Deep Learning for Autonomous Driving: Lab 2

Team 30
Alan Savio Paul
Loïc Houmard

July 3, 2022

Problem 1. Building a 2 Stage 3D Object Detector

1.1 Recall and IOU computation

In this first part, we were asked to compute the IOU of 3D bounding boxes and the recall between predictions and targets based on that. To compute the IOU, we first needed to compute the 8 corners of our boxes in the camera coordinate system given their 7 parameters (x, y, z, h, l, w, ry). For that, we just slightly modified our method from Project 1. The idea is basically to compute the 8 corners in the box coordinate (based on its center as origin), then compute a rotation matrix using ry and a translation vector which is the center of the box and apply this to our 8 points. In order to compute the IOU between every prediction-target pair, we looped over them and computed the pairwise IOU. For this, we took into account the fact that the vehicle only has a yaw rotation. We first compute the area of intersection in the bird's eye view between the two boxes. Then, since they cannot have rotation in the two other axis, we just check how much of the height intersect (we take the minimum of the two top faces minus the maximum of the two bottom faces). We then multiplied this intersection height by the intersection area to get the intersection volume. The IOU computation then only consists of this intersection volume divided by the union of volume (computed as the sum of the two individual boxes volume minus the intersection). To compute the recall, we then check whether any of the proposal has an IOU higher than the threshold for each target to compute the true positives and check whether all of the predictions have an IOU lower than the threshold for each target (meaning that no prediction matches a particular target) to compute the false negatives.

The recall is a good metric to evaluate the first stage detection because the false negatives are what we want to avoid the most since we want to be sure that there aren't many targets that were forgotten or predicted very badly, which would prevent our second stage to find them even with perfect refinement. Furthermore, since we are going to refine our predictions anyways, we don't care

as much about the false positive and therefore the precision. The average recall on the full dataset (as used later on to train) having 1712 frames is ≈ 0.81342, which is pretty good but clearly not perfect.

In this part, we passed the test with a pretty fast runtime of 30.0ms/63.7ms on AWS's CPU.

1.2 ROI pooling

In this section, we had to implement the ROI pooling. For that, we first enlarge the boxes on every side making sure that the center remains at the correct place. We then retrieved the 8 corners in camera coordinate of all our boxes using our function from task 1. Like this, we can easily check which points are inside each boxes using only '>' and '<' operations which are cheap to compute. Then we simply keep the boxes which have at least one point inside them and randomly sample the points to have 512 for each box. Figure 1 show 3 different scenes with on the left (figures 1a, 1c and 1e) only one box with its sampled points and on the right (figures 1b, 1d and 1f) the full scene after ROI pooling. Table 1 shows the corresponding frame name on the left and the values of the bounding box. Note that we used the minival set provided on Piazza as well as visualizations tool from project 1 in order to create the figures and used the 3 first frames.

For this task, we passed the test with a runtime close to the baseline provided (35.5ms/38.4ms). Note however that the runtime was varying quite a lot between different runs and was sometimes closer to 50ms.

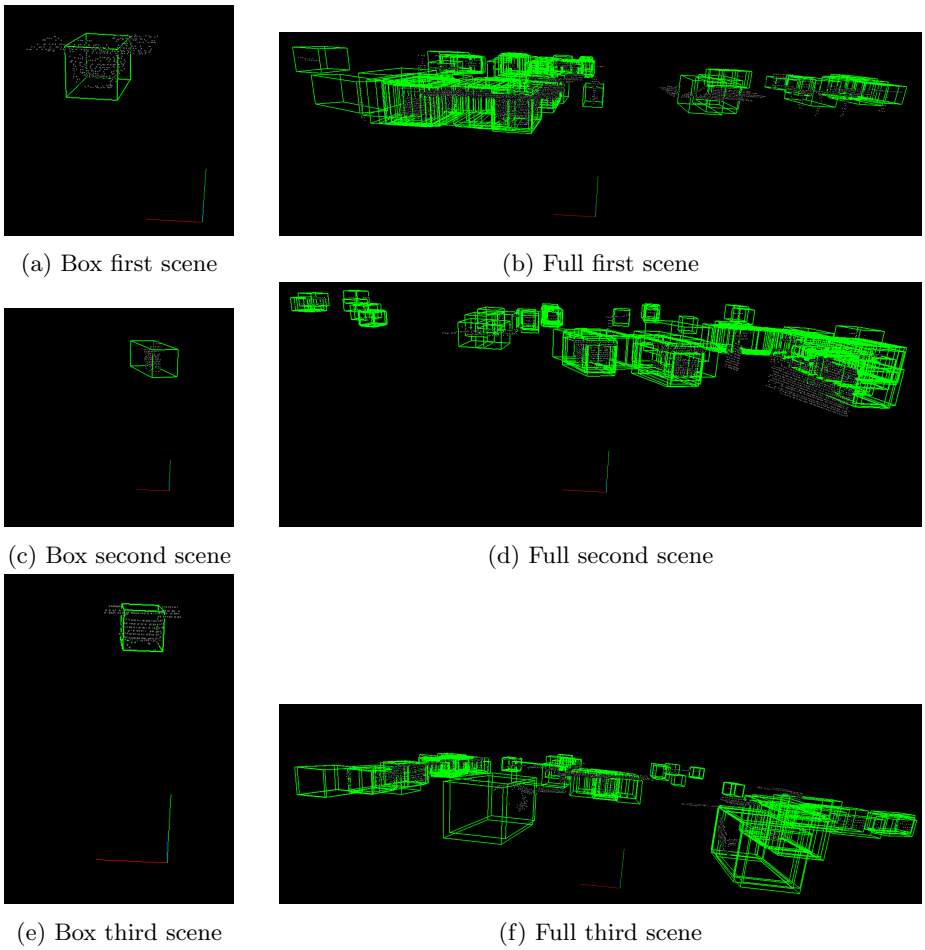
Frame name	Bounding box values
elpBlgmdIiJkAAVuYPB2	[3.5176 1.6643 11.0931 1.5604 1.6181 3.7449 -1.5691]
rWvdEifFU9YpFeFyMroa	[1.0739 1.6274 20.2643 1.6843 1.6877 4.1997 -1.1749]
DXdxO5H7RXbfVtSqBVd8	[1.1498 1.7912 24.3467 1.562 1.633 3.7346 -1.4909]

Table 1: Corresponding frame name and bounding box parameter for figure 1

1.3 Proposal Sampling

In this task, we assigned target bounding boxes to our inputs such that we can supervise our training, and then implemented a sampling strategy to have foreground and background samples forwarded to the second stage refinement network. For the first part, we simply assign each proposal the target box with which it scores the highest IOU. For the second part, we first prepared the foreground, background, and extended foreground sets, and then closely followed the instructions in the handout for implementing the sampling strategies for every listed case such as those where all samples are foreground/background.

One implementation detail worth mentioning is that in cases where we had fewer samples than required, we chose to generate a fully randomly sampled set of required size by taking random samples from the small sample set. An alternative would have been to use the original small sample set and pad it with



(a) Box first scene

(b) Full first scene

(c) Box second scene

(d) Full second scene

(e) Box third scene

(f) Full third scene

Figure 1: ROI pooling for 3 different scenes

random samples to obtain the required size. The reason for our choice is that we felt the random padding in the handout could be interpreted either way. However, we think one disadvantage of our choice is there is a chance that some rarely-occurring proposals could be left out. In Problem 2, we experiment with a variant where we use numpy's tile method to pad the array instead and we observed a small improvement.

The chosen sampling scheme is required because it makes the training process more effective. If we had instead randomly sampled from all proposals for each scene, we would end up with many more background proposals than foreground proposals in each batch (because of the nature of autonomous driving scenes) and our network will consequently not get useful supervision signals for regressing the boundary box parameters.

Easy background proposals are the background samples that the network is very confident about. These samples are crucial to guide the training in the early stages. Studies such as [1] have shown that maintaining a mix of both easy and hard samples in each batch is important for training the network.

If we consider 0.5 as the boundary for foreground and background sample classification, then we can end up with wrong or unreliable labels as proposals with IOU values around the 0.5 mark could actually belong to either class. To reduce the chance of including proposals with labels of high uncertainty, we use a margin so that 0.45 is the upper bound of background samples' IOU and 0.55 is the lower bound of the foreground samples' IOU.

We need to match the ground truth with its highest IoU proposal because this would help us ensure that every target having at least one overlapping proposal is included in the training. Without this step, we could have some targets with overlapping proposals being completely missed out because those proposals have a greater IoU with another target.

1.4 Loss functions

Here, we only had to implement the two loss functions for the regression of the bounding boxes and for the classification. There is not much to say as we simply used the provided formulas to compute them.

1.5 Non-maximum Suppression (NMS)

In this part, we implemented NMS to filter the final predictions which were close to each other. We simply implemented the algorithm shown in the handout using the bird's eye view (BEV) 2D IOU computed with our method from task 1 (setting $y = 0$ and $h = 1$). The main advantage of using BEV IOU is that the position (x and z) are more important to be kept than the height, since it is easier later on to constraint the cars to be on the floor. Also, most objects' y values are within a very small range and hence it must be easier for the network to learn how to correct the height of predictions than to correct the other two axes. Hence using 2D IOU will keep boxes which are correctly located, but might be at a wrong height. One disadvantage however is that it can provide

(a) Training loss base model

(b) Validation loss base model

(a) Easy mAP

(b) Moderate mAP

(c) Hard mAP

totally wrong results if there is a bridge for example and a car is actually on top of another.

1.6 Training

While training the network, we realized that everything was going as expected since the loss on both training set and validation set went down (Figures 2a and 2b). The mAP of each difficulty level (Figure 3a, 3b and 3c) went up after some time as expected as well, since at the beginning, when our network is not capable of finding any bounding box, the mAP will just stay quite low.

Figures 4, 5 and 6 show results of the network prediction on the same scene at different stages of the training. As we can see, at very early stages, the network seems to predict very randomly and most of the predictions are incorrect. After some training, it improves, especially on objects which are close since more points are available and hence they are easier to predict. At the end, the network provides pretty good results and manages to predict all of the bounding boxes with pretty high IOU. We obtained results a bit lower than those expected on the public leaderboard as shown in table 2.

	Our results	Expected Results
Easy	74.54	**83.99**
Moderate	67.52	**74.14**
Hard	61.99	**72.70**

Table 2: mAP metrics for our model against its expected result

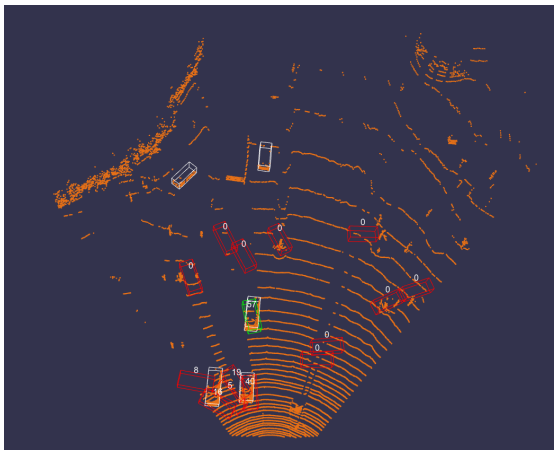


Figure 4: Scene prediction after epoch 1

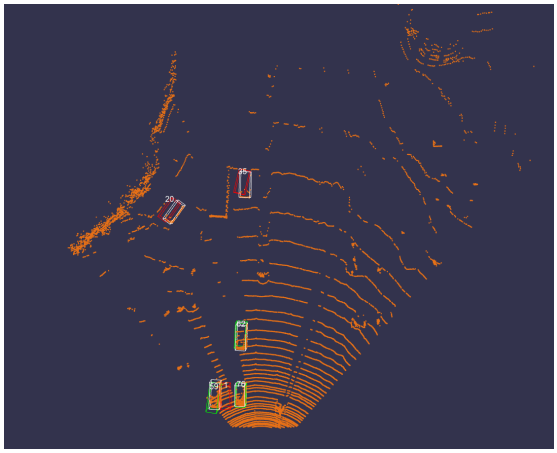


Figure 5: Scene prediction after epoch 12

References

[1] Tianxiang Gao and Vladimir Jojic. Sample importance in training deep neural networks. 2016.

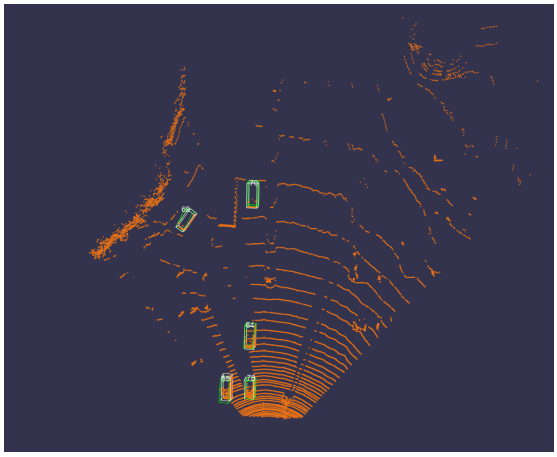


Figure 6: Scene prediction after epoch 34