Deep Learning for Autonomous Driving

3D Object Detection from Lidar Point Clouds

Project 3 Report - Problem 1

Marian Kannwischer (mkannwisc@student.ethz.ch)
Fabian Lindlbauer (flindlbauer@student.ethz.ch)

We hereby confirm that we are the sole authors of the written work here enclosed and that we have compiled it in our own words. We have committed none of the forms of plagiarism described in the ETH Zurich 'Citation etiquette' information sheet. We have documented all methods, data and processes truthfully and have not manipulated any data.

Contents

1 Problem 1: Building a 2-Stage 3D Object Detector

In this problem we implement functions to train a 2-stage object detector. Whereas stage-1, the region proposal network (RPN), is already given to us by the instructors, we perform necessary data preparation to feed the RPN results to stage-2 and implement metrics to evaluate performance. Specifically, the pooling of data points, i.e. an assignment of points to proposed bounding boxes, is carried out. Furthermore, each proposed bounding box is assigned a ground truth (GT) bounding box. Lastly, we implement the necessary loss functions for the otherwise given stage-2 and a non-maximum suppression (NMS) step to reduce the number of predictions in a scene.

1.1 Task 1: Computation of the average recall

Solving this task requires to solve three sub-problems: transforming the box descriptions into 3d coordinates of the 8 corners, calculating the intersection over union (IoU) in terms of volume and finally computing the recall.

The first sub-problem is similar to Problem 2.3 of the first project, hence the same approach could be reused. However we changed the implementation to speed up performance. The code is implemented in `task1.label2corners()`.

Next, we can calculate the IoU. The general approach to calculate the volume IoU is

$$\text{IoU} = \frac{V_{\text{intersec}}}{V_{\text{box}_1} + V_{\text{box}_2} - V_{\text{intersec}}}. \tag{1}$$

The boxes are only rotated around the y-axis (yaw) and not rolled or pitched. This allows to reduce the problem to two dimensions. We can calculate the intersection volume V_{intersec} by multiplying the intersecting area A_{intersec} with the overlap of the heights y_{overlap}:

$$V_{\text{intersec}} = A_{\text{intersec}} * y_{\text{overlap}}. \tag{2}$$

The *shapely* [2] library allows for an easy and fast computation of the intersection area of two rotated polygons. The code for calculating the IoU is implemented in `task1.calc_iou()`.

Lastly, the recall is defined by $\frac{TP}{TP+FN}$. A target is successfully predicted (true positive) if at least one prediction box has an IoU with the target that is greater than a certain threshold. The computation of the recall is implemented in `task1.compute_recall()`.

Running our implementation over the validation set gives an average recall of 0.81342.

Why is recall a good metric?

In the first stage, we are interested in proposing all possible regions where an object could be residing. While we can rule out false positives in the second stage by predicting a low confidence value, a missed object (false negative) cannot be recovered at a later stage. Hence, it is of paramount importance to cover all objects with stage-1 proposals and minimize the number of missed objects. This is achieved by maximizing the recall metric. Using precision ($= \frac{TP}{TP+FP}$) would focus on reducing the number of false positive predictions, which is not crucial in the first stage and completely ignores what we care about, the number of false negatives.

1.2 Task 2: ROI Pooling

To train the network, we want to feed it proposed bounding boxes along with LiDAR points in the region of interest (RoI). These RoIs are the regions closely surrounding the bounding boxes proposed by the RPN. The following approach is implemented to pool data points to proposed bounding boxes. The idea is to enlarge the bounding box by 1 m in all directions, i.e. increase length, width, and height of every bounding box by 2 m each and take only the points inside this enlarged box. By enlarging the bounding box as stated, the probability of feeding the network all relevant points for an object is increased.

Determining whether a point is inside the enlarged bounding box can again be transformed to a planar problem, since the proposal boxes are only rotated around the y-axis. Firstly, we check whether the y-coordinate of a point is inside the respective region of the bounding box (note that the y-axis is representative for the height of objects in our coordinate system). Then, we check if the point is inside the ground plane rectangle of the enlarged bounding box from a bird's eye view. This is done by projecting

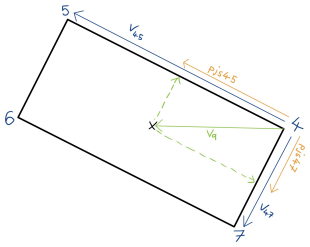


Figure 1: Projection.

the x and z coordinates of a point onto the vectors spanning the rectangle. Figure 1 illustrates the procedure.

After the points got projected onto the rectangle sides as shown above, the following conditions must hold in order for the point to lie withing the enlarged bounding box.

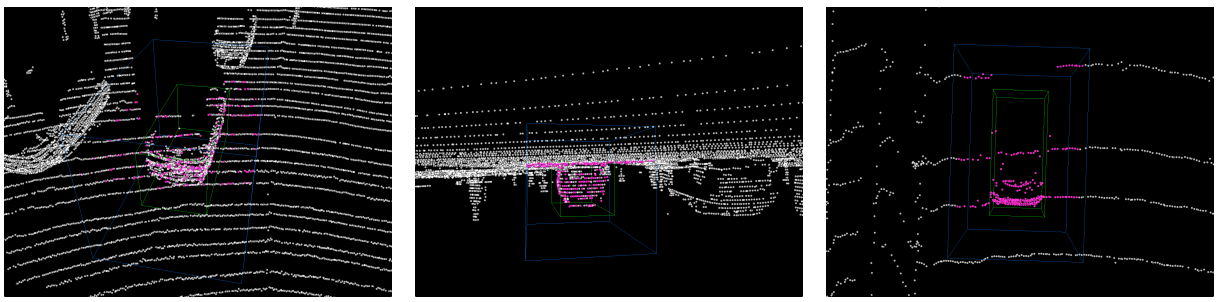
$$0 \leq \langle v_{45}, v_{4Q} \rangle \leq \langle v_{45}, v_{45} \rangle, \tag{3}$$

$$0 \leq \langle v_{47}, v_{4Q} \rangle \leq \langle v_{47}, v_{47} \rangle. \tag{4}$$

If the dot product between the query vector v_{4Q} and the projection vectors (v_{45} and v_{47}) yields a negative value, the point does not lie within the allowed range. Also, the maximum distance a point can lie away from point 4 in positive direction is the dot product of a rectangle side vector with itself ($\langle v_{45}, v_{45} \rangle$ and $\langle v_{47}, v_{47} \rangle$).

In addition, the network is fed a fixed number of points per proposal, 512 in our implementation. There are two cases that need to be respected in order to obtain the desired 512 points. If more than 512 points lie inside an enlarged bounding box, 512 points are randomly sampled from this pool of points. If fewer than 512 points could be gathered for a box, points are randomly repeated to arrive at the 512 points. Figure 2 shows the pooled RoIs for the first box in three different scenes.

A key requirement of this task was to implement it efficiently. We spent several days to optimize the code, hence we report the most rewarding changes. Firstly, instead of directly removing points and features from the arrays containing them, we switched to working with indices only and indexed the point and feature arrays with the computed index array just once in the very last step. Python actually allows to index the 16384×3 point array with a 100×512 array containing the 512 indices of the points for every one of the 100 boxes in a single operation. Secondly, there a three checks to find out whether a point is inside the enlarged bounding box. In the first place, we thought that it would make sense to conduct each subsequent check only with the points that passed the previous one. Surprisingly, it is actually faster to do each check with all the points and then combine three boolean masks. This is due to numpy dot products being extremely performant. The repeated indexing for points passing a test was the code bottleneck. Lastly, we could save even more time by disabling bound checks and negative indexing (wraparound) with the *cython* library [1].



(a) (b) (c)

Figure 2: **(a) Scene 0, box 0.** Not all points inside the extended bounding box are selected, since there are more than 512 available. **(b) Scene 1, box 0.** The pooling also works in terms of height. **(c) Scene 2, box 0**. Top view, the points are all exactly inside the extended bounding box. Here, all the points are selected, since there are less than 512 to begin with. **Colors:** Green: original bounding box. Blue: extended bounding box. White: All lidar points. Pink: pooled points.

1.3 Task 3: Generation of input-output pairs

In order to supervise the network predictions we assigned each bounding box proposal a GT bounding box. Specifically, the function `get_iou()` from task 1 was used to compute the IoU between every bounding box proposal and all GT bounding boxes in the scene. The GT bounding box that yielded the highest IoU value was assigned as GT bounding box for the proposal. (The reason for picking the target box with the highest IoU is explained in the questions section below.) At training time we sampled 64 bounding box proposals from the scene according to the sampling strategy proposed in the instructions sheet. It is important to stress that samples are categorized as foreground ($IoU \geq 0.55$) and background samples ($IoU < 0.45$), whereas background samples are further sub-categorized in easy ($IoU < 0.05$) and hard ($0.05 \leq IoU < 0.45$) background samples. This separation of IoU values is beneficial for the network to learn how to differentiate between found proposals (details can be inferred from the following questions section). Another vital aspect is to set a threshold between foreground and background samples. Setting $IoU = 0.5$ as threshold value between foreground and background predictions can easily lead to mispredictions as proposed boxes with IoU values slightly below the 0.5 mark could easily be actual background samples. In order to prevent such edge cases proposed boxes with $0.45 \leq IoU < 0.55$ are not considered during network training (details are further given below).

Why is such a sampling scheme required?

Each scene contains a varying number of cars, which is not known to the network a priori. As most neural networks expect to receive inputs of fixed size, a sampling scheme as the one we implemented guarantees fixed size input vectors. A reasonable number of inputs is 64 as a scene is unlikely to contain more than 64 cars, while still being a computational feasible number. The number of samples is a hyperparameter that can, of course, be tuned and changed if a network shows better performance with a different number of samples.

What would happen if we randomly sample from all proposals for each scene (1)?

The output of an RPN usually provides significantly more background samples than foreground samples as it tries to do its best finding all possible bounding boxes and detecting all foreground samples. Thus, by randomly sampling from the proposed bounding boxes the algorithm is prone to feeding the second stage far more background samples. This will ultimately lead to an imbalanced learning problem and bias the network's learning. The network could therefore learn to predict 'no car' (confidence = 0) also for cases in which a car or other objects in general are indeed present, as this is simply the most likely case from what it saw during training time. An alternative to the implemented sampling scheme could be a weighting of the update step according to whether the sample was a foreground or background one. A suitable loss function for this would be the focal loss [3].

What would happen if we don't sample easy background proposal at all (2)?

The network would not see easy background proposals during training and would thus be unable to learn information about this type of proposed bounding boxes by the RPN. As the network will see easy background samples during inference, its confidence could be inaccurate for this type of background samples and ultimately lead to false results.

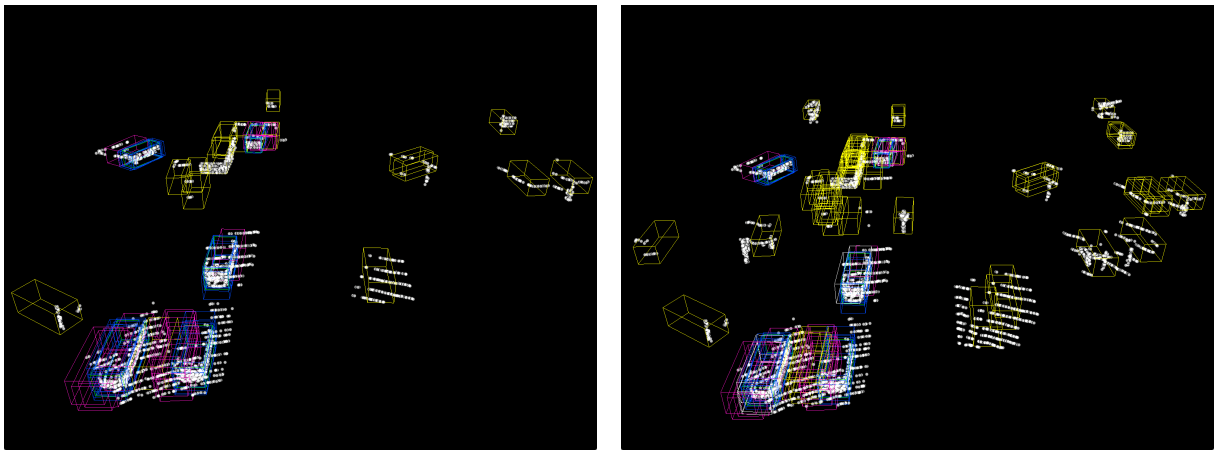
What would happen if we consider a sample to be foreground if it's above 0.5 IoU and background otherwise (3)?

There would be no margin between proposed background and foreground samples. For instance, two almost identical proposals could be counted as a foreground sample at one time and as a background one another time. By setting the threshold values of $IoU \geq 0.55$ for foreground samples and $IoU < 0.45$ for background samples we increase the probability of proposed foreground and background samples belonging indeed to the proposed category. I.e. edge cases which are hard to predict are ignored while training the network to avoid as much randomness concerning foreground or background predictions as possible.

Why do we need to match the ground truth with its highest IoU proposal?

In order to supervise the training, it is necessary to match proposed bounding boxes with GT boxes. More specifically, the GT values are required for bounding box regression and confidence score prediction using the implemented loss functions. By selecting the bounding box with the highest IoU it is possible to evaluate the RPNs best performing proposal. To answer the question why this makes sense it is possible to consider a different approach. For instance, one could select the median IoU value of proposed bounding boxes to evaluate the network performance. However, there is no point in training

the network such that half of the proposals are doing a good job, as a single good prediction is already enough to confidently find objects in a scene. There is no point in obtaining a great median IoU although the best IoU value is barely pushed throughout training as it is never considered as reference. By trying to have half of the proposals yielding a good IoU value training time might increase, too, due to further adjustments being necessary for some proposals despite having already found many reasonable bounding boxes. The same idea projects to any other value used apart from the highest IoU.



(a) Scene 0. `train=True`

(b) Scene 0. `train=False`

Figure 3: Blue: foreground samples, Orange: easy background samples, Pink: hard background samples, Green: targets (hard to see behind blue boxes). Dots: pooled points. As one can see, there are only 64 boxes whith a more balanced background sample ratio in training mode. For validation, the proposals were not sampled.

1.4 Task 4: Loss Functions

The loss functions are implemented in `task4.RegressionLoss` and `task4.ClassificationLoss` according to the instructions in the handout. Please, refer to this short file to check our implementation.

1.5 Task 5: Non-Maximum Suppression

In order to reduce the amount of predicted boxes from 64 to the most likely number, non-maximum suppression (NMS) was implemented in `task4.nms()`. We only want to keep the prediction with the highest confidence per overlapping set of predictions. Hence, we iteratively select the sample with highest confidence and remove all other predictions that overlap with the selected sample over a certain threshold (e.g. IoU > 0.1).

Must the IoU be calculated on BEV or can it also be calculated on 3D

3D IoU would likely work well, as 3D IoU behaves similar to 2D IoU only using ground plane areas instead of box volumes for the majority of cases. Especially as the heights and y-locations of bounding boxes varies little. However, the 2D IoU makes more sense in this case, for the reason given below.

What advantages does the 2D BEV IoU have over 3D (or vice versa) for NMS?

The first advantage is simple: There could be cases where predicted bounding boxes have a significant overlap in the x,z-plane, but do not overlap enough in y-direction for one box to be suppressed by the NMS-algorithm. This can happen due to different heights of objects. For instance, a truck and a car of little height, e.g. a sports car, will not have a substantial y-overlap. Even if their 2D overlap is optimal, NMS carried out in 3D space might mispredict, if it selects a proposal with worse 2D overlap, but better overlap in height.

Secondly, calculating 2D IoU is computationally a bit more efficient, hence speeding up training and inference.

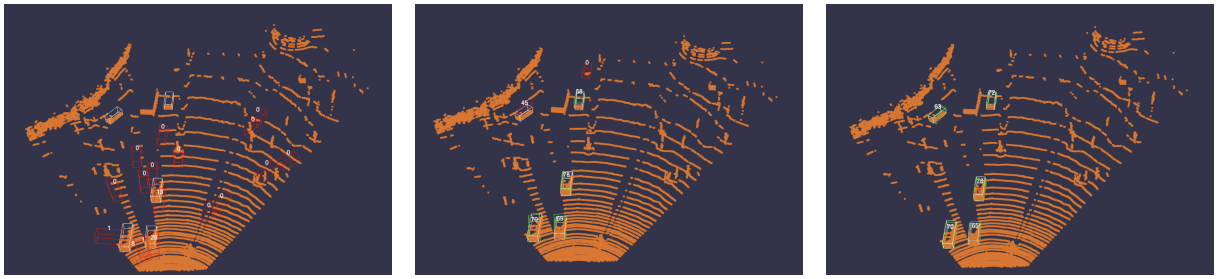
1.6 Task 6: Training of the network

The network training results shown in Figure 5 below illustrate that our network was able to reproduce the target results provided in the instructions sheet quite close. A slight contrast between our achieved results and the values given in the instruction sheet is given in the following.

metric	ours	solution
Easy mAP (e_map)	79.53	81.34
Moderate mAP (m_map)	69.57	72.35
Hard map (h_map)	64.10	71.11

The map values plotted in Figure 5 illustrate two noteworthy performance increases at epoches 12 and 24. Another significant increase can be observed at the last epoch trained. As it would have been interesting to see how the network performance develops over the next few epoches and if it can surpass the target map values we tried to train one more epoch, but it has been impossible over the last 3 days to get this epoch trained. We want to point out that it took us 7 days to get the whole run as presented in the figures below due to the AWS issues mentioned on piazza. The following 3 days we tried to train a few more epoches, but without any luck as AWS crashed our instances during the first epoch so that no results could be reported.

Additionally, the 3D visualizations in Figure 4 back the results shown in Figure 5. After the first epoch (epoch 0) the predicted bounding boxes are completely random as the network has barely learned anything about the objects yet. After epoch 18 three bounding boxes were predicted correctly, one false positive was found, and one foreground bounding box was not accurate enough. The results after epoch 34 reveal that the network was capable of finding all 5 bounding boxes and declared them correctly as foreground.



(a) Epoch 0. (b) Epoch 18. (c) Epoch 34.

Figure 4: 3D Visualization at wandb.ai for different epochs.

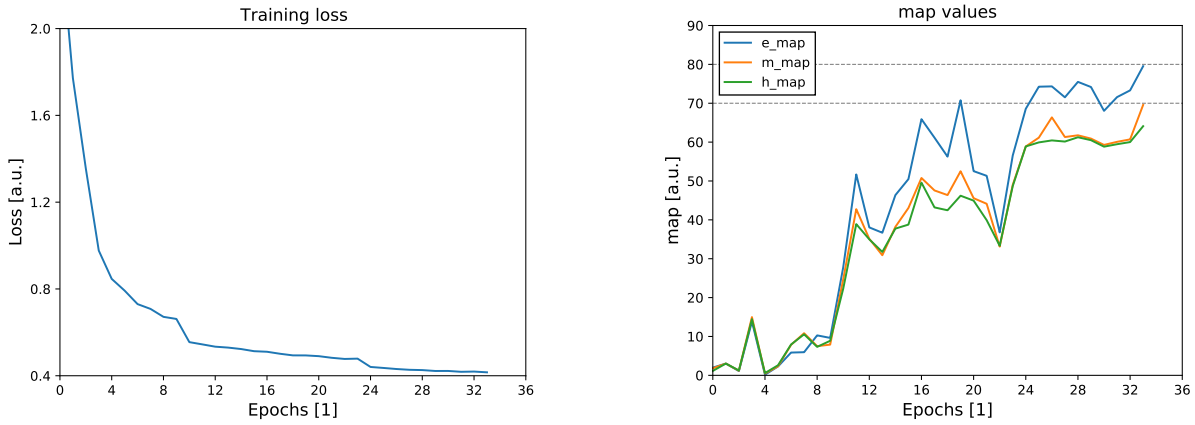


Figure 5: Train loss and map scores mapped against the number of epochs trained.

Appendix

Training experience

During the last week of the project we have not been able to train more than 2 epochs consecutively. Therefore, it was almost impossible to store zip files of our code as normal. The following figure illustrates the interruptions we had during training.

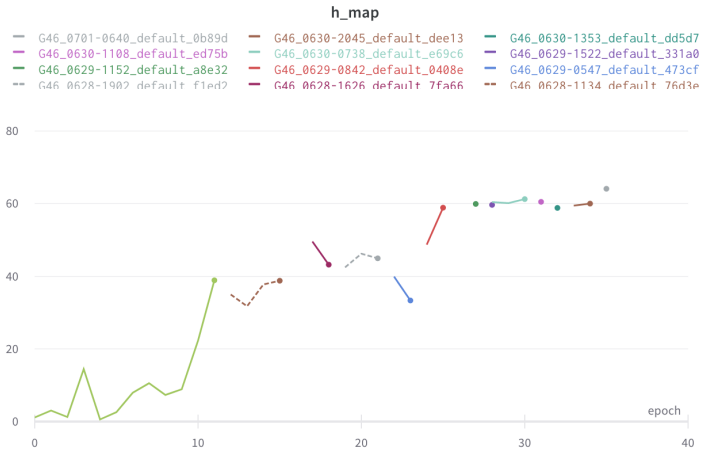


Figure 6: h_map mapped against the number of trained epochs. We want to show this plot to illustrate our mediocre experience during training. Each color represents a newly started epoch from a previously saved checkpoint. Some runs that were started as well are not shown as they have already crashed during the first epoch. This plot displays the same results as shown in Figure 5.

References

[1] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31 –39, April 2011.

[2] Sean Gillies et al. Shapely: manipulation and analysis of geometric objects, 2007–.

[3] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud. *CoRR*, abs/1812.04244, 2018.