3D Car Object Detection in KITTI Benchmark Dataset

Sushmitha Belede [1] **Monika Sri Vyshnavi Nagalla** [2]

Abstract

In this work we study the 3D object detection problem for autonomous vehicle navigation. We seek to understand the Frustum PointNets architecture and experiment with architectural improvements to measure their effect on performance metrics in the KITTI benchmark dataset. Then, we compare how changing three different parts of the Frustum PointNets architecture (the CNN block, the 3D Instance Segmentation block and the 3D Amodal BoxEstimation block) changes the overall performance of the model. We observed that adding two Instance Segmentation PointNet layers instead of one improved the final performance of the model on sparse and occluded objects. We also study incremental changes in a truncated training cycle, which indicate that dropout layers added after convolution layers in the instance segmentation network improved model performance.

1. Introduction

Problem before solution: In the Deep Learning community significant advances have been made in 2D perceptions tasks applied to autonomous vehicles, such as object detection, classification, and image segmentation (He et al., 2017) (Pendleton et al., 2017). However, as autonomous vehicles are being manufactured and outfitted with multiple sensor modalities such as RBG cameras and LIDAR systems, the research community has shifted attention to processing 3D data for perception tasks.

Our work is focused on one of the most significant to autonomous vehicle navigation tasks: 3D object detection. This task is consists of two components: (1) a regression step that encompasses estimating the location and orientation of 3D bounding boxes representative of physical objects in the 3D road data; and (2) a classification step that identifies the object inside the bounding box.

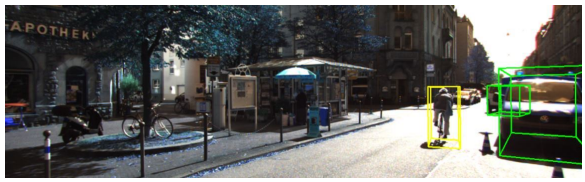

Figure 1. Example 3D object detection from KITTI dataset

Existing methods for 3D object detection have utilized the camera and LiDAR modalities independently. Only recently have methods been proposed that fuse the two data sources together. One such architecture, Frustum PointNets (Qi et al., 2018), operates on RGB-D data, which is a combination of LiDAR data and RGB images. Our focus is this architecture and the 2D detector used for instance segmentation.

2. Related Work

Existing state-of-the-art methods for 3D object detection work on point clouds, and RGB-D fused data.

Point Cloud Methods: Most existing work on LiDAR point clouds converts the raw point cloud into a volumetric occupancy grid, which differentiates between occupied, unknown and free space, prior to learning. (Maturana & Scherer, 2015) voxelize point clouds and use 3D CNNs to perform object detection. (Qi et al., 2017a) and (Qi et al., 2017b) create a network architecture (PointNet) that directly operates on the raw point clouds for single object classification and semantic segmentation. Frustum PointNets (Qi et al., 2018) extends the PointNets architecture to perform 3D instance segmentation in the 3D object detection network.

RGB-D methods: Researchers have varied their approach in how RBG-D data is represented as inputs to their networks. MV3D (Chen et al., 2017) takes image and LiDAR point cloud data, which is used as a front view as well as projected into a bird's eye view. The first step is to generates 3D object proposals from bird's eye view map and project them to three views. A fusion network combines region based features, from ROI pooling, for each

[1]Robotics Engineering Department, Worcester Polytechnic Institute, Worcester, MA [2]**AUTHORERR: Missing \icmlaffiliation.** . Correspondence to: Author <@wpi.edu>.

CS/DS 541 Deep Learning Final Project, Worcester Polytechnic Institute.

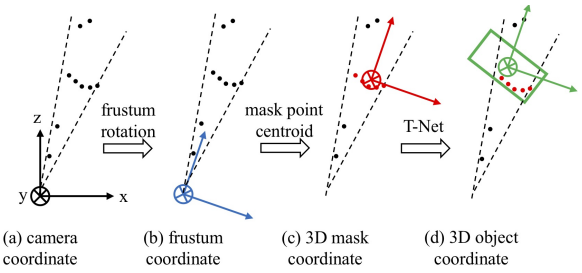

Figure 2. Translation of the frame of reference from the camera to the object (Qi et al., 2018)

view. Then 3D bounding boxes are drawn via a region proposal network (RPN) (Ren et al., 2015) and the fused features are used to predict the object class. This method has been shown to struggle in detecting smaller objects such as pedestrians compared to larger ones such as cars.

Training Frustum PointNets first uses a 2D CNN designed for object detection to detect, bound, and classify objects within 2D space. The classification is one-hot encoded and used alongside the derived 2D bounding box to derive what is called a "Frustum proposal" for 3D image segmentation. During this process, the frame of reference is shifted from the camera to the object (Figure 2).

Next, for each point cloud inside a frustum (n × c with n points and c channels of XYZ, intensity etc. for each point), the object is segmented via binary classification of every point using what it knows about that object class. The segmented point cloud is put into a regression PointNets(referred to as T-Net) to align the points so that their centroid is as close as possible to the 3D bounding box center. Finally, the last network estimates where to place the 3D bounding box.

3. Proposed Method

We sought to understand the original Frustum PointNets architecture and methods. Frustum PointNets was selected because of its state-of-the-art performance on the KITTI benchmark dataset. After analyzing the original (Qi et al., 2018) TensorFlow implementation, we choose to work with a newer PyTorch implementation by (Gustafsson & Linder-Norén, 2018) as a base architecture since it allowed for integration with newer 2D object detection models we intended to experiment with as replacements for the original model used within the Frustum PointNets architecture.

Note that the Frustum PointNets architecture requires 2D bounding box labels as one of the inputs for 3D segmentation. The original Frustum PointNets use ground-truth 2D bounding box labels for training for the 3D image segmentation and bounding box regression. However, for validation or testing 2D bounding box labels from the 2D object detector are used. The original Frustum PointNets uses Faster R-CNN (Girshick, 2015) as a 2D object detection network (for validation and testing only).

We compared popular 2D object detection models such as YOLOv3 (Redmon & Farhadi, 2018), MobileNet, and EfficientNet (Tan & Le, 2019). We chose to replace Faster R-CNN with EfficientNet. The sharp reduction in trainable parameters in EfficientNet motivated our choice, as we saw it as a way to potentially mitigate Frustum PointNets' notoriously long training cycle, which can be partially attributed to a large number of trainable parameters. We will refer to this model with EfficientNet as the ED+Frustum model. Similar to the original implementation, we implemented EfficientNet to calculate 2D bounding box labels in the ED+Frustum model. We used the original Frustum PointNets network which estimates the 3D bounding box pre-trained on the ground truth 2D bounding box labels for training. For validation though, we used the 2D bounding box labels we calculated from EfficientNet.

The original Frustum PointNets (Qi et al., 2018) use PointNet for object detection inside the frustum proposal in the 3D instance segmentation architecture. This part of the network, classifies every point inside the as a car or a non-car. Note that the frustum proposals, which are inputs to the 3D Instance Segmentation class, are sparse point-clouds (only about 1000–2000 nodes in the point cloud of the frustum proposal). PointNet++ is proved to be a sparsity aware network which uses nested PointNet recursively. In the original Frustum PointNets paper they also replaced the PointNet layers with PointNet++ layers and observed better results.

In our second proposed model, which we will refer to as PointNet++FPN model (Qi et al., 2017b), we implemented a nested 3D instance segmentation architecture analogous to the PointNet++ architecture in the original paper. Our nested 3D instance segmentation architecture takes 1024 points from the frustum proposal, down-samples and up-samples the point cloud nodes from 1024 to 512 to 1024 to 256 to 1024.The idea behind this is that the repeated down-sampling and upsampling helps in extracting hierarchical features. However, there is a downside to this process, it increases the number of trainable parameters.

We sought to understand how the changes in the convolution and fully connected layers of the T-Net and amodal 3D box estimator layers effects the performance of our 3D object detector. Hence we implemented a third change to the original Frustum PointNets model (Qi et al., 2018), by removing a convolution layer and adding fully connected layers in the T-Net and amodal 3D box Estimators. We will refer to this model as the Frustum+FCN model.

As an alternative approach, we used a different, publicly

Table 1. **Truncated Training Results** KITTI Full Validation Dataset (cars only)

Model	Segmentation Accuracy (%)	Bounding Box IoU	Box Estimation Accuracy (%)
Frustum baseline[1]	87.29	0.715214/0.656532	56.48
(A.) Droput added in ISeg	87.36	0.714380/0.651353	55.63
(B.) Baseline SGD+Momentum	83.78	0.671862/0.618743	53.75
(C.) Dropout in Iseg and SGD+Momentum	85.66	0.693844/0.636761	53.75

[1] (simon3dv, 2020)

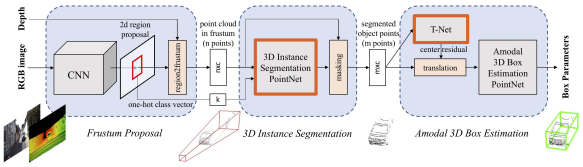

Figure 3. The Gustafsson implementation removed batch normalization layers (were indicated by the orange boxes) from the original Frustum PointNets architecture. (Qi et al., 2018)

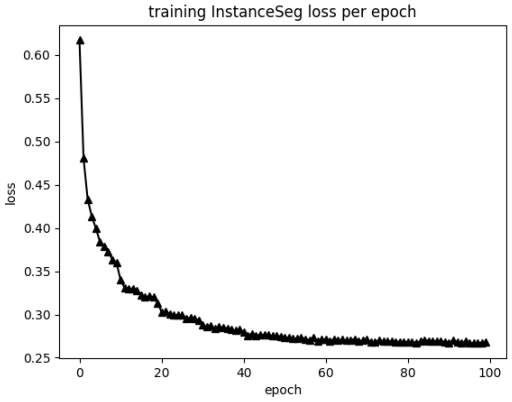

Figure 4. Training losses after the Instance Segmentation layer for the FrustumS model (with PointNet in the Instance Segmentation Layer).

available implementation (simon3dv, 2020) of Frustum PointNets (Qi et al., 2018) to experiment with incremental changes in the architecture over a small number of epochs on the full KITTI training and validation sets. We will refer to this baseline existing implementation as "simon3dv".

4. Experiments

Our first experiment was to train the Gustafsson implementation of Frustum PointNets on the full KITTI benchmark dataset. We evaluated this model on the validation set (using the same split as in (Qi et al., 2018)) and only classifying objects as cars versus non-cars to allow for a direct comparison of the 3d Object Detection average precision results with existing published results. The Gustafsson implementation (Gustafsson & Linder-Norén, 2018) removes batch normalization layers in the instance segmentation network and T-Net for classification. We kept this architectural change to observe the impact on average precision (AP) compared to the original Frustum PointNets implementation.

We replaced Faster-RCNN, in the Gustafsson implementation of the Frustum PointNets architecture (Qi et al., 2018), with EfficientNet. We used KITTI ground truth to train our EfficientNet in TensorFlow. We then used this trained model to output 2D bounding boxes while validating ED+Frustum on validation dataset. It is to be noted that end to end training of ED+Frustum is not performed.

In order to make and observe incremental changes, we modified the architecture to train on a smaller representative subset of the full KITTI dataset. The full KITTI dataset contains RGB images, 360 100 millisecond LiDAR point clouds, and labels for 7,400 examples. We made a smaller, representative version of the dataset containing 1,000 examples. We will refer to this at the "small KITTI dataset". Due to the computational limitations, we had to restrict to train the proposed models over a smaller data with fewer epochs. To have comparable results of the modified model with the baseline model, the Gustafsson Frustum PointNet, we retrained the baseline model with 1000 datapoints from the original KITTI training dataset and over 100 epochs as opposed to the 700 epochs in the baseline model. Consequentially, the model could not converge effectively and, as presented, the AP values on the full validation set in Table 3 show a huge drop from those in Table 2 with models trained on bigger datasets and more epochs.

In the PointNet++Frustum model, we observe a significant loss on the validation APs. It is also interesting to observe how loss values of the Instance Segmentation layer for the Frustum and PointNet++Frustum values are converging. The loss rate for the PointNet++Frustum model is higher towards the final epochs as opposed to that of the Frustum model. Evidently, if the PointNet++Frustum is trained more the loss is expected to further decrease. This proves that the proposed change in architecture in the InstanceSeg layer is promising.

We trained the Frustum+FCN on 1000 datapoints on 200

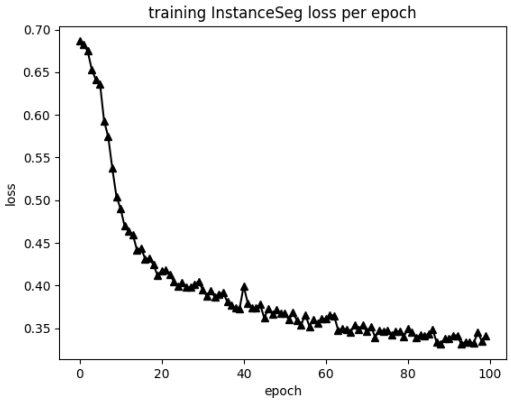

Figure 5. Training losses after the Instance Segmentation layer for the PointNet++FPN (with PointNet++ in the Instance Segmentation Layer).

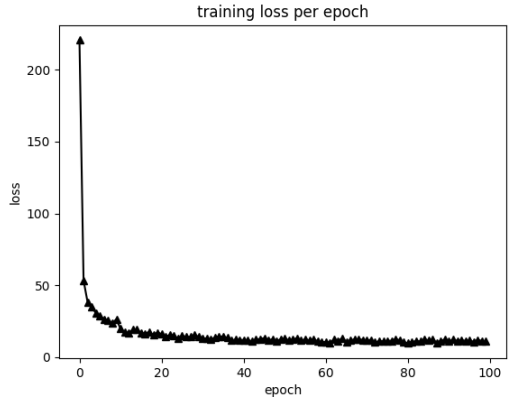

Figure 6. Training losses for the FrustumS model.

epochs. The final training loss values for FrustumS and Frustum+FCN are presented in the Figures 6 and 7. In the Frustum+FCN, there are a greater number of training parameters than in the original model, thus it needs more training than the original model for the loss to converge better, this is evident as seen from Figure 12, that the FrustumS model started converging at 100 epochs while Frustum+FCN model did not converge as much at 200 epochs too. With the results obtained, we can propose that if we train the Frustum+FCN model for more number of epochs and with more data, chances for the model to perform better can be improved.

We conducted the next series of experiments on the simon3dv implementation. Since this implementation is an exact PyTorch implementation of the Qi et al. Frustum PointNets we used it as the baseline model. Due to time and computational power constraints we limited our experiments to only five epochs, each with a batch size of 16 ex-

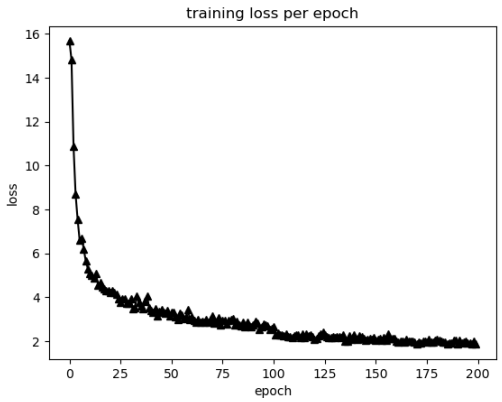

Figure 7. Training losses for the Frustum+FCN (with more FCN layers in the T-Net and BBbox Net layers).

amples. Our first step was to run the simon3dv implementation and report validation set results (as inspired by the results found in (Qi et al., 2018)) on the five epochs as way means of establishing a comparable baseline for our small-scale experiments. We then tested three different model configurations: (model A) the baseline implementation using Stochastic Gradient Descent (SGD) with momentum instead of the Adam optimizer; (model B) the baseline implementation extended to include a dropout layer after every convolution layer in the instance segmentation network; and (model C) the baseline implementation with dropout after every convolution in the instance segmentation network with SGD and momentum for optimization.

5. Results

The results of training and evaluating the Gustafsson implementation shows that the removal of batch normalization produces a slightly higher AP on the KITTI full validation set compared to that of the original implementation (Qi et al., 2018). This indicates that the extra regularization may not be needed for the network to generalize to unseen data. The AP percentages for the easy, moderate, and hard detection tasks can be seen in Table 2. We used the easy, moderate, and hard task description as provided by the KITTI dataset (Geiger et al., 2013).

Table 2. **3D Object Detection Average Precision** KITTI Full Validation Dataset (cars only)

Method	Easy	Moderate	Hard
Frustum [1]	83.76	70.92	63.65
Modified Frustum [2]	**84.89**	**73.61**	**70.32**
ED+Frustum	**81.34**	**60.67**	**58.06**

[1] (Qi et al., 2018) [2] (Gustafsson & Linder-Norén, 2018)

In Table 2, we can compare the results of ED+Frustum against the Frustum and Modified Frustum models. Though the results of the ED+Frustum model did not deteriorate much on the Easy dataset, we observe worse performance on the Moderate and Hard datasets. We hypothesize that training EfficientNet over a greater number of epochs would result in better 2D labels and eventually improve the AP for 3D object detection.

Table 3. **3D Object Detection Average Precision** KITTI Full Validation Dataset (cars only)

Method	Easy	Moderate	Hard
FrustumS [1]	36.34	31.54	29.6
PointNet++Frustum	**26.72**	**23.59**	**25.83**
Frustum+FCN	**32.12**	**29.02**	**24.8**

[1] (Qi et al., 2018) [2] (Gustafsson & Linder-Norén, 2018)

As seen in Table 3, we compared the performance of the proposed PointNet++Frustum and Frustum+FCN models on the full validation dataset. As expected, the performance of the baseline along with the proposed models on the full KITTI validation set is much worse. However, the most important result we observed from the PointNet++Frustum is that it performed very well on the Hard dataset, better than it performed the moderate dataset. As expected, the modified PointNet++ could extract better features with the advantage of hierarchical feature extraction.

The Frustum+FCN model performed better on the Easy and Moderate data-sets than the PointNet++Frustum model. One reason for this is that it the former model was trained for 100 more epochs than the latter. Nevertheless, the PointNet++Frustum performed better on the Hard dataset.

The result of our alternative approach, running a truncated training cycle, on the full KITTI dataset can be seen in Table 1. Following convention, we report results on the validation set (using the same split as in (Qi et al., 2018)) for cars only. We report Segmentation Accuracy, Bounding Box IoU, and Box Estimation Accuracy for five training epochs for the baseline model, and models 1—3 are changes with which we experimented. Segmentation accuracy represents the accuracy of the instance segmentation network in segmenting point cloud encoded objects. Bounding Box Intersection over Union (Bounding Box IoU), is an evaluation metric is a measurement of accuracy for 3d object detectors; it formulates the area of overlap between the predicted 3D box and ground truth box divided by the area of their unions. Box estimation accuracy represents the accuracy of the position of the 3D bounding boxes.

In Table 1 we can see that only our model A marginally improved segmentation accuracy; while the IoU and box estimation accuracies were not improved by any of the models, it is interesting to note that the evaluation metrics were not significantly degraded with our changes. Looking at model A, it appears that dropout may offer improvement in these metrics if training time was extended. The addition of dropout to the SGD optimizer with momentum (model C) also demonstrated promising results, indicating that dropout helped improve the performance of the SGD model with momentum (model B) in terms of the segmentation accuracy compared to just using the SGD model without dropout in the instance segmentation. Overall, model C reported better evaluation metrics than model B.

6. Discussion

It is often said that "the whole is only as good as the sum of its part", and this is applicable to fusion models that are constructed upon the shoulders of previously trained models. Any fault within a single piece is amplified, particularly if computational limitations reduce ability to compensate with a large set of training data. Integration of models is limited by a lack of modularity or input/output standardization. For example, substituting one 2D object detection model for another within Frustum PointNets was a non-trivial task.

Current implementations of Frustum PointNets only classify a limited number of objects. Furthermore, object of mixed types are not included; for example, pedestrians pushing baby carriages.

7. Conclusions and Future Work

In this project we attempted to understand the 3D Object Detection pipeline and architecture in the Frustum PointNets fusion network and implement modifications in order to observe their effect on the model performance. We explored changes to the 2D Object Detector, Instance Segmentation, and T-Net networks future works could consider alternatives to the T-Net (PointNet) as whole such as Shufflenet (Zhang et al., 2018). In the PointNet++Frustum and Frustum+FCN models, we independently changed the architectures within the Instance Segmentation and 3D Amodal Box estimation layers. We can further explore combining different changes into a single model and observe the results. We can also tune hyper-parameters in different layers to suit the changed architectures in each layers. Future work could consider extending the training period from the truncated training experiments to analyze what we hypothesize are positive results from the inclusion of dropout between convolution layers in the instance segmentation network.

References

Chen, Xiaozhi, Ma, Huimin, Wan, Ji, Li, Bo, and Xia, Tian. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1907–1915, 2017.

Geiger, Andreas, Lenz, Philip, Stiller, Christoph, and Urtasun, Raquel. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11): 1231–1237, 2013.

Girshick, Ross. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

Gustafsson, Fredrik and Linder-Norén, Erik. Automotive 3d object detection without target domain annotations, 2018.

He, Kaiming, Gkioxari, Georgia, Dollár, Piotr, and Girshick, Ross. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

Maturana, Daniel and Scherer, Sebastian. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928. IEEE, 2015.

Pendleton, Scott Drew, Andersen, Hans, Du, Xinxin, Shen, Xiaotong, Meghjani, Malika, Eng, You Hong, Rus, Daniela, and Ang, Marcelo H. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1):6, 2017.

Qi, Charles R, Su, Hao, Mo, Kaichun, and Guibas, Leonidas J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017a.

Qi, Charles R, Yi, Li, Su, Hao, and Guibas, Leonidas J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017b.

Qi, Charles R, Liu, Wei, Wu, Chenxia, Su, Hao, and Guibas, Leonidas J. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 918–927, 2018.

Redmon, Joseph and Farhadi, Ali. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

Ren, Shaoqing, He, Kaiming, Girshick, Ross, and Sun, Jian. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

simon3dv. simon3dv/frustum_pointnets_pytorch, May 2020. URL https://github.com/simon3dv/frustum_pointnets_pytorch. original-date: 2020-01-16T14:04:45Z.

Tan, Mingxing and Le, Quoc. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Zhang, Xiangyu, Zhou, Xinyu, Lin, Mengxiao, and Sun, Jian. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.