CVL, ETH Zurich | Deep Learning for Autonomous Driving

Project 3: 3D Object Detection from Lidar Point Clouds

13/05/22 - 03/07/22

General Info: This project accounts for 15% of your grade. You are welcome to ask questions in Piazza. If you think that your question may reveal the solution, please list your question for instructors only.

Training: The AWS accounts distributed during Project 2 can still to be used for this project. For exercise 3, you will have an **AWS budget of 350 USD**. If you exceed your budget, we might disable the AWS account without previous warning or deduct points from this exercise. Please refer to the README.md of the provided solution template for further important information.

Grader: The graders purpose is to evaluate a submission archive corresponding to one experiment run on the test data. This is required to report scores of solutions to the exercise problems. As in Project 2, the grader is hosted on CodaLab[1]. You should use the same team account as for exercise 2: `dlad22_teamXX`, where XX is your team number (with a leading zero if ≤ 9). After logging in, follow the grader URL (see Quick Links), and click `"Participate" -> "Register"`.

To get a submission graded, navigate to `"Participate"` $\rightarrow$ `"Submit / View Results"`, select the correct problem, then click on the `"Submit"` button, and wait to get the submission archive uploaded. Codalab begins file upload immediately after it was selected in the system dialogue, and does not indicate the upload progress. After the upload has finished, the status will change to indicate grading has been scheduled.

Submission: Your submission must contain two subdirectories: (1) `problem_1/` which contains the task files `utils/task*.py` that were used for problem 1 under a `solution` directory along with a `report_1.pdf` which contains a concise description of your solutions along with the resulting figures; (2) `problem_2/` which contains the necessary files of your repository (e.g. without the `tests/` directory) that was used for problem 2 along with a `report_2.pdf` which contains a concise description of your solution. For further detail on `report_2.pdf` please refer to the problem statement. Final reports must be sent by each team in a file named dlad_ex3_teamID_student1_student2.zip to Ozan Unal (ozan.unal@vision.ee.ethz.ch) by 23:59 CET 03.07.2022 with subject `"DLAD EX3: TEAM_ID"`.
Example (team 1): (name) dlad_ex3_01_ArthurDent_FordPrefect.zip (subject) DLAD EX3: 01
The submissions are automatically collected and sorted, so please make sure you follow the naming protocol.
Important: Sharing the code, dataset, configurations, and run artifacts (W&B) with anyone except your team mate is not allowed at any time (even after the end of the course). To use versioning (GitHub or GitLab), make sure to use a private repository. Do not change the visibility of W&B project to public.
Introduction: Unlike Project 2 where you have worked with 2D images, in this exercise we will be working directly with irregular 3D point cloud data. We will be looking into one of the core problems of 3D computer vision: 3D object detection. In specific, we will be building a 2-stage 3D object detector to detect vehicles in autonomous driving scenes, i.e. to draw 3D bounding boxes around each vehicle (remember Task 2.3 from Project 1). An object bounding box has seven degrees of freedom which need to be regressed: the center (x, y, z), the size (h, w, l) and the yaw rotation r in radians. In autonomous driving, we consider the pitch and roll to be zero as all object lie on the ground plane.

[1] http://dlad-codalab.com/competitions/21?secret_key=139a4657-5c3f-4f2c-87ec-23b0b0e8d8e7

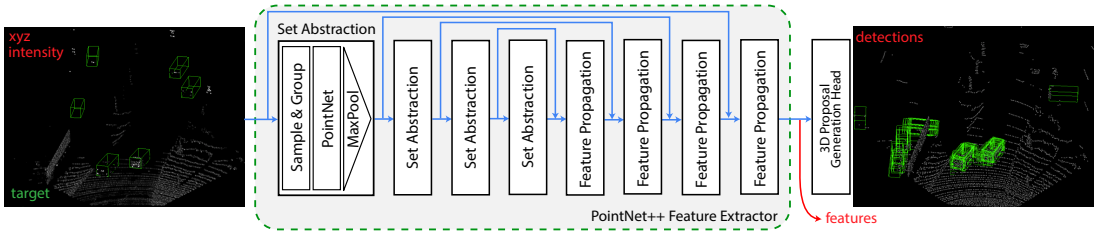

Figure 1: Stage-1 RPN architecture. Accessible data keys for the second stage input are written in red. For detections, shown are only the first 50 proposals generated by the first stage network. Best viewed in color.

On top of the seven parameters, two further values need to be predicted for each bounding box: (1) the class score and (2) the confidence score. The class score is used to determine to which class the box belongs, and since in this project we only consider the detection of a single class (car) we forgo the class score estimation all together. The confidence score on the other hand gives the likelihood at which an object lies within a bounding box, which, as the name suggests, essentially gives us the networks confidence in its prediction.

A 2-stage detection pipeline is a common approach employed in many state-of-the-art 3D object detectors [1, 2, 3]. The first stage, which is often referred to as the Region Proposal Network (RPN), is used to create coarse detection results from the irregular point cloud data. These initial detections are later refined in the second stage network to generate the final predictions. The idea behind such a 2-stage design is to process the entire point cloud in the first stage and to generate regions-of-interest (ROI), i.e. smaller pools of information at and around likely objects which later can be refined individually to form the final detection.

For this project, the first stage (which is commonly where the bulk of the network resides as one can imagine) is prebuilt and pretrained to save time. What you are given are therefore the initial predictions, i.e. the results of the first stage RPN alongside the extracted features and point cloud data (illustrated in Fig. 1). To learn more about the data we recommend investigating it on your own.

Materials: In the gitlab repository[2] we provide a solution template for this project. In the solution template you will find a dataset script to access the results of the first stage network.

- `dataset.py`: Pytorch Datatset object to import the provided dataset. Each data point describes an drivable scene with (N,3) point coordinates, (N,128) corresponding RPN features, (N,) corresponding intensity measurements along with RPN coarse detection results and ground truth annotations. To easily collate data for processing, point clouds of each scene were randomly sampled to contain N=16384 points. If a point cloud contained less than N points, it was padded by random point repetitions. Each scene contains K coarse detections that are described through 7 parameters (x, y, z, h, w, l, r) as in Project 1. For the training samples K=300, for the validation and test sampled K=100. The initial proposals are passed through a low threshold NMS, hence contain 0's that are filtered out during access. The implemented `get_data()` function can be used to access the data keys individually. Please refer to the function description for more information.
  The Dataset object requires Tasks 2 and 3 to be completed to function within the training script. This file does not need to be altered for Problem 1.

In addition to the dataset script, the solution template contains the following:

- `utils/`: Under the `utils/` directory you are given a python script designated for each task following the notation `task*.py`. Your solutions for Problem 1 *must* be implemented in these functions with pre-written return statements as these will be called later within various sections of the main code. All necessary libraries and modules for each task are imported for you, but feel free to import additional libraries that you require for your

[2] https://gitlab.ethz.ch/dlad21/exercise3

solutions.
We additionally provide evaluation utilities `eval.py` and a visualization script `vis.py`. These are called within the training loop and do not need to be altered.

- `tests/`: For each problem we also provide a test case which can be executed through `tests/test.py` to verify some of your results. It is highly recommended that you follow the ordering of the tasks and only move on when your results match that of the recorded tests. While you are not given a test case for tasks including randomness, we highly recommend that you validate your solutions through other means, e.g. by visualizing your implementation using your solution from Project 1 (`3dvis.py`).

- `config.yaml`: Configuration file for the data generation, network architecture, training and logging. The output directory is given by `trainer:default_root_dir` and must be set. Each high level key is treated as an iterable whose elements are passed on as arguments to the corresponding initialization functions. The trainer and logger declarations can be found here[3,4] respectively.
Example: Any argument in the pl.Trainer init function can be declared in `config.yaml` to be passed to the pl.Trainer object. To resume from a previous checkpoint simply declare `trainer:resume_from_checkpoint:'YOUR/CKPT/PATH'`.
All parameters given in the task descriptions can be found in the config file. Each function description contains the necessary keys to access these parameters.

- `model.py`: A base network implementation for the second stage decoder (illustrated in Fig. 2). The network takes in sampled pooled ROI's and predicts the 7 degrees of freedom of a 3D bounding box along with a confidence score. The network consists of three set abstraction layers with increasing radii. The last set abstraction layer acts globally on the entire subsampled set. Each task head consist of a two layer MLP. Architecture details are determined using the configuration file `config.yaml` and should not be modified for Problem 1.

- `train.py`: Pytorch lightning training script that requires Task 2, 3 and 4 to be completed to function, and Tasks 1, 5 for correct performance tracking. The first sample in the validation set will be visually logged in W&B every epoch. This file does not need to be altered for Problem 1.

Problem 1. Building a 2 Stage 3D Object Detector (10 point(s))

Our goal for Problem 1 is to refine the proposals provided by the first stage RPN.

Before we can begin with the refinement of our first stage results, we need to first determine if the results are good enough to be refined to begin with, i.e. we need to evaluate the performance of our first stage detector and check if we have successfully (coarsely) detected the vehicles in the given scenes. The question thus becomes: How? How can we determine if an object is detected, i.e. if a bounding box prediction is correct?

To answer this we need to first understand a crucial metric: the Intersection-over-Union (IoU). The IoU of a bounding box and its corresponding ground truth states the ratio of the intersecting volume over the union of the volumes. Thus, an IoU of 1 implies that the predicted bounding box perfectly overlaps with the ground-truth, and an IoU of 0 implies that the predicted bounding box and the ground truth do not intersect in the 3D space.

To determine if the detection is correct or wrong, a threshold value is set for the IoU. Given a threshold of t there are four cases that can be observed: (1) True Positive (TP), if the IoU is greater or equal to t the prediction is correct (2) False Positive (FP), if the IoU is less than t the prediction is wrong (3) False Negative (FN), if there exists a ground truth bounding box but no corresponding detection, i.e. if the IoU of all predictions within a scene never exceed t for a given ground truth box (4) and sticking by definition True Negative (TN), if there is no object

[3] https://pytorch-lightning.readthedocs.io/en/1.2.0/api/pytorch_lightning.trainer.trainer.html
[4] https://pytorch-lightning.readthedocs.io/en/stable/api/pytorch_lightning.loggers.wandb.html

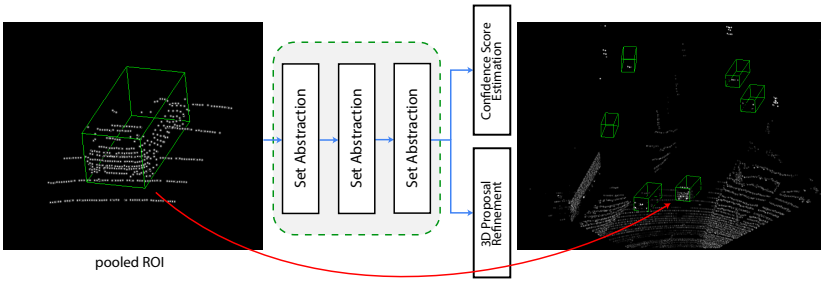

Figure 2: Stage-2 box refinement network architecture and illustration of ROI pooling on point coordinates.

and we did not predict one. As one can imagine, TN is not at all useful for object detection, hence we mostly ignore it.

Back to our quest of evaluating the first stage results... The recall, which is a metric that states the ratio of the true positives to all positives (Recall=TP/(TP+FN)), is perfect to assess the quality of our first stage proposal generator.

1. Given a threshold of $t = 0.5$, compute the average recall of the first stage network proposals for the *val*-set.
   Report: What is the average recall for the *val*-set? Why is recall a good metric to assess the quality of the first stage proposals (as opposed to e.g. average precision)?

   Hint: Make sure to check your implementation with the given benchmark as the IoU calculations will come up in most of the following tasks.
   Note: Implement this task in `utils/task1.py`. The `compute_recall` function should return the recall for a set of predictions and target bounding boxes. You can and *should* divide your workflow into other functions within the file to increase readability. For task 1, you are given two additional functions to complete: `label2corners` and `get_iou`. All three functions will later be needed for other tasks. Please refer to the function descriptions for more information.
   Test: You can test your implementation by either executing `python tests/test.py --task 1` to run on your local machine or by following the instructions in the README.md of the code template to run on a CPU node on AWS. Make sure you successfully pass the test before moving on. If you exceed the duration test, make sure to rethink about your implementation as this will cause a severe bottleneck when you start to train the network.
   Fun fact: Almost all 3D object detection benchmarks are evaluated using a threshold of 0.7 IoU for vehicles and 0.5 for harder object such as pedestrians and cyclists. However for 2-stage detectors, the common practice is to evaluate the first stage results with a 0.5/0.25 IoU threshold.

Now that we know we have a solid foundation, let's begin preparing our data for the second stage refinement network. For each scene we are given roughly 300/100 detections, which mean 300/100 possible inputs for the second stage network. We want to not only feed the point coordinates and intensities into the second stage but also all other information extracted in the first stage.

2. Pool ROI's to form the input for the second stage model. An illustration of ROI pooling on point coordinates can be seen in Fig. 2 - left.
   Report: Generate 3 such examples of ROI figures including the points and bounding box from 3 different scenes. You can use your results from Project 1 to perform the visualization or use the `utils/vis.py` to visualize with W&B. In your report, include the frame names and the bounding box parameters corresponding to each figure.

   (a) Enlarge predicted 3D bounding boxes by $\delta = 1.0$ meters in all directions. As our inputs consist of coarse detection results from the stage-1 network, the second stage

will benefit from the knowledge of surrounding points to better refine the initial prediction.
Hint: Make sure your bounding box center remains in the center of your enlarged bounding box! This will help when filtering the points in (b).

(b) Form ROI's by finding all points and their corresponding features that lie in each enlarged bounding box. Each ROI should contain exactly 512 points. If there are more points within a bounding box, randomly sample until 512. If there are less points within a bounding box, randomly repeat points until 512. If there are no points within a bounding box, the box should be discarded.

Note: Implement this task in `utils/task2.py/roi_pool()`. The function should return 3 values: (1) an np.array of valid bounding boxes of size (K',7) with K' indicating the number of bounding boxes in the scene that contain at least one point, (2) an np.array of size (K',512,3) for all points within each box and (3) an np.array of size (K',512,128) for the corresponding features of all points within each box. As always, you can and *should* divide your workflow into other functions within the file to increase readability.
Test: You can test your implementation by either executing `python tests/test.py --task 2` to run on your local machine or by following the instructions in the README.md of the code template to run on a CPU node on AWS. Important: This test will not check the sampling and point selection but only that you have the valid bounding boxes. Make sure you successfully pass the test and confirm that your implementation is correct through visualization which is required for the report. As mentioned before, you can use your results from Project 1 Task 2.3. If you exceed the duration test, make sure to rethink about your implementation as this will cause a severe bottleneck when you start to train the network.

Now that we have our inputs we need to assign each input a target bounding box such that we can supervise our training. In other words, we need our outputs. For this we will be employing a rather simple method where we assign each coarsely predicted bounding box a ground truth target with which it has the highest IoU. Luckily we have most of the code ready for this from Task 1.

Furthermore we need to be clever with what samples we input to our second stage network. Autonomous driving scenes inherently favor background samples (samples that are not objects) as opposed to foreground samples (objects) since there are at most a handful of objects in a typical scene.

For this project we consider a proposal to be a foreground sample when the IoU with its ground truth bounding box is greater than or equal to 0.55 and a background sample when the IoU is less than 0.45. If a background proposal has less than 0.05 IoU with its corresponding ground truth it is considered as an easy background sample. Otherwise it is considered a hard background sample. Furthermore, for 2-stage detectors, each bounding box that has the highest IoU for a ground truth bounding box is considered an additional foreground sample regardless of its IoU (assuming > 0). When training our network, to benefit more from the classification pipeline, it is vital to feed samples from all three classifications as to not overfit to a single set.

3. Generate the final input-output pairs per scene.
Report: Visualize a scene from the training set with all proposals. As opposed to Fig. 1 - right your scene should only contain the pooled point. In your report, include the frame name.

   (a) Using the highest IoU, assign each proposal a ground truth annotation. For each assignment also return the IoU as this will be required later on.

   (b) For training only: sample 64 proposals per scene. (1) If there are only foreground proposals in a scene, all 64 samples should be foreground. If less than 64 exist, they should be randomly repeated until 64, if more exist, they should be randomly sampled until 64. (2) If there are only background proposals in a scene, all 64 samples should be background. Background proposal sampling is discussed later. (3) If the scene contains both foreground and background proposals: (3-1) if there are more than 32

foreground samples, foreground proposals should be sampled until 32. The remaining 32 will then be background samples. (3-2) if there are less than 32 foreground samples, all foreground samples should be taken, and the remaining should be sampled from the background proposals such that a total of 64 are sampled per scene.

Background proposal sampling: 50% of the sampled background proposals should be easy background samples and the remaining 50% should be hard background samples when both exist within the scene (again, can be repeated to pad up to equal samples each, if odd number of samples need to be selected, it should be from the easy sample pool). If only one difficulty class exists, all samples should be of that class.

When matching ground truth to proposal, if a proposal is considered positive due to the highest ground truth IoU criteria, it should be considered a new sample (i.e. repeated) with the new ground truth annotation.

For validation: all RPN outputs should be fed as input to the seconds stage module with corresponding assigned targets and IoU's.

Report: 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?
Note: Implement this task in `utils/task3.py/sample_proposals()`. The function should return 4 values: (1) an np.array of shape (64,7) indicating the assigned target labels for each sampled point, (2) an np.array of shape (64,512,3) for the pooled points in each proposal, (3) an np.array of shape (64,512,128) for the pooled features of the points in each proposal and (4) an np.array of length 64 stating all IoU's of proposal - ground truth box pairing. As always, you can and *should* divide your workflow into other functions within the file to increase readability.
Hint: See the imported libraries? You can freely reuse your functions from task 1!

Now that we have our training samples we can already start training. For the regression task we will employ the commonly used smoothL1 loss, which can be interpreted as a combination of L1 and L2 as it acts as an L1 loss when the absolute value of the argument is high (usually > 1), and it acts as an L2 loss when the absolute value of the argument is close to zero (usually < 1). The exact equation is given by:

$$\text{smoothL1} = \begin{cases} 0.5x^2, & \text{if } |x| \leq 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{0.1}$$

For the classification task, i.e. the confidence score estimation, we will use binary cross entropy. The model prediction x after passing through a sigmoid function will be in the interval $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ and will state the likelihood of the predicted bounding box representing a detected object. The exact equation is given by:

$$\text{BCE} = -[y \cdot \log x + (1 - y) \cdot \log(1 - x)] \tag{0.2}$$

As seen in `utils/task4.py`, a base RegressionLoss- and ClassificationLoss modules are already implemented using pytorch.nn prebuilt blocks, which is convenient as these loss blocks automatically handle edge cases such as with possible log(0) and log(1) encounters in the information function calculations. However as you might imagine, these loss modules are not in working condition for our project.

4. Complete the implementation of the loss function modules for the regression and classification pipelines.

   (a) We do not want to define the regression loss over the entire input space. While negative samples are necessary for the classification network, we only want to train our regression head using positive samples as the negative samples most likely don't contain an object to begin with. Use 3D IoU ≥ 0.55 to determine if a sample is positive

and alter the RegressionLoss module such that only positive samples contribute to the loss. Additionally, as the size plays a crucial role in the average precision, following literature, the contribution to the loss of the three size parameters (h,w,l) should be multiplied by 3.
The regression loss will therefore be given by:

$$\mathcal{L}_{reg} = \mathcal{L}_{location} + 3 * \mathcal{L}_{size} + \mathcal{L}_{rotation} \tag{0.3}$$

with $\mathcal{L}_{location}, \mathcal{L}_{size}, \mathcal{L}_{rotation}$ being the smoothL1 loss defined over the respective parameters.

(b) Extract the target scores depending on the IoU. For the training of the classification head we want to be more strict as we want to avoid incorrect training signals to supervise our network. A proposal is considered as positive (class 1) if its maximum IoU with ground truth boxes is ≥ 0.6, and negative (class 0) if its maximum IoU $\leq$ 0.45.

Note: Implement this task in `utils/task4.py`. For both losses the predictions and targets should be filtered based on the given conditions before being input into `self.loss()`.
Test: You can test your implementation by either executing `python tests/test.py --task 4` to run on your local machine or by following the instructions in the README.md of the code template to run on a CPU node on AWS. Make sure you successfully pass the test before starting to train.

At this point, our network should be able to learn and predict a fixed set of vehicle bounding boxes. You can test this out by observing the running loss in the progress bar during training (Obviously, it should decrease). The loss however doesn't tell us an easy to understand story so don't start training just yet as there is one final step: evaluation.

Firstly let's summarise our current state. At this point we should have an array of predictions for each scene. To be precise, 64 refined bounding boxes for each scene as the refinement module only works on individual regions of interest. However, - crazy thought - it is very unlikely that all driving scenes contain 64 vehicles. In fact almost none of them do (think of the possible traffic, ouch!). So our task now is to reduce the number of predictions per scene in a clever way and for this we will employ Non-maximum-Suppression (NMS):

Algorithm 1: Non-maximum Suppression

Given: Set of predictions S_p, corresponding confidence scores C_p, threshold t
Result: Set of non-overlapping final predictions S_f
while S_p *is not empty* **do**
  $D_i = S_p$.pop(i) with $i = \text{argmax}\, C_p$
  S_f.append(D_i)
  S_p.remove(S_{iou}) with $S_{iou} \subset S_p$ predictions where the IoU with $D_i \geq$ t
  C_p.remove(C_{iou}) with $C_{iou} \subset C_p$ scores of predictions where the IoU with $D_i \geq$ t
end

In layman's terms the proposal with highest confidence score is selected. The IoU with remaining proposals are calculated. If the IoU exceeds a given threshold, the overlapping less confident predictions are removed. This is repeated until all there are no proposals left in the initial set.

5. Implement NMS with a threshold of 0.1 to reduce the number of predictions per frame. The IoU should be calculated only on the BEV.
Report: 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?
Hint: If you want to reuse your 3D IoU calculation function from task 1 for the 2D IoU calculation, you can set the (y, h) to (0, 1).
Note: Implement this task in `utils/task5.py/nms()`. You can and *should* reuse your already implemented functions from task 1.
Test: You can test your implementation by either executing `python tests/test.py`

`--task 5` to run on your local machine or by following the instructions in the README.md of the code template to run on a CPU node on AWS. Make sure you successfully pass the test before starting to train if you want to get the correct visualisation samples.

We have our baseline and now we can train! In the evaluation utilities script, we provide a method to calculate the mean average precision (mAP) which is a crucial metric in 3D object detection. You do not have to alter the file as this utilizes your implementations from task 1 and is baked into the training loop. Careful, this is not the mean of the precision but the area under the precision-recall curve. For a more detailed explanation on the mAP metric we highly recommend the following blog post[5].

The resulting predictions from our network are evaluated on three ground truth bounding box difficulty levels: easy, moderate, hard (determined based on distance, truncation and occlusion). The leaderboard is ordered by the moderate difficulty mAP.

6. Train the network.
   Report: In your report, document the loss- and precision curves along with 3 example scene visualizations from the beginning, mid-way and end of the training cycle. Submit the generated `submission.zip` to the Codalab leaderboard to verify your implementation and report your test metrics.
   Note: The leaderboard standings for this problem have no impact on the grading. While we allow for up to 5 submission, one should suffice. The test set evaluation is there such that you (and we) can verify your implementation.
   Expected results (see qwang): 83.99 (Easy), **74.14** (Moderate), 72.70 (Hard) %mAP
   For training configurations please refer to `config.yaml`. You do not need to identically match the expected results as they may vary due to random initialization among other factors.

Problem 2. Taking Matters Into Your Own Hands (5+2 point(s))

Now that we have established a baseline method and can successfully detect vehicles directly from LiDAR point clouds it is time to tackle some of the shortcomings of the second stage refinement network and therefore improve its performance.

1. Propose a method to improve the network performance. For this problem you are free to come up with a novel idea, or implement and analyse a method from the current literature to improve the networks performance or its speed.

Report: Hand in your report as a PDF with a maximum of 6 pages (additional pages can be used for listing references). The final report should contain an accurate and complete description of your solution following the given formatting of the provided LaTeX template (in gitlab). If you have implemented changes to the training procedure, you must include a README file explaining how to run your code and reproduce your experiments.
Grading: You will be graded based on the following:

> **Quality of your paper:** Background, method, and experiments are clear. The paper is well written and the visuals are interesting. Scores and baselines are well documented.
> **The execution of idea:** The experiments done make sense in order to tackle the proposed issues of the Problem 1 baseline given the proposed method. There are no obvious experiments not done that could greatly increase clarity. The submitted code and other supplementary material is understandable, commented, complete, clean and there is a README file that explains it and describes how to reproduce your results if necessary.

Furthermore you can receive up to **+2 bonus** points from (1) the leaderboard standings on Codalab based on a linear interpolation (0.25 increments) from the highest performing network

[5] https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

to our first stage benchmark based on the moderate difficulty mAP or (2) from the novelty of your contribution. The bonus points will be taken as the maximum of (1) and (2).
Note: As it should be obvious, the performance gains of the proposed method do not play a role in the grading of this problem but only on the possible bonus points that can be acquired from the leaderboard. What is important is correct argumentation with sufficient experimental backing.
Tip: Make sure you run ablation studies on your components! Isolate their effects in the performance of the network. Try and visually show us what improvements are made after introducing a module to your system. You can use your visualization tools from Project 1.
Tip: You are on a strict budget. Plan your runs before submitting them. You can use AWS CPU instances or Google Colab to debug your code in order to reduce your AWS expenses.

References

[1] Li, Z., Yao, Y., Quan, Z., Yang, W., Xie, J.: Sienet: Spatial information enhancement network for 3d object detection from point cloud. arXiv preprint arXiv:2103.15396 (2021)

[2] Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., Li, H.: Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10529–10538 (2020)

[3] Shi, S., Wang, X., Li, H.: Pointrcnn: 3d object proposal generation and detection from point cloud. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 770–779 (2019)