and tracking systems in real-world applications.