DLAD Homework 3 - Problem 1
Building a 2 Stage 3D Object Detector

Yunke Ao, Kaiyue Shen

July 4, 2021

Contents

1 Introduction

3D object detection plays an essential role in autonomous driving. In this task, we build a 2 stage 3D object detector. To be more specific, we refine the proposals provided by the first stage Region Proposal Network (RPN) through a detection pipeline consisting: Recall Computation, ROI Pooling, Sample Proposals, Loss Computation and Non-maximum Suppression (NMS). Then we train the complete network, which would be used as the baseline for comparison in Problem 2.

2 Compute Recall

In this task, we implement 3D IoU computation given predicted and target boxes, and compute the recall of the first stage network proposals for the validation set. The eight corners are first extracted based on the parameters of boxes. Then the 2D intersection between boxes in the X-Z plane are calculated using Shapely library. Finally, this intersection is multiplied by the overlap in the Y dimension and divided by the volume of the union of the 2 boxes. To compute the recall, we calculate IoUs between each prediction-target pairs and summarize in the matrix, where the rows and columns represents predictions and columns respectively. The recall is just the ratio between the column where exists an item larger than the threshold and the column number. Here the average recall of the validation dataset is 80.1%. It shows that the first stage can already generate a coarse but meaningful prediction.

In this stage we use recall rather than precision, because for RPN network, there could exist multiple overlapped proposal that correspond to the same ground truth. While how many proposals are for the same target is not clear, and it may influence the precision result, recall is more reliable because the targets do not overlap.

For this task, we achieve the result 80.1/80.1 and duration 22.5/63.7.

3 ROI Pooling

We achieve ROI pooling efficiently. The process with most computational cost is to select the points of the scene within the increased box region. Instead of computing each selection condition (range of points in each dimension in the box coordinate) independently and combining them together, we implement the selection step by step. Before transformation of points to the box centered coordinate, we first apply a coarse selection based on a larger square centered at the box center. Then we only compute the X coordinates of selected points in the box centered frame and select the points within the enlarged box in X direction. Then among the selected points, we similarly compute the rotated Y and Z coordinates to further select all the points in the enlarged box. The code for this part is shown below:

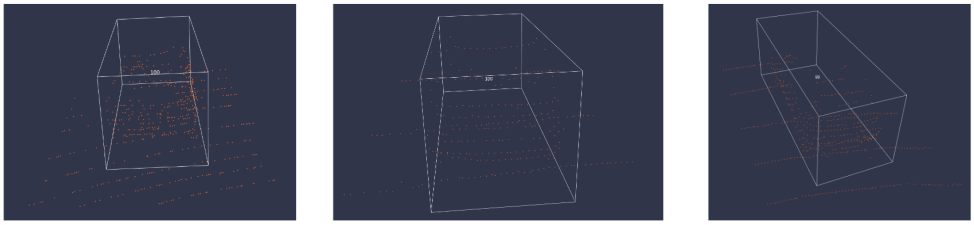
```
    # transform points to the box coordinate
        ry = pred[i,-1]
        bx = pred[i, 0]
        bz = pred[i, 2]
        by = pred[i, 1]
        bh = pred[i, 3]

        # init coarse filter to speedup selection
        cond = cond_1[abs(x - bx)+abs(z - bz)<1.415*max_dist[i]]

        # filter points in the box
        y_unrotate = y[cond] - by
        cond_y = abs(y_unrotate + bh/2)< bh/2 + delta
        cond = cond[cond_y]

        z_unrotate = math.sin(ry) * (x[cond] - bx) + math.cos(ry) * (z[
                                                cond] - bz)
        #z_unrotate = z_unrotate_all[i, :]
        cond_z = abs(z_unrotate) < range_z[i]
        cond = cond[cond_z]

        x_unrotate = math.cos(ry) * (x[cond] - bx) - math.sin(ry) * (z[
                                                cond] - bz)
        cond_x = abs(x_unrotate) < range_x[i]
        cond = cond[cond_x]
```

After this, the selected points are sampled with or without replacement according to the number of the points. Specifically, if the number of points is larger than the required number, we sample with replacement. Otherwise, we first take in all selected points and then sample more points from them with replacement until we get the required number of points. The same random sampling strategy is used in task 1.3.

With this procedure and careful usage of data types for indexes and variables, we achieve duration of 30.8/38.4 ms. Results for this task is shown in Figure 1, the extracted points are all surrounding or inside the proposal boxes.

4 Sample Proposals

From task 1.2, we already get the input of the refinement network. In order to do supervised learning, we generate the input-output pair for each scene in this task. First, we compute the IoU matrix as in task 1.1. Then we separate input samples into different categories, including foreground, easy background and hard background, based on the criteria in Figure 2. There are two cases. We first perform max operation along the row of IoU matrix, to find the ground truth (GT) for each input and assign the sample into certain category based on the IoU value of the input-GT pair. We then perform max operation along the column of IoU matrix, and take each pair as the additional



(a) (b) (c)

Figure 1: Example ROI pooling results. They are the first valid proposal from the 1st, 2nd and 3rd frames respectively. The parameters (x, y, z, h, w, l, r) for the three examples are [3.5176, 1.6643, 11.0931, 1.5604, 1.6181, 3.7449, -1.5691], [1.0739, 1.6274, 20.2643, 1.6843, 1.6877, 4.1997, -1.1749] and [1.1498, 1.7912, 24.3467, 1.562, 1.633. 3.7346, -1.4909] respectively.

foreground. After this, we put them together and if we are in the training mode, we move on to the next step, otherwise, we directly output these pairs. In the extra step, we re-sample proposals under certain rules. You can find details in the following code.

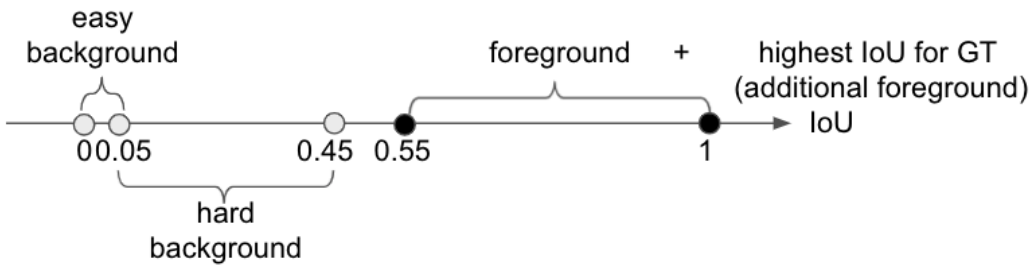


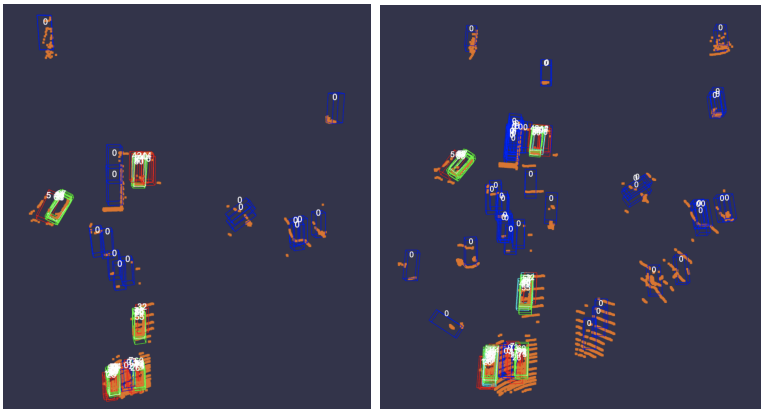
Figure 2: Illustration of how we divide samples into different categories depending on IoU

```
    # If there are only foreground proposals in a scene
    if num_easy_bg + num_hard_bg == 0:
        sample_mask = random_sample(num_fg, num_samples)
        pred_targ_IoU_pair = fg[sample_mask,:]
    # If there are only background proposals in a scene
    elif num_fg == 0:
        pred_targ_IoU_pair = bg_sample_proposals(num_samples,
                                                 bg_hard_ratio, num_easy_bg,
                                                 num_hard_bg, easy_bg,
                                                 hard_bg)
    # If there are more than 32 foreground samples
    elif num_fg > num_fg_sample:
        fg_sample_mask = random_sample(num_fg, num_fg_sample)
        fg_pred_targ_IoU_pair = fg[fg_sample_mask,:]
        bg_pred_targ_IoU_pair = bg_sample_proposals(num_samples-
```

```
                                            num_fg_sample, bg_hard_ratio
                                            , num_easy_bg, num_hard_bg,
                                            easy_bg, hard_bg)

        pred_targ_IoU_pair = np.concatenate((fg_pred_targ_IoU_pair,
                                            bg_pred_targ_IoU_pair), axis
                                            =0)
    else:
        bg_pred_targ_IoU_pair = bg_sample_proposals(num_samples-num_fg,
                                            bg_hard_ratio, num_easy_bg,
                                            num_hard_bg, easy_bg,
                                            hard_bg)
        pred_targ_IoU_pair = np.concatenate((fg, bg_pred_targ_IoU_pair),
                                            axis=0)

def bg_sample_proposals(num_needed, bg_hard_ratio, num_easy_bg,
                                    num_hard_bg, easy_bg, hard_bg):
    if num_easy_bg == 0:
        sample_mask = random_sample(num_hard_bg, num_needed)
        pred_targ_IoU_pair = hard_bg[sample_mask,:]
    elif num_hard_bg == 0:
        sample_mask = random_sample(num_easy_bg, num_needed)
        pred_targ_IoU_pair = easy_bg[sample_mask,:]
    else:
        num_needed_hard = int(num_needed * bg_hard_ratio)
        num_needed_easy = num_needed - num_needed_hard

        sample_mask_hard = random_sample(num_hard_bg, num_needed_hard)
        pred_targ_IoU_pair_hard = hard_bg[sample_mask_hard,:]

        sample_mask_easy = random_sample(num_easy_bg, num_needed_easy)
        pred_targ_IoU_pair_easy = easy_bg[sample_mask_easy,:]

        pred_targ_IoU_pair = np.concatenate((pred_targ_IoU_pair_hard,
                                            pred_targ_IoU_pair_easy),
                                            axis=0)

    return pred_targ_IoU_pair
```

From Figure 3, you can more intuitively understand the sample proposals. Here, for visualization, we do not consider the second case: additional foreground got from the highest IoU for GT. We simply label different proposals with different colors based on their IoUs. The image on the left is under the training mode so it undergoes the re-sampling while the one on the right not as it is in the validation/test mode. Compared them with each other, you can easily see there are more blue boxes than other colors in the right image, so with our re-sampling scheme, the number of bounding boxes in different categories are more balanced.

Now, we explain the idea of re-sampling scheme and criteria of category division from the theoretical point of view. The re-sampling scheme plays an important part in solving the problem of class imbalance. For example, if we simply random sample from all



(a) #bounding box=64 (b) #bounding box=#RPN outputs

Figure 3: Visualization of sample proposals in different modes. Left: train, Right: val/test. Green: foreground, Red: hard background, Blue: easy background, White: ground truth, Yellow: others. #Frame=1

proposals, a large amount of data in the final proposal may only belong to the majority. So that during the classification, the model will tent to predict that class. Also, since the model hardly see the minority, when do the validation, it cannot generalize well and will easily fail on that class when doing the regression task. If we don't sample easy background proposal at all, we will meet the class imbalance problem as well. Moreover, the hard background proposal actually contain a lot of points that are also contained in the foreground proposal. For similar point inputs, the model need to predict different labels, which makes the training harder. The criteria of category division is also very important. Now there is a gap between the upper bound of hard background 0.45 and lower bound of foreground 0.55. If we drop this gap and let 0.5 to be the split point of background and foreground, there will be two problems. First, samples close to the split point may have similar IoU but different class, making the training procedure unstable. Second, if we don't separate background to be easy ones and hard ones, the model may work well on easy ones but confuse when meeting the hard ones. But why we match the ground truth with its highest IoU proposal? The reason behind it is that otherwise some ground truth bounding box might not be used if every predict in the first stage has low IoU with them. In reality, it means the missed detection of a car, which might be dangerous.

5 Compute Loss

We implement two losses for two tasks. For the regression task, we first filter out the negative samples (with 3D IoU < 0.55) since negative samples are more likely to be the background and there is no physical meaning predicting the location of the background. The description of a bounding box contains 7 parameters: (x, y, z) for the center lo-

cation, (h, w, l) for the box size, ry for the box orientation, where the box size weigh more importance. So we separately compute the smooth-L1 loss for the location, size (multiplied by 3), and rotation and sum them up to get the final regression loss. For the classification task, we first filter out the samples that are neither positive nor negative (3D IoU lies in ambiguous section (0.45, 0.6)) to make the training more stable. Then we label the positive samples with 1 and negative with 0. Finally, we compute the Binary Cross Entropy (BCE) loss. Our implementation passed the test.

6 Non-maximum Suppression (NMS)

Before directly evaluate the output of the refinement network, we make use of Non-maximum Suppression Algorithm to remove the highly overlapping bounding boxes. The algorithm works like this: we have two sets, one holding the prediction before NMS and one after. First, the former is initialized with outputs of the network and the latter is empty. Second, we move one with the highest confidence score in the former to the latter. We accept this prediction because we highly believe in it. Third, we compute its 2D BEV IoU with remaining ones in the former set. If the IoU is larger than the given threshold, meaning two bounding boxes are highly overlapped and they are likely to predict the same object. We just keep the one we believe more and remove another from the former set. We iterate between step 2 and 3 until the former set becomes empty. The latter is the set we want.

Although we use 2D BEV IoU in this task, we can of course use 3D IoU computed as the previous task. The thing is our goal is to remove the overlapping bounding boxes. From our daily experience, most of the time, there is only one car at each position of the 2D plane - cars are not vertically stacked. Also, in our case, the car must be on the ground and have similar heights. Therefore, if two bounding boxes have a high 2D BEV IoU, they are very likely to contain the same object. Using 2D BEV IoU is good enough to find the redundant boxes. Moreover, compared with 3D IoU, it is more computational efficient and is easier for us to find a proper threshold.

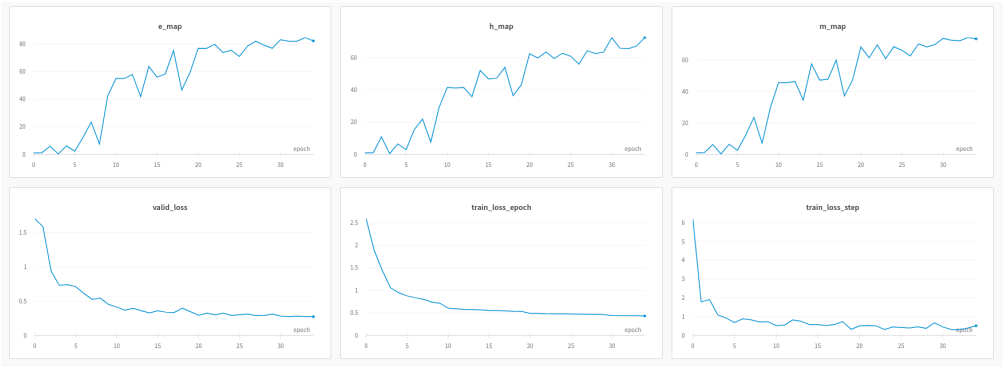
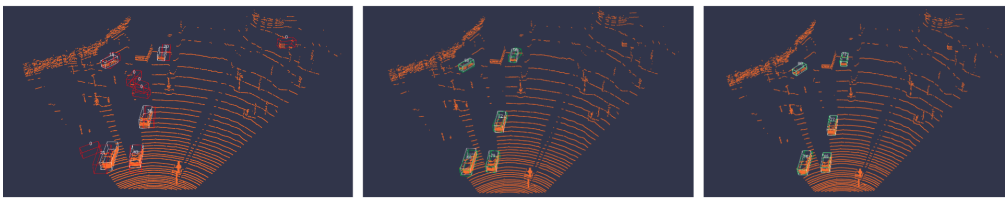


Figure 4: Loss and precision curves for training the baseline.



(a) (b) (c)

Figure 5: Scene visualization examples from the beginning, mid-way and end of the training cycle.

7 Training the Network

In this section we train the network for 35 epochs, in 8.5 h in total. The loss curve and precision trend is shown in Figure 4. The loss continuously goes down and the precision increase during the training process until reaching the performance "e_map: 82.18946, mmap: 73.39667, hmap: 72.52704". The scene visualization examples through the training process is shown in Figure 5. It can be seen that at the beginning the proposals do not overlap well with the targets. At the mid-way of training, many predicted boxes have better overlaps with the target, while the poses of boxes are still not accurate enough. At the end of training, most of the predicted boxes have similar pose as the targets.