Deep Learning for Autonomous Driving

3D Object Detection from Lidar Point Clouds

Project 3 Report - Problem 2

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 2: Taking Matters Into Your Own Hands

In order to improve our model, we identified four shortcomings. We decided to ameliorate them by employing data augmentation, canonical transformation, an extended extra feature vector and bin-based losses. Unfortunately, we were not able to conduct an ablation study due to severe instance crashing on AWS which made it impossible to train an extended network as we had already been working on a successful problem 1 run for a whole week.

1.1 Introduction

After successfully completing the implementation of problem 1 we focused on improving our developed baseline method. We studied several papers to find ideas on how we could improve the performance of the original approach. Some papers described very interesting improvements to the first region proposal stage, the so-called region proposal network (RPN). We found that generally two approaches are used to propose object bounding boxes. On the one hand, the voxel based approach splits the point cloud into different sub-spaces, a process called "voxelization". In general, the point cloud is clustered into small cuboids (values of length, width, and height are on the order of 0.5 m). A 3D convolution operation is then applied on these voxels to obtain semantic information from the point cloud. A voxel can be thought of as a pixel of an image in 2D convolution. The advantage of the voxel-based approach is its fast computation. The major drawback is the lost information of context due to the convolutions.

Another common approach we found in the literature is the point based one where a neural network is directly applied on the LiDAR point cloud without any previous sub-clustering. Such approaches tend to yield better performances as they consider spatial information more than voxelized approaches. However, training is computationally more expensive. In the paper [1] a mixture of the point-based and voxel-based approaches is proposed and excells at bounding box detection outperforming almost all previous works that only implemented one approach.

As the RPN outputs had already been given to us we focused on implementing an improved version of the second stage according to the techniques proposed in [2]. The authors of this paper managed to achieve a mean average precision (mAP) of `easy`: 88.88, `medium`: 78.63 and `hard`: 77.38 on the KITTI dataset. As these values outperform our network quite substantially we decided to adopt some of the proposed network parts, in particular we used the RCNN network on the official GitHub (link in [2]) repository and adapted it to our previous code.

The main idea of improvement proposed in the above mentioned paper is to alleviate the problem of sparse points far away from the LiDAR sensor. We decided to adopt 4 promising performance boosters to our network, which originate from [2] and are briefly described below.

(1) We can make the most of the given data by conservatively augmenting the training data. This is a very common step and known to yield performance improvements for most learning problems. Hence we implemented functions to randomly flip, scale and rotate the scene around the y-axis. This involved processing the point cloud as well as the prediction and target bounding boxes. The code herefore can be found in `data_augmentation.py`.

(2) By directly regressing the bounding boxes to the target boxes at various locations, we make the box refinement step unnecessarily complicated. The boxes are scattered around the scene with various orientations. In the baseline approach, we are aiming to refine the box to its absolute target values, which vary significantly. A much easier problem is to perform a so-called *canonical transformation*, basically translating each box with its points $p^{(p)}$ to the origin and rotating it such that it is facing forward. This yields the transformed points $\tilde{p}^{(p)}$. Now, we only need to regress a small offset between the prediction and target instead of absolute values. We implemented this step ourselves as well in `canonical_transformation.py`. However, the canonical transformation bares a downside: depth information form the absolute point coordinates is lost. Luckily, this issue can be addressed with the next network alteration.

(3) It is possible to recover the lost depth information by extending the $N \times 128$ feature vector proposed by the RPN. This is achieved by creating an extra feature vector of 4 to 6 entries - depending on the implementation details and strategy. In our approach we decided to use 4 features for this extra feature vector. Three features are made up by the canonical coordinates of each LiDAR point, which are computed as described under (2) above. As a fourth feature we compute the distance $d^{(p)}$ of a point from the LiDAR sensor to obtain depth information of every point. The depth is calculated by applying

the L2 norm on the point coordinates, i.e. by computing $d^{(p)} = \sqrt{(x^{(p)})^2 + (y^{(p)})^2 + (z^{(p)})^2}$, where $x^{(p)}$, $y^{(p)}$ and $z^{(p)}$ are the coordinates of a LiDAR point in the original coordinate system.

Additionally, the proposed paper includes the reflectance values of each point in the extra feature vector. Furthermore, a value for the segmentation mask 0, 1 is also proposed and refers to a point belonging to a foreground or background bounding box. Our 4 features are then fed into a neural network consisting of 3 fully connected (FC) layers with 128 neurons to extend it to a 128 feature vector. This vector is then concatenated with the feature vector obtained from the RPN. This 256 feature vector is then downsampled to a 128 feature vector by another FCNN consisting of two 256 and one 128 feature layers. The resulting vector is then fed through 3 stages of sampling, pooling, and a PointNet layer. Firstly, points are sampled using the farthest point algorithm (FPA). Secondly, surrounding points are pooled around these sampled points using a ball. The radius of the ball is increased with every stage. Thirdly, the pooled points are fed into point net. The resulting 512 feature vector is then fed into a regression and classification head.

(4) Lastly, [2] reports performance gains and faster training convergence by incorporating *bin-based regression losses*. Instead of regressing the ith box proposal's location and orientation directly, they propose to split the prediction space into several bins. Then, the network predicts a bin and a residual from the bin center. As the bin prediction is a classification problem, the cross-entropy loss is used for them and the smooth L1 loss for the residuals. As most of the boxes are on a similar height only one residual is used for regression in y. This is goes hand-in-hand with our problem one assumption of bounding boxes having a planar base. For the size, they employ smooth L1 loss to regress a residual to the average car size. Hence, the output of the network would be bins and residuals for the location $[\text{bin}^i_{\Delta x}, \text{bin}^i_{\Delta z}, \text{res}^i_x, \text{res}^i_y, \text{res}^i_z]$, only residuals for the size $[\text{res}^i_{\Delta h}, \text{res}^i_{\Delta w}, \text{res}^i_{\Delta l}]$, and bins and residuals for the orientation $[\text{bin}^i_\theta, \text{res}^i_\theta]$.

1.2 Implementation idea

We planned to conduct an ablation study documenting the additional performance gain of each of the improvements. Firstly, we planned to make use of data augmentation as it is industry standard and makes training models more robust. The two experiments we wanted to contrast are as follows. The first study would involve training the RCNN net with our loss functions from problem 1. I.e. we decided to use the RCNN net as our stage-2 network, send its bin based outputs through a transformation function (`bbox_trasform.py`, taken from the official GitHub repo) to obtain the 7 bounding box features, which would then be used to compute losses as done in problem 1. Our second plan was to train the RCNN network according to what is proposed in paper [2]. In this study the losses would be computed using the original bin based outputs of the RCNN network. Finally, we would have contrasted the results of these two networks.

Unfortunately, the training of Task 6 (Section 1.6 in the report for problem 1) took us an unexpectedly large amount of time, due to instances crashing at the first epoch. Hence, we only got to implement the the data augmentation, canonical transformation and RCNN model implementations with losses proposed in the paper. In order to make our approach trainable we would further need to adjust the training files for the different proposed implementations.

References

[1] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection, 2021.

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