Evaluation of 3D Object Detection for Cars by improving existing frameworks: CS 4803/7643 Spring 2020

Sanmeshkumar Udhayakumar
sudhayakumar3@gatech.edu

Tarushree Gandhi
tgandhi9@gatech.edu

Leon Carter Price
lprice8@gatech.edu

Disha Das
ddas71@gatech.edu

Georgia Institute of Technology,
777 Atlantic Dr. N. W., Atlanta, GA 30332-0355, USA;

Abstract

In this work, we tackle the problem of 3D object detection of cars on the Kitti dataset. We approach this problem through two parallel deep learning frameworks, PointRCNN, and Frustum ConvNet, and we try to improve the performance from the default versions of these models through a variety of experiments. Overall, we were able to achieve higher performance in the medium and hard 3D AP (Average Precision) categories than the default models for PointRCNN and Frustum ConvNet. When comparing our two best models from the two respective deep learning frameworks, while our best Frustum ConvNet model performed better in the Easy Category, our best PointRCNN model performed better in the Medium and Hard 3D AP Categories.

1. Introduction

LiDAR (Light detection and ranging) technology allows precise 3D measurement data over short and long ranges inspite of weather and lighting conditions. A LiDAR gathers point clouds as data, which are a collection of points that represent a 3D shape or feature. 3D object detection from point cloud data poses a fundamental challenge in the field of autonomous driving. In this report, we attempt two different techniques of 3D object detection, Point-RCNN [9] and Frustum ConvNet [10] and compare their performance.

Existing 3D detection techniques mostly employ either grid or point based methods. In grid-based methods, the irregular point clouds are transformed to structured representations which can then be processed by CNNs to obtain point features for 3D object detection. On the other hand, point-based methods directly extract discriminative features. The state of the art framework known as PointVoxel-RCNN [8], integrates both of these methods. Currently, the precision value for the car class of KITTI dataset [1] stands at 90.25, 81.43 and 76.82 on easy, moderate and hard difficulty levels. This is a baseline that can be improved upon.

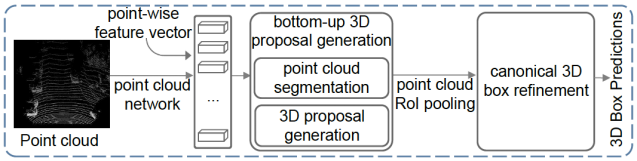

Figure 1. PointRCNN 2-stage detector

For safe and reliable driving, an autonomous vehicle has to accurately perceive and interpret the surrounding environment. The experiments presented in this report is an effort to improve upon the baseline object classification accuracy.

For data we used the KITTI data set [1], this data set was created for tasks such as stereo, optical flow, visual odometry, 3D object detection and 3D tracking. We used the 3D object detection data set which includes 2D RGB images with a corresponding 3D point cloud from a LIDAR scan, ground truth labels of the 3D bounding boxes, and camera calibration data.

2. Approach

2.1. Methodology

For this project, we have conducted a set of experiments with two well-known 3d object detection networks - PointRCNN [9] and Frustum ConvNet [10]. A subset of KITTI 3D object detection dataset has been used to be able to perform as many experiments as possible (owing to the limited time/hardware resources).

The main idea behind this project is to get exposure to 3D Object Detection methods by trying out some of the existing techniques. Our approach is to train, test and validate the current models, followed by experimenting with some of the components of the code to understand and analyse the role of various aspects of Deep Learning frameworks.

Both PointRCNN [9] and Frustum ConvNet [10] research papers were published in early/mid 2019. They stand

at 45th and 50th rank on The KITTI Vision Benchmark Suite for 3D object detection (Car) respectively.The reason behind choosing these two frameworks is to be able to get a flavor of comparatively conventional, yet efficient approaches as compared to the current state of the art.

In our experiments, we attempted at improving the current accuracy of the networks by trying out various techniques such as hyper-parameter tuning, loss function modification, employing different optimization techniques and tweaking other parameters that are specific to the respective networks. At the end, visualizing the outputs helped us validate our results.

Some of the experiments showed increase in the current accuracy, while some did not do well. A few of the experiments gave positive results that proved our underlying intuition to be correct. There were a few errors regarding cuDNN, mismatched tensor shapes, inability to use pretrained model for Transfer Learning etc. that were eventually dealt with by help from the internet.

In-depth study of the code has helped us appreciate the role of various aspects of the pipeline and better understand the flow of the research papers.

2.2. Network Architectures

We chose PointRCNN and Frustum-ConvNet to solve the problem of 3-D object detection. These two methods are some of the best performing methods for 3D Object detection. PointRCNN is known to achieve best accuracy for 'Cyclist' class. However, all our experiments have been done only on 'Car' class as KITTI datset has maximum samples of cars. Also, these method can easily be adapted to other classes.

2.2.1 PointRCNN

PointRCNN architecture [9] is composed of 2 stages : proposal generation and proposal refinement to get the final detection results.(See Fig 1)

Stage I : Bottom-up proposal generation (RPN)
Stage I is Region Proposal Network. PointRCNN works directly on the raw point cloud data to generate box proposals. Instead of using 3D anchor-based approach, this method significantly limits the search space by employing a bottom up approach. By learning point-wise features to segment points into foreground and background and generating 3D proposals from the segmented foreground points simultaneously, high-recall proposal are generated.

3D box proposal and foreground segmentation are performed simultaneously by appending 2 heads to the point features.

Segmentation(Classification) Loss
There are significantly less foreground points compared to the background points. Therefore, Focal Loss [3] has been used to tackle the class-imbalance problem.

$$L_{focal}(p_t) = -\alpha_t(1-p_t)^{\gamma} log(p_t), \quad (1)$$
$$\text{where } p_t = \begin{cases} p \text{ for foreground point} \\ 1-p \text{ otherwise} \end{cases}$$

Regression(Localization) Loss
A 3D bounding box is represented by (x,y,z,h,w,l,θ). In order to generate bounding boxes using foreground points, the surrounding region of each foreground point up to a search distance S is divided into a number of bins along X and Z axis to estimate the bounding box center. Loss terms consists of 2 terms each for bin classification along X,Z and regression within the classified bin.

$$\mathcal{L}_{bin}^{(p)} = \sum_{u \in \{x,z,\theta\}} (\mathcal{F}_{cls}(\widehat{\text{bin}}_u^{(p)}, \text{bin}_u^{(p)}) + \mathcal{F}_{reg}(\widehat{\text{res}}_u^{(p)}, \text{res}_u^{(p)})),$$
$$\mathcal{L}_{res}^{(p)} = \sum_{v \in \{y,h,w,l\}} \mathcal{F}_{reg}(\widehat{\text{res}}_v^{(p)}, \text{res}_v^{(p)}), \quad (2)$$
$$\mathcal{L}_{reg} = \frac{1}{N_{\text{pos}}} \sum_{p \in \text{pos}} (\mathcal{L}_{bin}^{(p)} + \mathcal{L}_{res}^{(p)})$$

Stage II : Proposal Refinement (RCNN)
Stage II is Region Convolutional Neural Network. The second stage uses robust box proposals and point-features from Stage I to perform region pooling on learned representations. Pooled points are transformed to canonical coordinate system and these local spatial features are combined with semantic features for bounding box refinement. The second-stage loss is given by:

$$\mathcal{L}_{\text{refine}} = \frac{1}{\|\mathcal{B}\|} \sum_{i \in \mathcal{B}} \mathcal{F}_{\text{cls}}(\text{prob}_i, \text{label}_i) + \frac{1}{\|\mathcal{B}_{pos}\|} \sum_{i \in \mathcal{B}_{pos}} (\tilde{\mathcal{L}}_{\text{bin}}^{(i)} + \tilde{\mathcal{L}}_{\text{res}}^{(i)}) \quad (3)$$

Both the stages use PointNet++ [6] as the backbone architecture to get discriminative point-features from the point cloud and features for confidence classification and box refinement respectively. Non-maximum suppression (NMS) is used in the both the stages for refining the proposals as well as bounding box results.

The code for PointRCNN is available at https://github.com/sshaoshuai/PointRCNN

2.2.2 Frustum ConvNet

The Frustrum ConvNet [10] utilizes the 2D images in addition to the point cloud itself. The 2D images are used to generate region proposals. These region proposals are

then run through PointNet [5] to generate a feature vector of length 128 for each region proposal. These feature vectors are then concatenated to generate a grid and passed through a fully connected convolution network to predict class and 3D bounding box location.

Region Proposals
The region proposals were generated by applying the 2D object detector utilized in the [4], which uses FPN [2] for region proposals and then FasterRCNN [7] for final bounding box. This 2D bounding box is then projected into the point cloud to generate a frustum of points. This frustum is then further sub-divided using a predefined stride s and height u. This generates a series of smaller overlapping point clouds contained within the projected frustum.

Feature Generation and FCN
Each of the proposal point clouds are then fed into PointNet [5] and generates a feature vector of length 128. Theses feature vectors are then combined by forming a 2D grid (L x 128). This grid is then fed into a Fully convectional Network (FCN). The FCN contains 4 convolution layers with Batch Normalization and Relu and 3 De-convolution layers for the Kitti data set. The entire process is repeated for different values of s and u, so that the region proposals can be at different levels of resolution.

Loss
Out of the FCN, there are two parallel convolution layers; one for classification and the other for regression. The classification head uses focal loss due to the high class imbalance. For predicting the 3D bounding box, the center of the box is estimated along with the length, width, height and angle of the box. These all use a smooth l1 regression loss as well as a corner loss for regularization.

```
# Weighted sum of all losses
loss = cls_loss +
    BOX_LOSS_WEIGHT * (center_loss +
                       heading_class_loss + size_class_loss +
                       HEAD_REG_WEIGHT * heading_res_norm_loss +
                       SIZE_REG_WEIGHT * size_res_norm_loss +
                       CORNER_LOSS_WEIGHT * corners_loss)
```

Figure 2. Loss equation for Frustum ConvNet

cls_loss = classification loss, center_loss = error of predicted box center position, heading_class_loss = error of heading bin, size_class_loss = error in predicting box size class, heading_res_norm_loss and size_res_norm_loss have to do with residual error within the binning of the heading/category classification, and corners_loss equation from Frustum PointNet [4] . Corner Loss is essentially the sum of the distances between ground truth corners and the predicted corners.

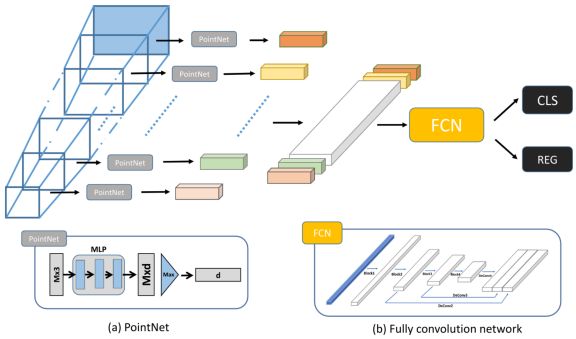

Figure 3. Frustum-ConvNet Achitecture

Final Refinement
After the model has been trained using the model architecture described above, it goes through an additional refinement phase that aims to account for any inaccuracies in the 2D object detector used to create the region proposals. The 2D proposal regions are increased by a factor of 1.2 and then run through a different F-ConvNet.

The code for F-ConvNet is available at https://github.com/zhixinwang/frustum-convnet

3. Experiments and Results

The code is available in Pytorch and all the experiments are conducted on Google Cloud Deep Learning VM with NVIDIA Tesla K80 GPUs, Pytorch 1.4 and Cuda 10.0.

3.1. PointRCNN

Dataset and Implementation Details

Dataset. PointRCNN is trained and evaluated on 3D object detection benchmark of KITTI dataset. The dataset originally contains 7418 training and 7518 testing samples. Original PointRCNN implementation used train/val split by dividing the training samples into train and val sets of sizes 3712 and 3769 each.

As the primary motivation of this project is to understand the underlying aspects of Deep Learning, the dataset was sub-sampled at 1000 (Train), 1000 (Val) and 3000 (Test) images for the purpose of conducting experiments.
Unlike many other approaches, PointRCNN only uses Lidar's point cloud data without using image data to detect 3d objects.

Network Parameters. According to the original implementation of PointRCNN, $16,384$ points are sampled per point cloud sample for training.

PointNet++ [6], which is used as backbone architecture

for both Stage I and Stage II, uses the default parameters. In RPN stage, four abstraction layers are used to sub-sample points into groups of $4096, 1024, 256, 64$. Four feature propagation layers are added on top to get point-wise features for segmentation and box proposals.

In RCNN stage, 512 points are randomly sampled from each proposal and fed to refinement sub-network. Three abstraction layers are used to sub-sample points into groups of $128, 32, 1$ to get feature vector for both confidence classification and localization refinement.

For RPN training, all points within the bounding box are considered 'foreground', while the remaining points are 'background'. In order to avoid any small ground truth box errors, background pixels within $0.2m$ of the box are ignored.

After all the points have been classified into foreground and background, box centers are estimated by using binning approach. A search range of $S = 3m$ with $binsize = 0.5m$ and 12 orientation bins are used for tracing box center around each foreground pixel. NMS with IoU = 0.85 threshold is used to refine proposals and top 300 proposals are sent forward to RCNN stage. In the second stage, IoU above 0.6 between a proposal and ground truth box is considered a positive sample, while IoU less than 0.45 is used as a negative example for the Classification head. IoU = 0.55 is threshold for training the Regression head.

For bin-based proposal refinement, a search range of $S = 1.5m$ with $binsize = 0.5m$ and orientation bin size = $10\,\text{deg}$ is used.

Training parameters. All trainings are done on 'Car' class. The two sub-networks : RPN and RCNN and trained separately. In the original implementation, RPN is trained for 200 epochs, with batch size = 16 and learning rate = $2e-3$, while RCNN is trained for 50 epochs with batch size = 256 and learning rate = $2e-3$.
In our implementation, RPN training configuration is 80 epochs, 32 batch size and $2e-3$ LR, RCNN training configuration is 30 epochs, 64 batch size and $2e-3$ learning rate, due to smaller sub-sampled dataset.
Adam optimizer is used. No LR decay is used in our experiments. Momentum values are 0.95 and 0.85.

Data Augmentation. Code enables data augmentation by performing scaling, flipping and rotation. New ground-truth boxes are also added by using object simulation. Also, slight variations to proposals are done to augment bounding box proposals from stage I.

Network	IoU				
	0.1	0.3	0.5	0.7	0.9
Box Proposal	0.95	0.94	0.92	0.69	0.003
Box Refinement	0.95	0.94	0.92	0.82	0.15

Table 1. Recall of PointRCNN pre-trained model on val set

3.1.1 PointRCNN Experiments

In all our experiments, the numbers are reported on 'val' split because KITTI doesn not provide ground truth for 'test' split.

The final pre-trained PointRCNN model for 'Car' class is provided on the code Github page. Results are evaluated at different Intersection over Union (IoU) values. Threshold of 10%, 30%, 50%, 70% and 90% are used to measure the success.

In order to verify the efficiency of this model, we first test it on our new validation dataset. The model gives good recall value for both ROI box proposals and final bounding box results. The results are presented in Tab. 1

We start by training RPN network to learn the impact of various training parameters.

Region Proposal Network

i) Batch Size
Batch size is one of the most important hyper-parameters to tune as every network has an optimal batch size value which gives the best accuracy. There is a trade-off between using small and large batch-sizes. Large batch size can give more computational speed but can also lead to poor generalization. Whereas, too small batch sizes with a small learning rate can facilitate gradual learning but can also lead to under-fitting due to increased variance (noise).

Hence, we experiment with 3 different values of batch-size-16, 32 and 64 to find the most optimal one for RPN. PointRCNN's original implementation uses a batch-size of 16. The intuition behind this experiment is to try larger batch sizes in order to reduce noise and encourage better generalization.

We trained RPN with these 3 values for 80 epochs, keeping all the other parameters same. The model trained on batch size = 32 gave the best Recall for all the 5 IoU values, while batch size values of 16 and 64 gave almost similar results. The results are presented in Tab. 2.

ii) Focal Loss
Focal Loss [3] was published in 2018 and proved to be very useful for object detection task. It is particularly helpful in cases of heavy class imbalance, where one class has a significantly higher representation than the other. As the Loss function aggregates the loss over all the mis-classified sam-

Batch Size	IoU				
	0.1	0.3	0.5	0.7	0.9
16	0.876	0.859	0.827	0.551	0.002
32	**0.881**	**0.863**	**0.830**	**0.552**	0.002
64	0.878	0.860	0.828	0.551	0.002

Table 2. Recall of RPN with different values of batch size

α (f,b)	IoU				
	0.1	0.3	0.5	0.7	0.9
0.75, 0.25	0.881	0.863	0.830	0.552	**0.002**
0.80, 0.20	**0.902**	**0.883**	**0.843**	**0.556**	0.001
0.85, 0.15	0.880	0.862	0.828	0.546	0.002

Table 3. Recall of RPN with different values of alpha (foreground,background) in Segmentation head's Focal Loss

ples, easily classified negatives can dominate the gradient and overwhelm the rare class.

In our case, foreground points are sparse as compared to background points. Eq.1 shows the equation of Focal Loss built on top of Cross-Entropy loss.

Here, α is a weighting factor for class 1 and 1-α for class -1. α balances the importance of positive/negative samples by giving weight to the rare class. While α balances the importance of positive/negative samples, γ stresses on easy/hard samples. γ is used to down-weight easy samples and focus training on hard negatives.

The default values in original Focal Loss implementation are α = 0.25 and γ = 2 and the original PointRCNN implementation uses these same values. It is important to note that α is acting as a scaling factor that can increase the role of the minority class in the overall loss. It seems as a good idea to weigh the loss according to the class frequency by setting α equal to inverse class frequency. We question the use of α value = 0.25 and experiment with a few different values to test the hypothesis.

The intuition behind this experiment is that the smaller the class, the higher should be it's representation in proportion to its sample size in the dataset. In order to find the ratio of car foreground pixels, we calculate the size of foreground pixels in our train dataset and set α to the inverse proportion (i.e background class's ratio). We find foreground pixels to be around 20%.

We experiment with (foreground,background) α values = (0.75,0.25), (0.80,0.20) and (0.85,0.15). Our hypothesis is proved correct as we get the best Recall values at α = (0.80,0.20). The results are shown in Tab 3.

iii) Search Range and Bin Size

Search Range S is the area around a foreground point where the box center is to be estimated. S distance along X,Z axes is divided up into b bins. PointNet implementation uses S = 3m and b = 0.5m. In order to get a more precise center,

S, b	IoU				
	0.1	0.3	0.5	0.7	0.9
3m, 0.5m	0.902	0.883	0.843	0.556	0.001
2m, 0.2m	**0.905**	**0.885**	**0.851**	**0.563**	**0.002**

Table 4. Recall of RPN with different values of Search range (S) and bin size (b) for Regression head

RCNN Bin size	Network	IoU				
		0.1	0.3	0.5	0.7	0.9
0.5 m	Box Proposal	0.928	0.904	0.864	0.560	0.002
	Box Refinement	**0.929**	0.907	0.887	0.745	**0.083**
0.3 m	Box Proposal	0.928	0.904	0.864	0.560	0.002
	Box Refinement	0.928	0.907	**0.888**	**0.756**	0.079

Table 5. Recall of RCNN with Search range = 1.5 m and varying bin sizes

we experiment with S and b by decreasing the values to 2m and 0.2m respectively. Now there are 10 bins instead of 6 to get more accurate localization. The new S,b showed same recall for lower IoUs and a higher recall for higher IoU values. Thus, proving this experiment to be positive. Results are shown in Tab 4.

Proposal Refinement Network. Using the best RPN model achieved (using batch size = 32, α = (0.8,0.2), search range = 2m and bin size = 0.2m), we then train RCNN by changing α to (0.8,0.2) and keeping scope, bin = 1.5m, 0.5m (same as original). We then changed the bin size to 0.3m to check if changing bin size will give any improvements like seen for RPN. This experiment did not show any significant difference between the two results. The results are shown in Tab. 5.

These experiments resulted in an overall increase in accuracy for Stage I region proposal network as compared to the original model. Using the improved RPN model, we are able to get pretty good numbers on the final model. There is only a 2 to 3 percent difference between our model and the original pre-trained model (see Tab 1.), owing to the fact that we trained our models for comparatively fewer epochs i.e 80,30 versus 200,50 (original code). Also, we trained on a smaller subset.Final results can be seen in Tab. 7.

3.2. Frustum ConvNet

3.2.1 Dataset and Implementation Details

Dataset. Please refer to Dataset section in section 3.1. Unlike PointRCNN however, Frustum ConvNet does use the 2D images.

Network and Feature Parameters. According to the original implementation of Frustum ConvNet, after the 2d region proposal is completed by an off-the-shelf object detector, 4 sets of frustums are generated with a stride s and a half height u_{half} of (0.25,0.5, 1, 2) respectively. Frustums were generated from depth range 0 to 70 m. Within each frustum, 1024 points were sampled to feed into PointNet (num_samples).
For the FCN that takes in the frustum feature map, there are multiple hyper-parameters. The batch size = 32, the max epoch = 50, the base learning rate = 0.001, and the learning rate is multiplied by a decay factor of gamma = 0.1, every 20 epochs (LR_STEPS). The ADAM optimizer was used. There is also a weight decay factor of 0.0001 to protect against over fitting. The loss function mentioned in the loss section of section 2.2.2, has hyper-parameters BOX_LOSS_WEIGHT = 1, SIZE_REG_WEIGHT = 20, HEAD_REG_WEIGHT = 20, and CORNER_LOSS_WEIGHT = 10. The focal loss function that feeds into the cls_loss has an alpha value = 0.25 and gamma = 2. The heading bins are 30 deg in resolution (12 bins).

These same parameters are used for the FCN and loss function in the refinement stage, except the refinement stage takes the first stage bounding box proposals and multiplies then by 1.2 and looks at points only within those boxes, uses 4 sets of stride s and half height u_{half} = (0.1, 0.2, 0.4, 0.8), and randomly samples 512 points from each frustum.

IoU threshold for car classification is set as 0.7 as required by the Kitti Evaluation requirement. All trainings are done on 'Car' class.

To prepare positive and negative samples, the ground truth bounding box is shrunk by a factor of 0.5, and anchor boxes with centers inside the shrunk bounding box are positive samples, while anchor boxes with centers outside the original ground truth are negative samples.
Data Augmentation. Translation and scaling are performed on the 2d region proposals from the off-the-shelf object detector. Also, just as for PointRCNN, 3d points are scaled, flipped and rotated.

3.2.2 Experiments - Frustum ConvNet

Just as for PointRCNN, in all our experiments, the numbers are reported on 'val' split because KITTI doesn't not provide ground truth for 'test' split. The final pre-trained

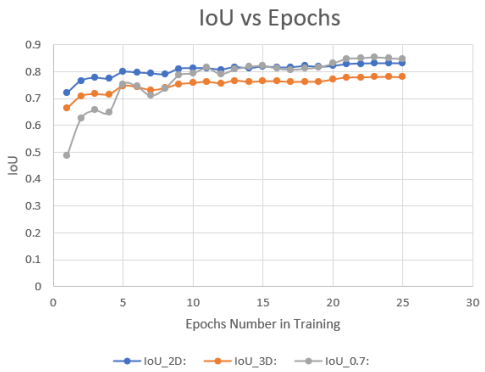

Figure 4. IoU vs Epochs Plot in Training of Frustum using default parameters

F-ConvNet model for 'Car' class is provided on the code Github page. In order to verify the efficiency of this model, we first test it on our new validation dataset.

Complete success was defined as achieving a higher 3D Average Precision than the base model in all difficulty categories, of Easy, Medium and Hard, while partial success was achieving improvement in one of the categories.

For the Frustum ConvNet, we approached improving performance of the baseline model through modifying individual hyper-parameters, and seeing if the 3D AP scores improved. Training/validation was done only using the first state, with the assumption that the the model that performs best in the first stage, will also perform the best in the refinement stage because an initial better estimate will lead to a better refinement. We did this to save training time. We limited the training to 25 epochs because we saw that the IoU didn't increase much past 25 epochs as shown in Fig. 4.

i) Batch Size

The tradeoffs of increasing/decreasing the batch size is covered in the "Batch Size" section of section 3.1.1. We tried doubling the batch size from 32 to 64, and reducing the batch size to 8.

As can be seen from the results Tab. 6, both batch size changes resulted in worse performance, with batch size = 8 having a huge drop in performance, and batch size= 64 having a small drop. Thus this experiment failed as no improvement of AP was achieved in any category. This is a goldilocks parameter, so probably 32 batch size was just the right batch size to get optimal performance.

ii) Frustum height and stride

Optimal frustum size is really dependent on the shape and dimensions of the objects we are trying to identify. Smaller size frustums can probably get finer resolution on

(0.5,1.0)	(1.0,2.0)	(2.0,4.0)	Easy	Moderate	Hard
			84.09	75.32	67.45
✓			84.19	74.88	66.95
	✓		85.41	75.63	67.44
		✓	86.12	76.04	67.97
✓	✓		86.21	76.12	67.96
	✓	✓	86.69	76.30	68.02
✓	✓	✓	86.51	76.57	68.17

Figure 5. Table showing performance of previous frustum combinations that were previously tried.

capturing car points, but will be harder to classify. Bigger size frustums have smaller resolution, but can be classified easier as containing parts of the car.

Table 5 in the original Frustum ConvNet paper [11], which is shown in Fig. 5, suggests having bigger strides and height than the base model values will give higher 3D AP. The original values for s and u_{half} are (0.25, 0.5,1,2), and in the table in Fig. 5, we see that we got better performance with (1,2) over just (0.5,1), which indicated bigger size frustums than the default lead to higher accuracy, so we tried frustums of s and u_{half} = (0.5, 1,2,4) and s and u_{half} = (1, 2,4,8).

As can be seen in the FrustumConvNet results Tab. 6, both frustum stride and height changes lowered the average precision for all difficulty of data by 3+ AP. Thus it seems that this experiment failed as we weren't able to achieve an improvement in AP. This is another goldilocks parameter and this is why the default values of frustum height and stride worked. In the future, we might run more experiments to investigate the optimal frustum size and strides for cars, as it should hopefully be applicable for all car point cloud data.

iii) Focal loss

We tried modifying the focal loss alpha value. The description of this value, along with the intuition behind modifying it is described in the "Focal Loss" section of section 3.1.1. We first reduced alpha value from 0.25 to 0.20, and as can be seen from frustum convNet results Tab. 6, there was a marginal increase in the 3D AP medium result of 0.1, while there was a marginal decrease in the other AP categories. Thus we deem this experiment as a partial success. However, because we didn't see this as a full success, we decided to try a higher alpha value = 0.3 instead.

This resulted in a marginal increase of 1% AP in both medium and hard categories. This indicates that the amount of foreground points in frustums are probably around 30%.

iv) Loss equation weights

We tried changing the loss equation loss weights, by increasing the weights on the classifications that were suffering the most. We found that predicting the correct heading to be the most challenging.

However, in the default loss equation with appropriate weights, we didn't see an emphasis on heading losses, as the magnitude of those losses is about the same as the other losses. Thus we did experiments to increase HEAD_REG_WEIGHT from 20 to 30, and another experiment to increase the multiplication factor of heading_class_loss to 1.5.

Increasing the HEAD_REG_WEIGHT made a marginal change in the 3D AP, with slight decreases in the easy and medium categories, but a slight increase in the hard category. Thus, we achieved partial success.

Changing the Heading_class_loss multiplier from 1 to 1.5 made a marginal decrease in all categories, and thus was a failed experiment.

In the future, we will have to look into further why the performance didn't improve as expected.

Final Experiment:

Overall the only change that improved the AP for at least 2 of the difficulty categories was changing the focal loss alpha to 0.3. Thus for our final experiment, we decided to only change that parameter for both the first stage and the refinement stage.

As you can see from the results Tab. 6, while in the first stage, we didn't have much of a noticeable improvement in AP, in the final refinement stage, we have an improvement in the Medium and Hard categories, with a significant 6% AP increase for the 3D AP Hard category. Thus, our final experiment was able to make a significant improvement for the classification and creation of bounding boxes of the hardest sets of data, which we deem as a huge success.
Tab. 6 is the table of experiment results for the Frustum ConvNet.

3.3. Performance Comparison

Tab. 7 shows a comparison between the performance of the two methods. Frustum ConvNet achieved higher accuracy on Easy samples by a good margin, whereas PointRCNN performed better on Medium ans Hard. This verifies the ranks on KITTI bechmark Suite, where PointRCNN ranks 45th and Frustum ConvNet ranks 50th.

First Stage Comparison 3D AP			
	Easy	Medium	Hard
Default Parameters	83.713112	73.037697	65.761147
Batch Size			
Batch Size = 64	82.642235	72.510872	64.060799
Batch Size = 8	66.371849	60.635315	54.011326
Frustum half height "u_half" and stride "s"			
s and u_half = 1,2,4,8	73.619942	62.302696	54.892056
s and u_half = 0.5,1,2,4	75.024857	65.560303	62.687279
Focal Loss Alpha			
Alpha = 0.2	82.491898	**73.199524**	65.710693
Alpha = 0.3	83.121544	**73.866096**	**66.039665**
Loss equation Weights			
Head_REG Weight = 30	82.722076	72.913101	**65.946609**
Heading_class loss multiplier = 1.5	81.656914	72.092766	64.928650
Final Refinement comparison			
Default Parameters	87.879013	77.236847	68.865837
Our Final Parameters	88.279839	**77.455292**	**74.803123**

Table 6. Frustum ConvNet Experiment Results. Results that performed better than the default parameter values are bolded.

3.4. Qualitative Results

In Fig. 6, we see a comparison of the the predicted bounding boxes for a the two models. This gives a good example of a few false positive results for both models at the farther distances while the cars closer to the LIDAR seem to be more accurately bounded and classified. This suggest that the distance from the LIDAR and the density of the point clouds may play a larger role than is accounted for in the models. More visualizations can be found in the appendix.

4. Conclusion

This work utilized two recent works in 3D object detection, Frustum ConvNets and PointRCNN, to experiment with hyper-parameters and gain a deeper understanding of what elements impact the 3D object detection task. Outside of the typical parameters that have a significant impact in deep learning such as batch size and learning rate, we noticed that the region proposal mechanisms as well as the focal loss α parameter play an important role. For PointRCNN, we saw some significantly positive changes in AP by making changes to the focal loss α, bin size and search range. For F-ConvNet, we saw a significant improvement, relative to our base test, with a focal loss $\alpha = 0.3$.

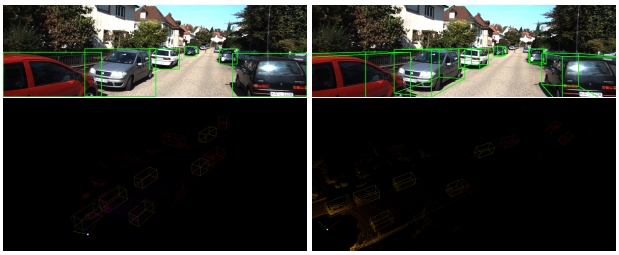

Figure 6. PointRCNN Detection [bottom left] F-ConvNet Detections [Bottom right]. Ground Truth is in green. Predictions in red.

In future, we think that the 3D object detection research community could continue to improve upon the region proposal methodology. For example, with F-ConvNet the frustums are separated into sub-regions to try to capture different scale objects. The sub-region divisions were somewhat arbitrarily determined based on the predefined u and s parameters. Combining the approach of PointRCNN using foreground segmentation with the use of 2D object detector proposal regions seems like a promising research direction; this would likely bring greater focus to the proposal regions. F-ConvNet can also be augmented by exploring different networks beyond PointNet to generate the feature vectors for the FCN. Lastly, the size of these models is generally very large making inference slow; research into decreasing model size and improving inference speed is likely a helpful research direction. This is particularly important for self-driving vehicles which need to make decisions in real time.

5. Work Division

Refer to Tab. 8

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 1

[2] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016. 3

[3] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017. 2, 4

Method	Car AP (IoU = 0.7)								
	Easy			Medium			Hard		
	bbox	bev	3d	bbox	bev	3d	bbox	bev	3d
PointRCNN	89.6	85.7	79.6	**90.1**	87.5	**86.1**	**90.1**	**87.4**	**85.9**
Frustum ConvNet	**98.7**	**90.3**	**88.3**	89.8	**87.8**	79.3	88.3	77.4	74.8

Table 7. Performance comparison of 3D object detection between the two methods on KITTI val split. The evaluation metric is Average Precision(AP) with IoU threshold 0.7 for Car.

Student Name	Contributed Aspects	Details
Sanmeshkumar Udhayakumar	Experiments - Frustum ConvNet Report - Abstract, Frustum Convnet Experiments And Results	Experiments - Frustum ConvNet (all experiments except for focal loss). Report - Frustum Convnet Experiments and Results, Abstract
Leon Price	Experiments - Frustum ConvNet Report - Frustum ConvNet Approach, Visualizations	Experiments - Frustum ConvNet (focal loss). Report - Frustum ConvNet Approach, and Visualizations of 3D and 2D bounding box predictions
Tarushree Gandhi	Experiments - PointRCNN, Report - Methodology, PointRCNN Approach, PointRCNN Experiments and Results	Experiments - PointRCNN (RPN focal loss, search range, bin size, RCNN focal loss and bin-size) Report - Methodology, all PointRCNN sections
Disha Das	Experiments - PointRCNN, Report - Introduction	Experiments - PointRCNN RPN (batch size), Report - Introduction, equations, formatting, references

Table 8. Contributions of team members.

[4] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. *CoRR*, abs/1711.08488, 2017. 3

[5] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016. 3

[6] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5099–5108. Curran Associates, Inc., 2017. 2, 3

[7] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. 3

[8] 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. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 1

[9] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. *CoRR*, abs/1812.04244, 2018. 1, 2

[10] Zhixin Wang and Kui Jia. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. In *IROS*. IEEE, 2019. 1, 2

[11] Zhixin Wang and Kui Jia. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. *CoRR*, 2019. 7

A. Visualizations

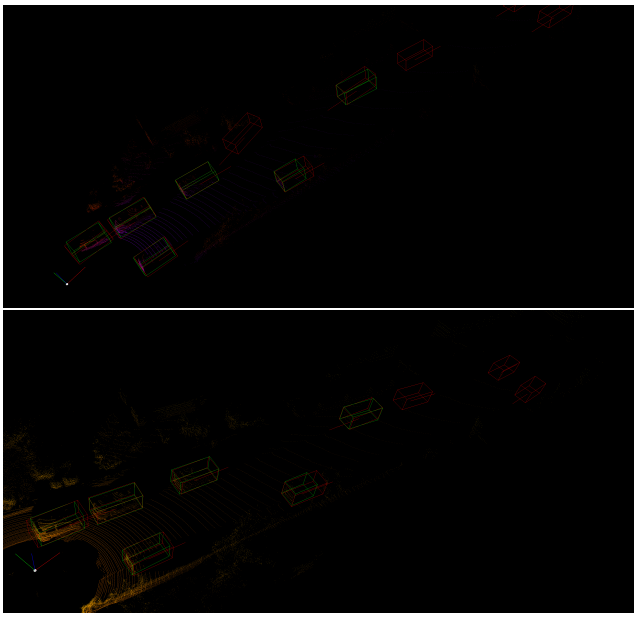

Figure 7. Detection Results for F-ConvNet. First stage results (top) After refinement (bottom)

Fig. 7 gives a great example of how the refinement stage helps to improve the predictions of the F-ConvNet model. The false positive prediction is removed near the middle and the bounding boxes around the cars closest to the LIDAR more closely match the ground truth predictions.

Figure 8. 2D object detection results for F-ConvNet method [bottom]. Ground Truth [top]

In Fig. 8, we see an example of the 2D object detector utilized by F-ConvNet compared with the ground truth label. The bounding box for the truck on the left side of the image is clearly too small for the object detector compared with the ground truth. We also see some additionally unnecessary bounding boxes around the cars farther down the streets. It is clear how theses inaccuracies in the 2D object detector can negatively impact the region proposals.