Project 3: 3D Object Detection from Lidar Point Clouds

Problem 1: Building a 2 Stage 3D Object Detector

Nicolas Hoischen

Student ID: 16-819-880

Dario Bolli

Student ID: 16-832-685

FS 2021

227-0560-00L: Deep Learning for Autonomous Driving

Eidgenössische Technische Hochschule Zürich

Due Date: 02-07-2021

1 Recall and IoU

The recall is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

and the precision as:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall is an important metric to measure how good all the positives can be found. The average recall obtained on the validation set is 0.801, which means that 80.1% of all positives are identified correctly. In Autonomous driving it is important to make sure that all cars are detected, to avoid collision. Hence, the recall assesses the ability of the model to find all relevant relevant objects (cars, pedestrian etc...). On the other hand, precision quantifies how accurate the predictions are, in other words the percentage of predictions that were identified as correct and which were indeed correct.
The average precision however, is the area under the precision-recall curve. Thus it represents a trade-off between the False Positive and the False Negative. If we want to be able to detect almost all the positives, we have to accept that some of the boxes in our network detected as positives might be in fact negative (False Positive).
The aim of the first stage network, is to learn to detect the maximum of objects (cars in this case), that is to minimise the False Negatives.
Indeed, the refinement of the proposals is implemented in stage 2, so the recall metric assesses how well the first stage detects any possibility of having a car at a given position.
Thus for safe driving, the recall is a good metrics to evaluate the first stage, as we then refine the propositions in stage 2 using a sampling scheme.

To implement the intersection over union(see code in appendix [3]), we first get the corner coordinates of the boxes, and then use the intersection and area functions from shapely, to compute the intersection area of the boxes in Bird Eye View (see appendix [2]). Then, we get the length of the boxe's intersection on the vertical axis (y-axis) with min of maxs and max of mins. Multiplying this with the intersection area found previously gives us the intersection volume of the boxes.

2 ROI pooling

For each box and the respective corners obtained from *label2corners* (see [1]), three vectors (u, v, w) and their norm are computed following:

$$\begin{aligned} \mathbf{u_i} &= c_{i,5} - c_{i,6} \\ \mathbf{v_i} &= c_{i,7} - c_{i,6} \\ \mathbf{w_i} &= c_{i,2} - c_{i,6} \end{aligned} \tag{3}$$

where $c_{i,j}$ represents corner j of bounding box i. Vectors u, v, w are attached to corner 6 of the respective bounding box. Then, the vector $\mathbf{P_{i,k}}$ pointing from box i to all points k is calculated, using the centre of the bounding box i as:

$$\mathbf{P_{i,k}} = xyz(k) - \frac{c_{6,j} + c_{0,j}}{2} \tag{4}$$

To determine if each point k belongs to a certain box all the following conditions have to be

satisfied:

$$\begin{aligned}
\mathbf{P_{i,k}} \cdot \frac{\mathbf{u_i}}{||\mathbf{u_i}||} &\leq \frac{||\mathbf{u_i}||}{2} \\
\mathbf{P_{i,k}} \cdot \frac{\mathbf{v_i}}{||\mathbf{v_i}||} &\leq \frac{||\mathbf{v_i}||}{2} \\
\mathbf{P_{i,k}} \cdot \frac{\mathbf{w_i}}{||\mathbf{w_i}||} &\leq \frac{||\mathbf{w_i}||}{2}
\end{aligned} \tag{5}$$

The function *indexInBox* [5] loops through all boxes and is vectorized over the points and the previous calculation, whereas the function *points_in_box* [4] checks the conditions in equation [5]. Finally the point indices satisfying these conditions are returned for each respective box. Sampling with replacement to 512 points per box is implemented in case less than 512 points satisfy the conditions described in [5]. Otherwise, if one box contains more than the required 512 points, sampling without replacement is performed to discard the extra points.

Visualization:

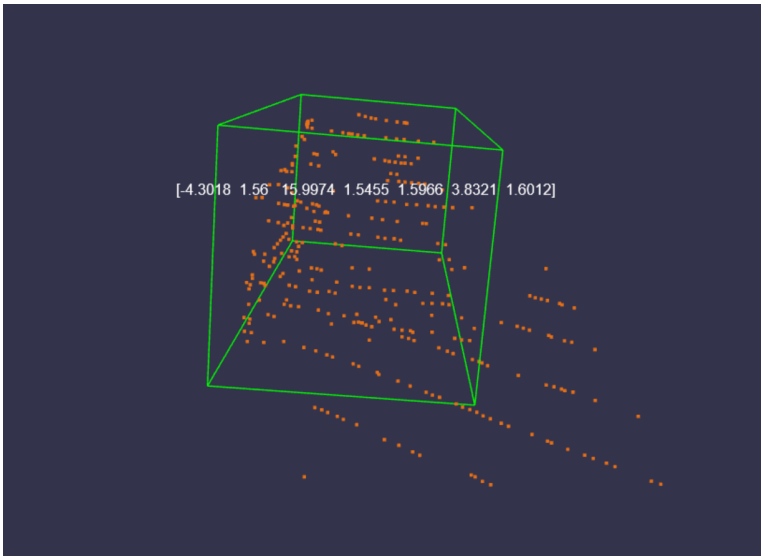

Figure 1: Frame 4 at epoch 34

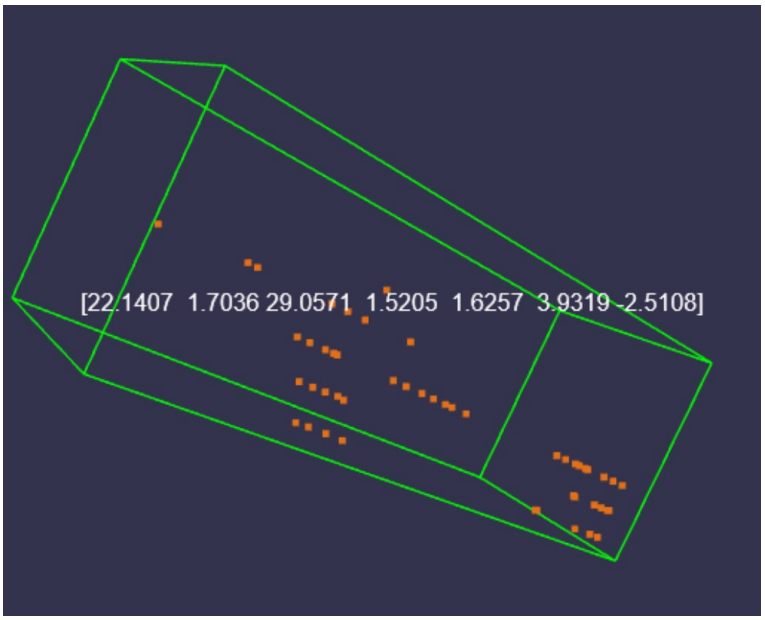

Figure 2: Frame 50 at epoch 34

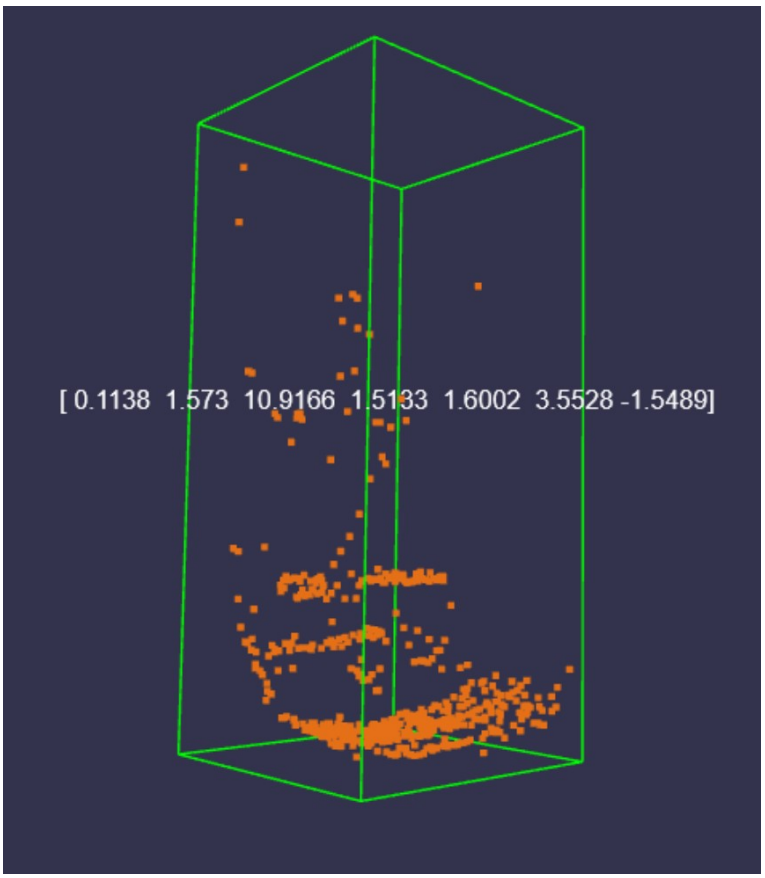

Figure 3: Frame 100 at epoch 34

As observed in the three examples of the ROI figures for scene 4, 50 and 100, the majority of the pooled points are within the predicted 3D bounding boxes. Indeed, the points were sampled for the **enlarged** bounding boxes (by $\delta = 1$m in all directions) which explains that some of the visualized points are actually outside the normal predicted box in figure [1].

3 Background and Foreground sampling

Why is such a sampling scheme required? What would happen if we (1) randomly sample from all proposals for each scene? (2) don't sample easy background proposal at all? (3) consider a sample to be foreground if it's above 0.5 IoU and background otherwise?
Why do we need to match the ground truth with its highest IoU proposal (4)?

(1) If we sample randomly from all the proposed boxes, there would be a bias towards the background samples, since there are not a lot of objects (cars) in a scene. So randomly sampling would skew the sample in favour of background samples. However, sampling solely foreground samples is not the best sampling scheme neither, as the background samples allow us to classify (train the network to differentiate background object from foreground object). On top of that, if we randomly sample, we might discard all the prediction boxes corresponding to one target (more details in (4)).

(2) The easy-background proposals sampling plays also a role for proper training of the model. Indeed, as we have a classification module, it is interesting to cluster the data properly, hence selecting background cases that are as distinct as possible from the foregrounds samples (easy-background). Conversely, the hard-background will allow the model to learn to make a distinction between two close samples (hard-background and foreground samples), increasing the model's discriminative ability. It is more difficult for the model to differentiate hard background samples from foreground samples than differentiating foreground samples from easy background samples. This is why it is important to represent the entire space of background sample, without preference for

those similar or dissimilar to foreground samples.

(3) The gap between foreground and background sampling is introduced because is preferable not to pride the with conflicting information e.g a foreground sample which would have an IoU of 0.5 and a background sample with an IoU of 0.4999. It will be harder for the model to make a distinction between the two cases.

(4) Our model should learn to predict bounding boxes where the targets are, and optimise this boxes to match the target boxes. Hence, if one target is left without prediction, the model would not be able to learn a box for this ground truth. In other words, it is important for the loss to have a prediction associated to a target to properly compute the regression loss. It makes sense to use the highest IoU because it is the closest prediction to the ground truth.

Visualization: The visualization in figure 4 shows the 64 sampled boxes with the corresponding refined pooled points (of shape: [64,512,3]). The predicted boxes coincide with the pooled points (considering that the pooled points where taking for the enlarged boxes).

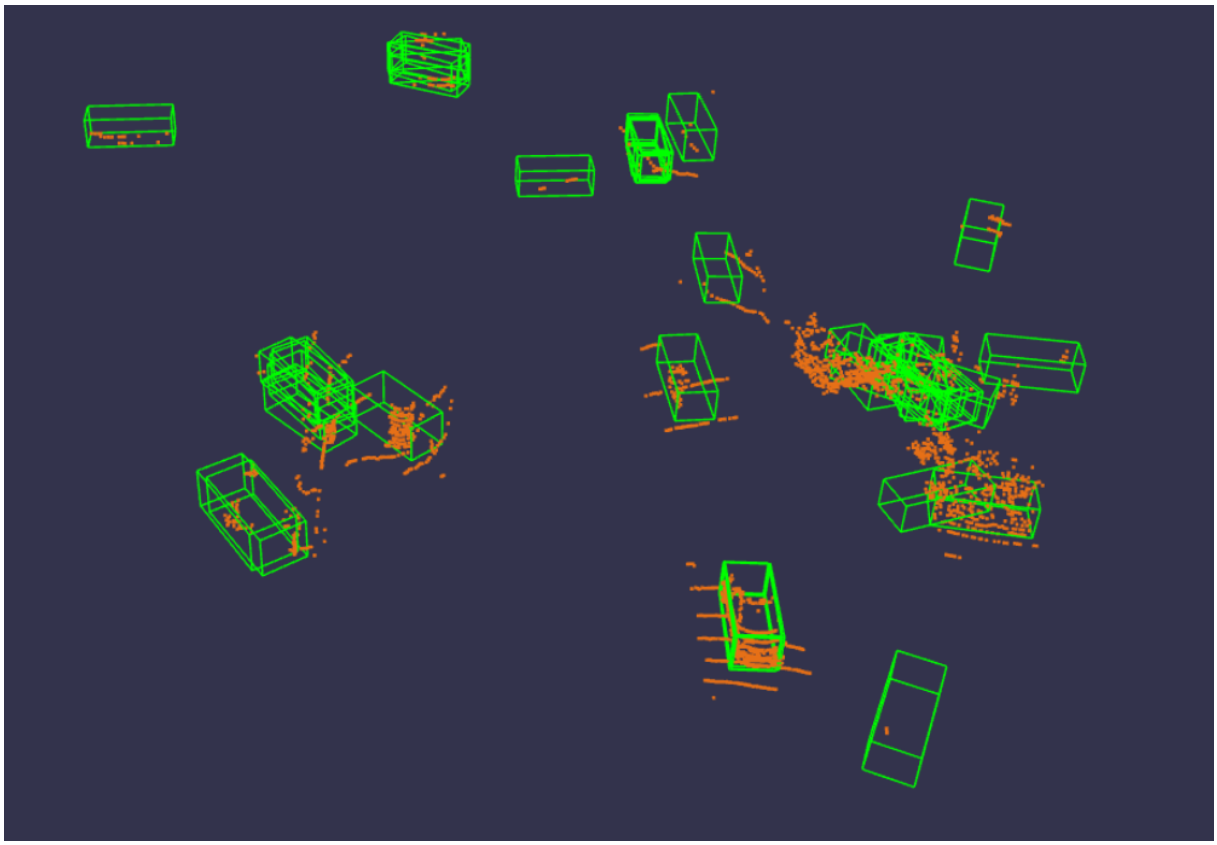

Figure 4: scene 5 from the training set with all 64 proposals

4 Non-maximum suppression

Must the IoU be calculated on BEV or can it also be calculated on 3D? What advantages does the 2D BEV IoU have over 3D (or vice versa) for NMS?

The 3D IoU could also be used to achieve this task and would give precise results as well.

However, as the Bird Eye View (BEV) is a projection of the points in the x-z ground plane, the 2D BEV provides enough information for the NMS to discard non relevant boxes. The advantage of computing the IoU on the BEV is that only the alignment in the x-z plane (ground plane) is taken into account and is well suited for autonomous driving in order to avoid collisions.

Indeed, as the points are projected on the ground, the surface of the car which is projected on the ground is the biggest possible. So it is the most restrictive when it comes to avoid car crashes.

Of course, the 3D IoU is the most accurate when comparing two boxes in three dimensions, and also allows good detection to avoid cars to collide. However, as cars are driving on the ground, the height dimension of a box is less important than it's position and rotation in the x-z plane. Moreover, a 2D IoU is also easier to compute and generally computationally less intensive. An edge case example where it would be more advantageous to compute a 2D IoU:

- One box perfectly aligned with the ground truth car, but for which the height is way too large
- One box not very well aligned with the ground truth car, but with perfectly matching height

So if we were to take 3D-IoU, the NMS algorithm would probably discard the well aligned box in the x-z plane, because the 3D IoU would not be very large compared to the other IoU of boxes with matching height but not as well aligned.
However, we do not want to discard this well aligned box, even if the height is too large.
Indeed, it is more important for the car to know the precise location and rotation of the car, rather than its height. Drive safely!

5 Train the network

5.1 example scene visualisations

We observe a significant improvement in the scene visualisations from epoch 0 to epoch 34. Indeed, after 17 epochs, we achieve 100% recall and precision on scene 0 and 30. However, as our network does not have a 100% maximum Average Precision, we surely have some scene where we do not achieve similar performances.
This is the case in scene 10, even after 34 epochs, we still have some False negatives. We do not detect the 4 target boxes on the right of the Lidar point cloud.
They are on the side of the Lidar point and quite far from the car origin frame. There might be some Lidar points reflected on obstacles before reaching those targets, and there is also less Lidar points than for targets closer to the origin frame. Those target boxes might be harder to predict.

5.1.1 example scene 0 visualisations

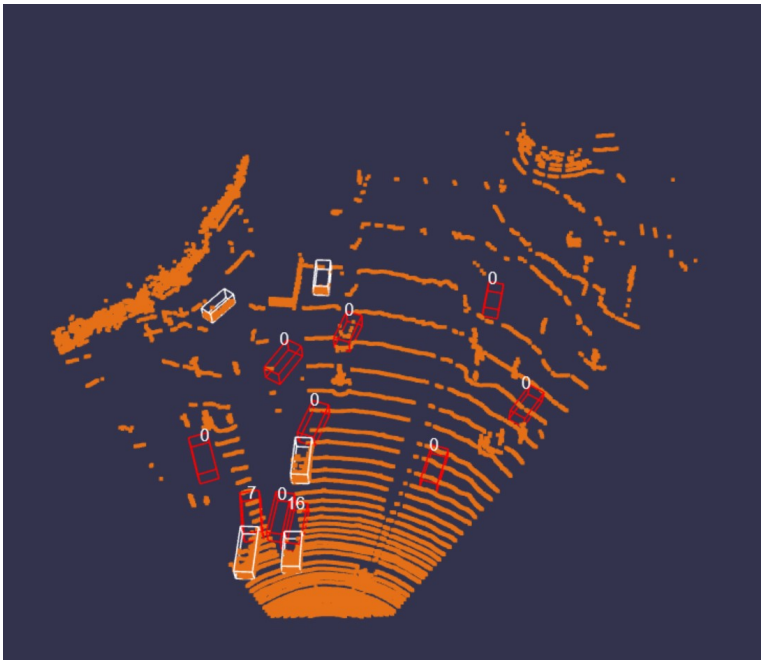

Figure 5: Scene 0 visualisation at epoch 0

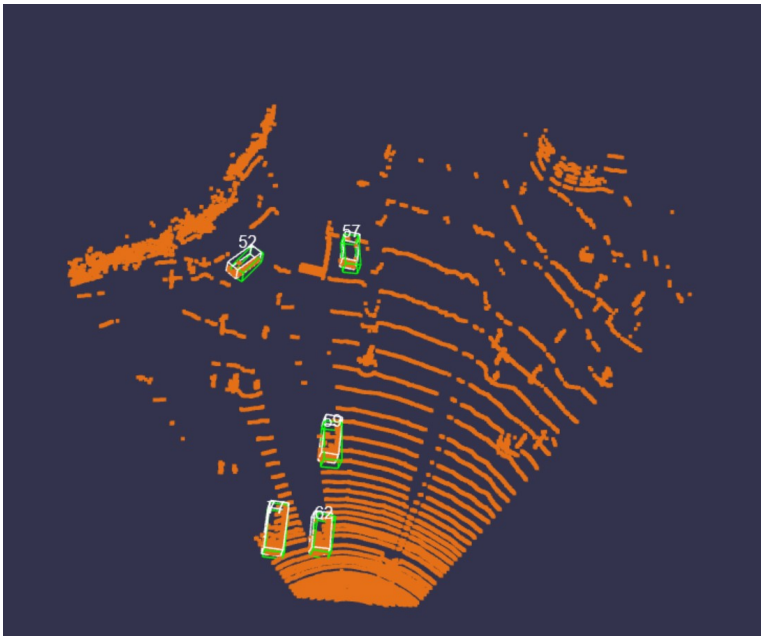

Figure 6: Scene 0 visualisation at epoch 17

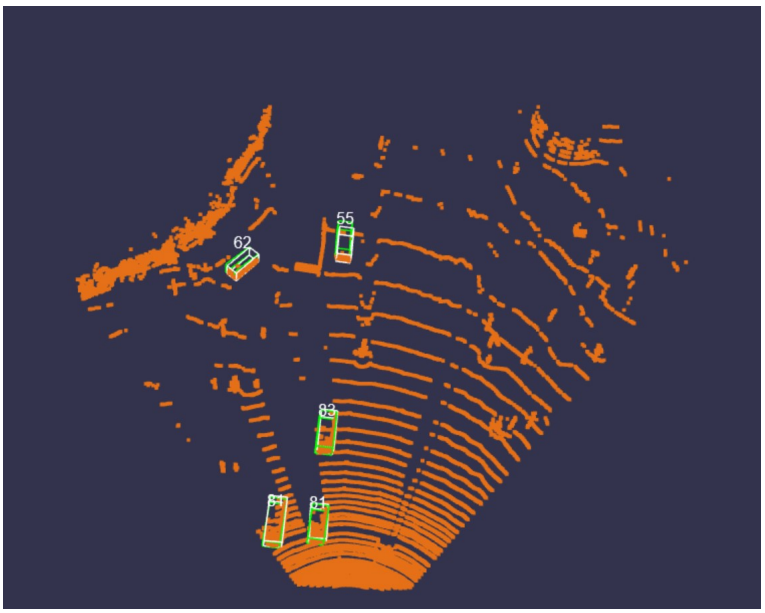

Figure 7: Scene 0 visualisation at epoch 34

5.1.2 example scene 10 visualisations

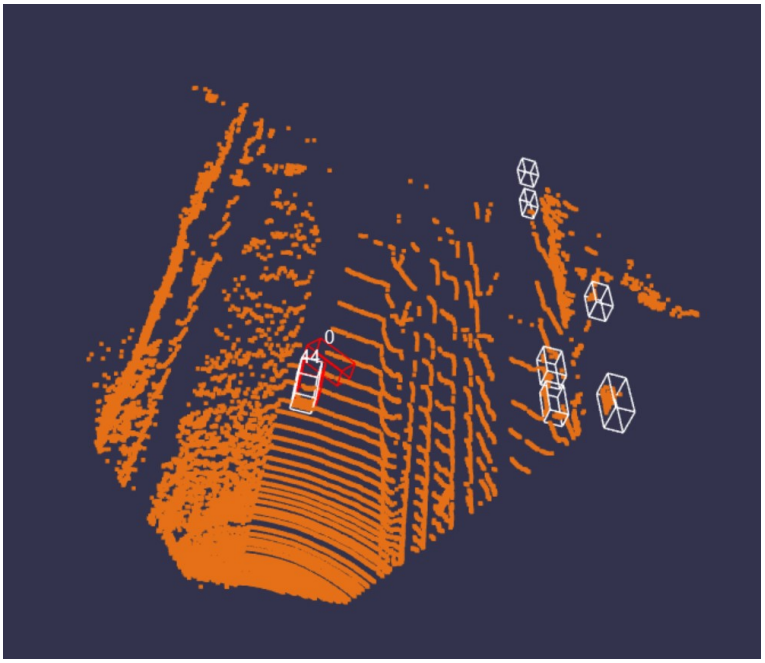

Figure 8: Scene 10 visualisation at epoch 0

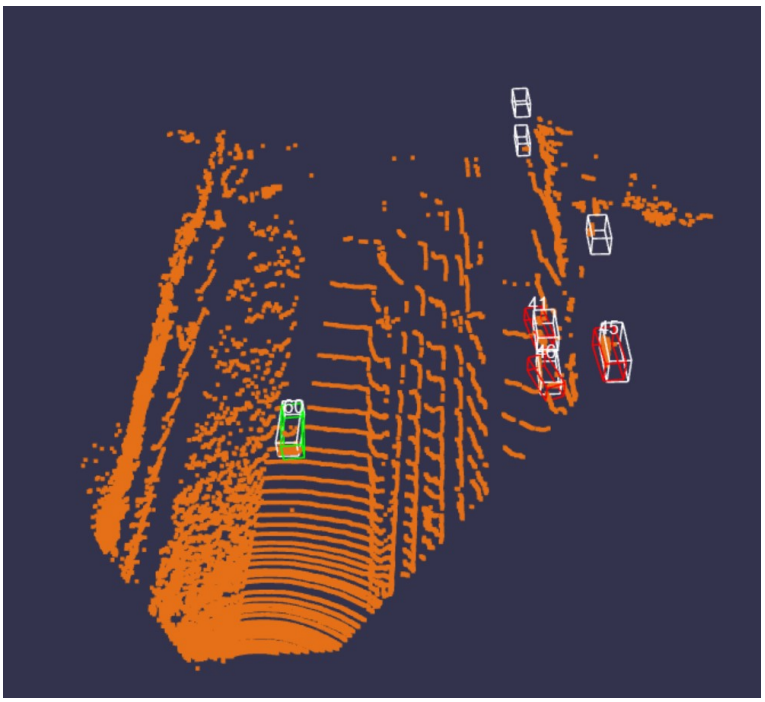

Figure 9: Scene 10 visualisation at epoch 17

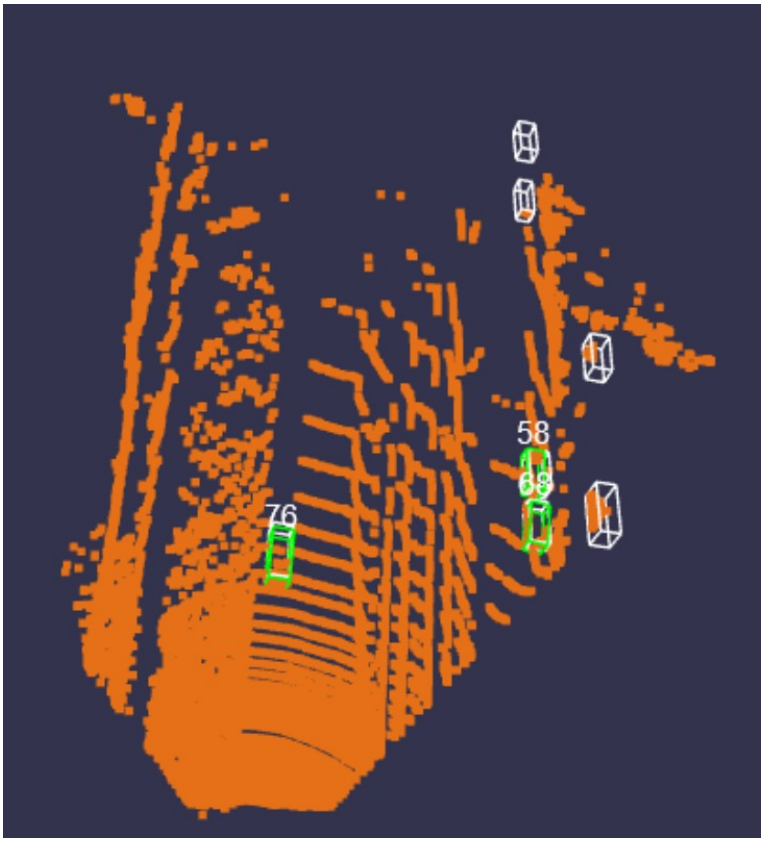

Figure 10: Scene 10 visualisation at epoch 34

5.1.3 example scene 30 visualisations

Figure 11: Scene 30 visualisation at epoch 0

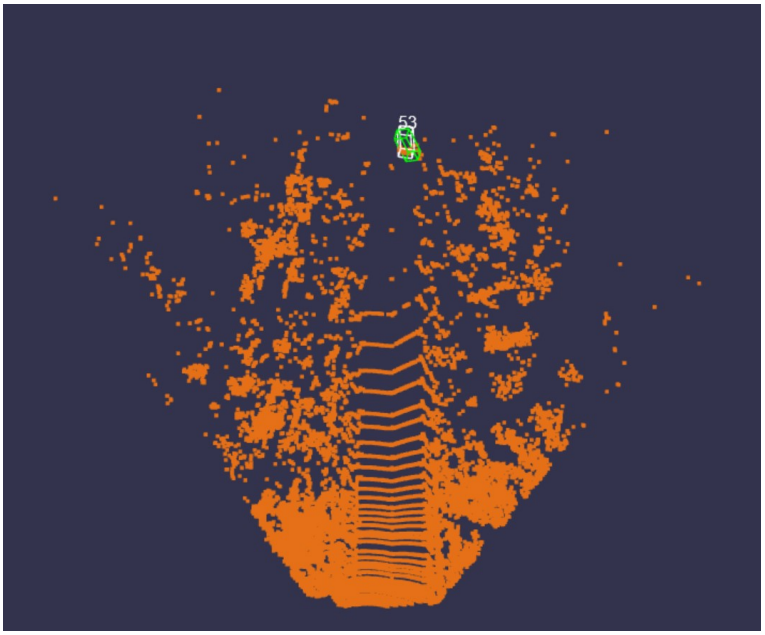

Figure 12: Scene 30 visualisation at epoch 17

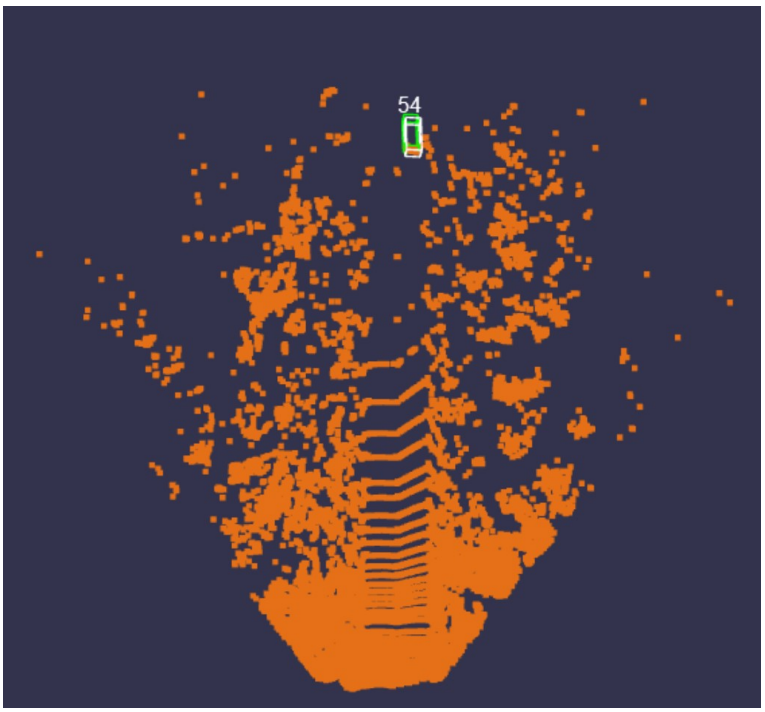

Figure 13: Scene 30 visualisation at epoch 34

5.1.4 Loss curves

We experienced some troubles with the aws instances, and hence had to train the model in multiple times to be able to run it on 35 epochs.

Figure 14: Train loss with respect to epochs

Figure 15: Validation loss with respect to epochs

We observe a decrease proportional to $\frac{1}{exp}$ in the loss on the training set and on the validation set as well. As we have similar datasets, and the same model, the same kind of performances are expected, and this is indeed what we observe.
The final losses after 35 epochs are the following:

Train Loss	0.3853
Validation Loss	0.334

Table 1: Final Losses at epoch 34

5.1.5 Precision curves

To assess the final performances of our model, the maximum Average Precision is a good metric, as it allows a balance between recall and precision. So it quantifies at the same time how accurate our predictions are, and how well we can predict all the objects.

Figure 16: Easy mAP curve

Figure 17: Medium mAP curve

Figure 18: Hard mAP curve

	mAP	Expected mAP	Deviation mAP - Expected mAP
Easy mAP	81.908	81.43	0.478
Medium mAP	73.389	72.35	1.039
Hard mAP	67.319	71.11	-3.791

Table 2: Final mAP at epoch 34

The scores for the model are a bit above the baseline for the easy mAP and medium mAP but the hard mAP performs a bit worse than the expected results. However, such small variations are expected, and these discrepancies are within the range for which the model can be assumed to be correctly implemented.

6 Appendix

Task 1.1 functions in task1.py

```
def label2corners(label):
    '''
    Task 1
    input
        label (N,7) 3D bounding box with (x,y,z,h,w,l,ry)
    output
        corners (N,8,3) corner coordinates in the rectified reference frame

        (8, 3) array of vertices for the 3D box in
        following order:
            1 -------- 0
           /|         /|
          2 -------- 3 .
          | |        | |
          . 5 -------- 4
          |/         |/
          6 -------- 7

    '''
    corners = np.zeros((label.shape[0],8,3))

    for i in range(label.shape[0]):
        # extract the dimensions of the box
        height = label[i,3]
        width = label[i,4]
        length = label[i,5]

        # Corners location 3D in camera0 frame
        corners_x = [length/2,-length/2, -length/2, length/2, length/2, -length/2, -length/2, length/2]
        corners_y = [-height, -height, -height, -height, 0, 0, 0, 0]
        corners_z = [width/2, width/2, -width/2, -width/2, width/2, width/2, -width/2, -width/2]

        #Rotation around y-axis
        Rot = y_rotation(label[i,6])
        box_dim = Rot@np.vstack([corners_x, corners_y, corners_z])
        # center the box around (x,y,z)
        box_dim[0,:] += label[i,0]
        box_dim[1,:] += label[i,1]
        box_dim[2,:] += label[i,2]
        # Dimensions of box_dim: 3 x 8 i.e. rows are (x,y,z) and columns are the corners
        corners[i,:,:] = np.transpose(box_dim)
    return corners
```

Listing 1: Implementation of the corner location computation

where the rotation around the Y-axis in the camera0 coordinates is calculated using the rotation matrix:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \text{with } \theta \text{ the label } ry \tag{6}$$

```
def get_intersection_area(corners_pred, corners_target):

    box_p = shapely.geometry.Polygon([tuple(corners_pred[0,(0,2)]), tuple(corners_pred[1,(0,2)]), tuple(corners_pred[2,(0,2)]), tuple(corners_pred[3,(0,2)])])
    box_t = shapely.geometry.Polygon([tuple(corners_target[0,(0,2)]), tuple(corners_target[1,(0,2)]), tuple(corners_target[2,(0,2)]), tuple(corners_target[3,(0,2)])])
    if box_p.intersects(box_t):
        inter_area = box_p.intersection(box_t).area
    else:
        inter_area = 0
    return inter_area
```

Listing 2: Implementation of the intersection area

```
def get_iou(pred, target):
    '''
    Task 1
    input
        pred (N,7) 3D bounding box corners
        target (M,7) 3D bounding box corners
    output
```

```
        iou (N,M) pairwise 3D intersection-over-union
    '''
    corners_pred = label2corners(pred)
    corners_target = label2corners(target)

    Vol_pred = get_volume(pred)
    Vol_target = get_volume(target)

    iou = np.zeros((pred.shape[0],target.shape[0]))
    for i in range(pred.shape[0]):
        for j in range(target.shape[0]):
            #Height of the intersection boxe
            max_pred = max(corners_pred[i,0,1], corners_pred[i,6,1])
            max_target = max(corners_target[j,0,1], corners_target[j,6,1])
            min_pred = min(corners_pred[i,0,1], corners_pred[i,6,1])
            min_target = min(corners_target[j,0,1], corners_target[j,6,1])
            y1 = max(min_pred, min_target)
            y2 = min(max_pred, max_target)

            interArea = get_intersection_area(corners_pred[i, 0:4,:], corners_target[j,
    0:4,:])
            interVolume = interArea*max(0, y2 - y1)
            # compute the volume of union
            unionVolume = Vol_pred[i] + Vol_target[j] - interVolume

            #intersection over union
            iou[i,j] = interVolume/unionVolume

    return iou
```

Listing 3: Implementation of the IoU

Task 1.2 functions in task2.py

```
def points_in_box(projection, norm_u, norm_v, norm_w):
    '''
    output
        flag (N,) bool vector: true if point is in bounding box
    '''
    flag = (2*projection[:,2] <= norm_w) & (2*projection[:,0] <= norm_u) & (2*projection
    [:,1] <= norm_v)
    return flag
```

Listing 4: Boolean array for all points given one box

```
def indexInBox(xyz, corners, max_points):
'''
    input
        xyz (N,3) points in rectified reference frame
        corners (N',3,8) corner coordinates
        max_points sampling points number
    output
        valid_indices (K',M) indices of points that are in each k' bounding box
        valid (K') index vector showing valid bounding boxes, i.e. with at least
                    one point within the box
    '''
    u =  corners[:,5,:] - corners[:,6,:]
    norm_u = np.linalg.norm(u,axis=1)
    u = u/norm_u[:,None]
    v = (corners[:,7,:] - corners[:,6,:])
    norm_v = np.linalg.norm(v,axis = 1)
    v = v/norm_v[:,None]
    w = (corners[:,2,:] - corners[:,6,:])
    norm_w = np.linalg.norm(w,axis=1)
    w = w/norm_w[:,None]
    valid = []
    valid_indices = np.zeros((corners.shape[0],max_points),dtype=int)
    for i in range(corners.shape[0]):
        directions = np.stack((u[i,:], v[i,:], w[i,:]), axis = 1)
        center2point = np.subtract(xyz, (corners[i,6,:]+corners[i,0,:])/2,dtype=np.float32)
        projection = np.absolute(np.matmul(center2point,directions,dtype=np.float32) )
        xyz_indic = np.flatnonzero(points_in_box(projection, norm_u[i], norm_v[i], norm_w[i
    ]))

        if 1 < len(xyz_indic) < max_points:
            idx = np.random.choice(xyz_indic, size=max_points, replace=True)
```

```
            valid_indices[i,:] = idx
            valid.append(i)
        elif len(xyz_indic) > max_points:
            idx = np.random.choice(xyz_indic, size=max_points, replace=False)
            valid_indices[i,:] = idx
            valid.append(i)
        else: # just one point in this box, discard
            continue

    return valid_indices[valid,:], valid
```

Listing 5: Returns points indexes which are in the corresponding boxes