Frustum PointNets for 3D Object Detection from RGB-D Data

Charles R. Qi[1*] Wei Liu[2] Chenxia Wu[2] Hao Su[3] Leonidas J. Guibas[1]
[1]Stanford University [2]Nuro, Inc. [3]UC San Diego

Abstract

In this work, we study 3D object detection from RGB-D data in both indoor and outdoor scenes. While previous methods focus on images or 3D voxels, often obscuring natural 3D patterns and invariances of 3D data, we directly operate on raw point clouds by popping up RGB-D scans. However, a key challenge of this approach is how to efficiently localize objects in point clouds of large-scale scenes (region proposal). Instead of solely relying on 3D proposals, our method leverages both mature 2D object detectors and advanced 3D deep learning for object localization, achieving efficiency as well as high recall for even small objects. Benefited from learning directly in raw point clouds, our method is also able to precisely estimate 3D bounding boxes even under strong occlusion or with very sparse points. Evaluated on KITTI and SUN RGB-D 3D detection benchmarks, our method outperforms the state of the art by remarkable margins while having real-time capability.

1. Introduction

Recently, great progress has been made on 2D image understanding tasks, such as object detection [13] and instance segmentation [14]. However, beyond getting 2D bounding boxes or pixel masks, *3D understanding* is eagerly in demand in many applications such as autonomous driving and augmented reality (AR). With the popularity of 3D sensors deployed on mobile devices and autonomous vehicles, more and more 3D data is captured and processed. In this work, we study one of the most important 3D perception tasks – 3D object detection, which classifies the object category and estimates *oriented 3D bounding boxes* of physical objects from 3D sensor data.

While 3D sensor data is often in the form of point clouds, how to represent point cloud and what deep net architectures to use for 3D object detection remains an open problem. Most existing works convert 3D point clouds to images by projection [36, 26] or to volumetric grids by quantization [40, 23, 26] and then apply convolutional networks.

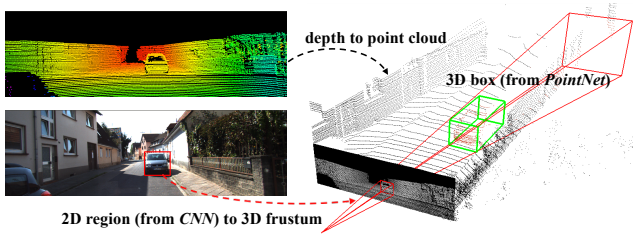

Figure 1. **3D object detection pipeline.** Given RGB-D data, we first generate 2D object region proposals in the RGB image using a CNN. Each 2D region is then extruded to a *3D viewing frustum* in which we get a point cloud from depth data. Finally, our frustum PointNet predicts a (oriented and amodal) 3D bounding box for the object from the points in frustum.

This data representation transformation, however, may obscure natural 3D patterns and invariances of the data. Recently, a number of papers have proposed to process point clouds directly without converting them to other formats. For example, [25, 27] proposed new types of deep net architectures, called *PointNets*, which have shown superior performance and efficiency in several 3D understanding tasks such as object classification and semantic segmentation.

While PointNets are capable of classifying a whole point cloud or predicting a semantic class for each point in a point cloud, it is unclear how this architecture can be used for instance-level 3D object detection. Towards this goal, we have to address one key challenge: how to efficiently propose possible locations of 3D objects in a 3D space. Imitating the practice in image detection, it is straightforward to enumerate candidate 3D boxes by sliding windows [8] or by 3D region proposal networks such as [33]. However, the computational complexity of 3D search typically grows cubically with respect to resolution and becomes too expensive for large scenes or real-time applications such as autonomous driving.

Instead, in this work, we reduce the search space following the dimension reduction principle: we take the advantage of mature 2D object detectors (Fig. 1). First, we extract the 3D bounding frustum of an object by extruding 2D bounding boxes from image detectors. Then, within the 3D space trimmed by each of the 3D frustums, we consecutively perform 3D object instance segmentation and *amodal*

*Majority of the work done as an intern at Nuro, Inc.

3D bounding box regression using two variants of PointNet. The segmentation network predicts the 3D mask of the object of interest (i.e. instance segmentation); and the regression network estimates the amodal 3D bounding box (covering the entire object even if only part of it is visible).

In contrast to previous work that treats RGB-D data as 2D maps for CNNs, our method is more *3D-centric* as we lift depth maps to 3D point clouds and process them using 3D tools. This 3D-centric view enables new capabilities for exploring 3D data in a more effective manner. First, in our pipeline, a few transformations are applied successively on 3D coordinates, which align point clouds into a sequence of more constrained and canonical frames. These alignments factor out pose variations in data, and thus make 3D geometry pattern more evident, leading to an easier job of 3D learners. Second, learning in 3D space can better exploits the geometric and topological structure of 3D space. In principle, all objects live in 3D space; therefore, we believe that many geometric structures, such as repetition, planarity, and symmetry, are more naturally parameterized and captured by learners that directly operate in 3D space. The usefulness of this 3D-centric network design philosophy has been supported by much recent experimental evidence.

Our method achieve leading positions on KITTI 3D object detection [1] and bird's eye view detection [2] benchmarks. Compared with the previous state of the art [6], our method is *8.04%* better on 3D car AP with high efficiency (running at 5 fps). Our method also fits well to indoor RGB-D data where we have achieved *8.9%* and *6.4%* better 3D mAP than [16] and [30] on SUN-RGBD while running one to three orders of magnitude faster.

The key contributions of our work are as follows:

- We propose a novel framework for RGB-D data based 3D object detection called Frustum PointNets.
- We show how we can train 3D object detectors under our framework and achieve state-of-the-art performance on standard 3D object detection benchmarks.
- We provide extensive quantitative evaluations to validate our design choices as well as rich qualitative results for understanding the strengths and limitations of our method.

2. Related Work

3D Object Detection from RGB-D Data Researchers have approached the 3D detection problem by taking various ways to represent RGB-D data.

Front view image based methods: [4, 24, 41] take monocular RGB images and shape priors or occlusion patterns to infer 3D bounding boxes. [18, 7] represent depth data as 2D maps and apply CNNs to localize objects in 2D image. In comparison we represent depth as a point cloud and use advanced 3D deep networks (PointNets) that can exploit 3D geometry more effectively.

Bird's eye view based methods: MV3D [6] projects LiDAR point cloud to bird's eye view and trains a region proposal network (RPN [29]) for 3D bounding box proposal. However, the method lags behind in detecting small objects, such as pedestrians and cyclists and cannot easily adapt to scenes with multiple objects in vertical direction.

3D based methods: [38, 34] train 3D object classifiers by SVMs on hand-designed geometry features extracted from point cloud and then localize objects using sliding-window search. [8] extends [38] by replacing SVM with 3D CNN on voxelized 3D grids. [30] designs new geometric features for 3D object detection in a point cloud. [35, 17] convert a point cloud of the entire scene into a volumetric grid and use 3D volumetric CNN for object proposal and classification. Computation cost for those method is usually quite high due to the expensive cost of 3D convolutions and large 3D search space. Recently, [16] proposes a 2D-driven 3D object detection method that is similar to ours in spirit. However, they use hand-crafted features (based on histogram of point coordinates) with simple fully connected networks to regress 3D box location and pose, which is sub-optimal in both speed and performance. In contrast, we propose a more flexible and effective solution with deep 3D feature learning (PointNets).

Deep Learning on Point Clouds Most existing works convert point clouds to images or volumetric forms before feature learning. [40, 23, 26] voxelize point clouds into volumetric grids and generalize image CNNs to 3D CNNs. [19, 31, 39, 8] design more efficient 3D CNN or neural network architectures that exploit sparsity in point cloud. However, these CNN based methods still require quantitization of point clouds with certain voxel resolution. Recently, a few works [25, 27] propose a novel type of network architectures (PointNets) that directly consumes raw point clouds without converting them to other formats. While PointNets have been applied to single object classification and semantic segmentation, our work explores how to extend the architecture for the purpose of 3D object detection.

3. Problem Definition

Given RGB-D data as input, our goal is to classify and localize objects in 3D space. The depth data, obtained from LiDAR or indoor depth sensors, is represented as a point cloud in RGB camera coordinates. The projection matrix is also known so that we can get a 3D frustum from a 2D image region. Each object is represented by a class (one among k predefined classes) and an *amodal* 3D bounding box. The *amodal* box bounds the complete object even if part of the object is occluded or truncated. The 3D box is

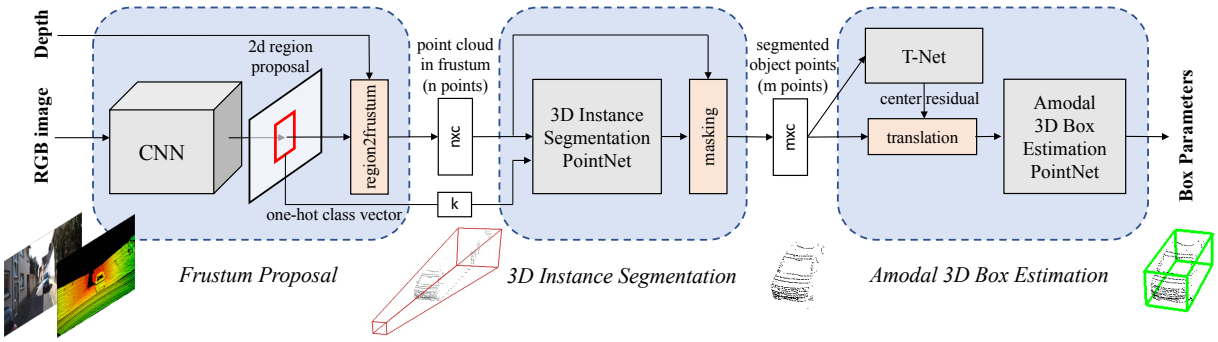

Figure 2. **Frustum PointNets for 3D object detection.** We first leverage a 2D CNN object detector to propose 2D regions and classify their content. 2D regions are then lifted to 3D and thus become frustum proposals. Given a point cloud in a frustum ($n \times c$ with n points and c channels of XYZ, intensity etc. for each point), the object instance is segmented by binary classification of each point. Based on the segmented object point cloud ($m \times c$), a light-weight regression PointNet (T-Net) tries to align points by translation such that their centroid is close to amodal box center. At last the box estimation net estimates the amodal 3D bounding box for the object. More illustrations on coordinate systems involved and network input, output are in Fig. 4 and Fig. 5.

parameterized by its size h, w, l, center c_x, c_y, c_z, and orientation θ, ϕ, ψ relative to a predefined canonical pose for each category. In our implementation, we only consider the heading angle θ around the up-axis for orientation.

4. 3D Detection with Frustum PointNets

As shown in Fig. 2, our system for 3D object detection consists of three modules: frustum proposal, 3D instance segmentation, and 3D amodal bounding box estimation. We will introduce each module in the following subsections. We will focus on the pipeline and functionality of each module, and refer readers to supplementary for specific architectures of the deep networks involved.

4.1. Frustum Proposal

The resolution of data produced by most 3D sensors, especially real-time depth sensors, is still lower than RGB images from commodity cameras. Therefore, we leverage mature 2D object detector to propose 2D object regions in RGB images as well as to classify objects.

With a known camera projection matrix, a 2D bounding box can be lifted to a frustum (with near and far planes specified by depth sensor range) that defines a 3D search space for the object. We then collect all points within the frustum to form a *frustum point cloud*. As shown in Fig 4 (a), frustums may orient towards many different directions, which result in large variation in the placement of point clouds. We therefore normalize the frustums by rotating them toward a center view such that the center axis of the frustum is orthogonal to the image plane. This normalization helps improve the rotation-invariance of the algorithm. We call this entire procedure for extracting frustum point clouds from RGB-D data *frustum proposal generation*.

While our 3D detection framework is agnostic to the exact method for 2D region proposal, we adopt a FPN [20] based model. We pre-train the model weights on ImageNet classification and COCO object detection datasets and further fine-tune it on a KITTI 2D object detection dataset to classify and predict *amodal* 2D boxes. More details of the 2D detector training are provided in the supplementary.

4.2. 3D Instance Segmentation

Given a 2D image region (and its corresponding 3D frustum), several methods might be used to obtain 3D location of the object: One straightforward solution is to directly regress 3D object locations (e.g., by 3D bounding box) from a depth map using 2D CNNs. However, this problem is not easy as occluding objects and background clutter is common in natural scenes (as in Fig. 3), which may severely distract the 3D localization task. Because objects are naturally separated in physical space, segmentation in 3D point cloud is much more natural and easier than that in images where pixels from distant objects can be near-by to each other. Having observed this fact, we propose to seg-

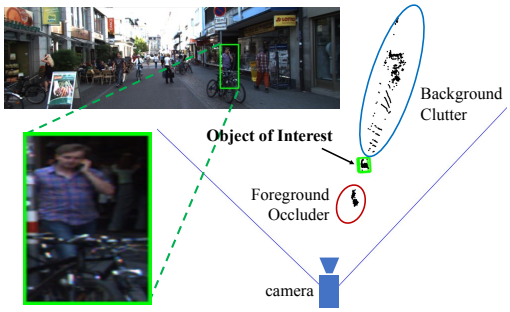

Figure 3. **Challenges for 3D detection in frustum point cloud.** *Left:* RGB image with an image region proposal for a person. *Right:* bird's eye view of the LiDAR points in the extruded frustum from 2D box, where we see a wide spread of points with both foreground occluder (bikes) and background clutter (building).

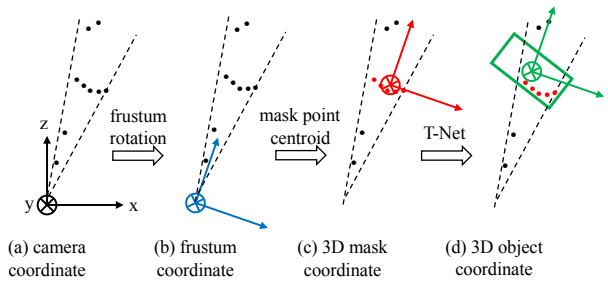

Figure 4. **Coordinate systems for point cloud.** Artificial points (black dots) are shown to illustrate (a) default camera coordinate; (b) frustum coordinate after rotating frustums to center view (Sec. 4.1); (c) mask coordinate with object points' centroid at origin (Sec. 4.2); (d) object coordinate predicted by T-Net (Sec. 4.3).

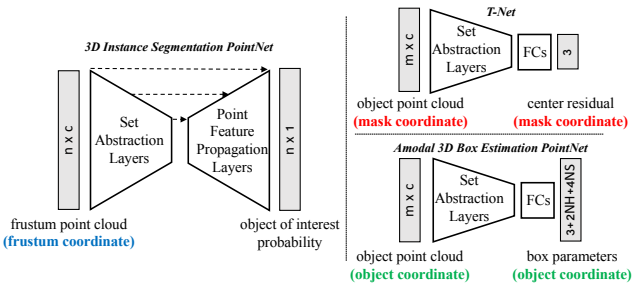

Figure 5. **Basic architectures and IO for PointNets.** Architecture is illustrated for PointNet++ [27] (v2) models with set abstraction layers and feature propagation layers (for segmentation). Coordinate systems involved are visualized in Fig. 4.

ment instances in 3D point cloud instead of in 2D image or depth map. Similar to Mask-RCNN [14], which achieves instance segmentation by binary classification of pixels in image regions, we realize 3D instance segmentation using a PointNet-based network on point clouds in frustums.

Based on 3D instance segmentation, we are able to achieve *residual* based 3D localization. That is, rather than regressing the absolute 3D location of the object whose offset from the sensor may vary in large ranges (e.g. from 5m to beyond 50m in KITTI data), we predict the 3D bounding box center in a local coordinate system – 3D mask coordinates as shown in Fig. 4 (c).

3D Instance Segmentation PointNet. The network takes a point cloud in frustum and predicts a probability score for each point that indicates how likely the point belongs to the object of interest. Note that each frustum contains exactly one object of interest. Here those "other" points could be points of non-relevant areas (such as ground, vegetation) or other instances that occlude or are behind the object of interest. Similar to the case in 2D instance segmentation, depending on the position of the frustum, object points in one frustum may become cluttered or occlude points in another. Therefore, our segmentation PointNet is learning the occlusion and clutter patterns as well as recognizing the geometry for the object of a certain category.

In a multi-class detection case, we also leverage the semantics from a 2D detector for better instance segmentation. For example, if we know the object of interest is a pedestrian, then the segmentation network can use this prior to find geometries that look like a person. Specifically, in our architecture we encode the semantic category as a one-hot class vector (k dimensional for the pre-defined k categories) and concatenate the one-hot vector to the intermediate point cloud features. More details of the specific architectures are described in the supplementary.

After 3D instance segmentation, points that are classified as the object of interest are extracted ("masking" in Fig. 2). Having obtained these segmented object points, we further normalize its coordinates to boost the translational invariance of the algorithm, following the same rationale as in the frustum proposal step. In our implementation, we transform the point cloud into a local coordinate by subtracting XYZ values by its centroid. This is illustrated in Fig. 4 (c). Note that we intentionally do not scale the point cloud, because the bounding sphere size of a partial point cloud can be greatly affected by viewpoints and the real size of the point cloud helps the box size estimation.

In our experiments, we find that coordinate transformations such as the one above and the previous frustum rotation are critical for 3D detection result as shown in Tab. 8.

4.3. Amodal 3D Box Estimation

Given the segmented object points (in 3D mask coordinate), this module estimates the object's amodal oriented 3D bounding box by using a box regression PointNet together with a preprocessing transformer network.

Learning-based 3D Alignment by T-Net Even though we have aligned segmented object points according to their centroid position, we find that the origin of the mask coordinate frame (Fig. 4 (c)) may still be quite far from the *amodal* box center. We therefore propose to use a light-weight regression PointNet (T-Net) to estimate the true center of the complete object and then transform the coordinate such that the predicted center becomes the origin (Fig. 4 (d)).

The architecture and training of our T-Net is similar to the T-Net in [25], which can be thought of as a special type of spatial transformer network (STN) [15]. However, different from the original STN that has no direct supervision on transformation, we explicitly supervise our translation network to predict center residuals from the mask coordinate origin to real object center.

Amodal 3D Box Estimation PointNet The box estimation network predicts amodal bounding boxes (for entire

object even if part of it is unseen) for objects given an object point cloud in 3D object coordinate (Fig. 4 (d)). The network architecture is similar to that for object classification [25, 27], however the output is no longer object class scores but parameters for a 3D bounding box.

As stated in Sec. 3, we parameterize a 3D bounding box by its center (c_x, c_y, c_z), size (h, w, l) and heading angle θ (along up-axis). We take a "residual" approach for box center estimation. The center residual predicted by the box estimation network is combined with the previous center residual from the T-Net and the masked points' centroid to recover an absolute center (Eq. 1). For box size and heading angle, we follow previous works [29, 24] and use a hybrid of classification and regression formulations. Specifically we pre-define NS size templates and NH equally split angle bins. Our model will both classify size/heading (NS scores for size, NH scores for heading) to those pre-defined categories as well as predict residual numbers for each category ($3 \times NS$ residual dimensions for height, width, length, NH residual angles for heading). In the end the net outputs $3 + 4 \times NS + 2 \times NH$ numbers in total.

$$C_{pred} = C_{mask} + \Delta C_{t-net} + \Delta C_{box-net} \tag{1}$$

4.4. Training with Multi-task Losses

We simultaneously optimize the three nets involved (3D instance segmentation PointNet, T-Net and amodal box estimation PointNet) with multi-task losses (as in Eq. 2). L_{c1-reg} is for T-Net and L_{c2-reg} is for center regression of box estimation net. L_{h-cls} and L_{h-reg} are losses for heading angle prediction while L_{s-cls} and L_{s-reg} are for box size. Softmax is used for all classification tasks and smooth-l_1 (huber) loss is used for all regression cases.

$$\begin{aligned} L_{multi-task} =& L_{seg} + \lambda(L_{c1-reg} + L_{c2-reg} + L_{h-cls} + \\ & L_{h-reg} + L_{s-cls} + L_{s-reg} + \gamma L_{corner}) \end{aligned} \tag{2}$$

Corner Loss for Joint Optimization of Box Parameters While our 3D bounding box parameterization is compact and complete, learning is not optimized for final 3D box accuracy – center, size and heading have *separate* loss terms. Imagine cases where center and size are accurately predicted but heading angle is off – the 3D IoU with ground truth box will then be dominated by the angle error. Ideally all three terms (center,size,heading) should be *jointly* optimized for best 3D box estimation (under IoU metric). To resolve this problem we propose a novel regularization loss, the *corner loss*:

$$L_{corner} = \sum_{i=1}^{NS}\sum_{j=1}^{NH} \delta_{ij} \min\{\sum_{k=1}^{8} \|P_k^{ij} - P_k^*\|, \sum_{i=1}^{8} \|P_k^{ij} - P_k^{**}\|\} \tag{3}$$

In essence, the corner loss is the sum of the distances between the eight corners of a predicted box and a ground truth box. Since corner positions are jointly determined by center, size and heading, the corner loss is able to regularize the multi-task training for those parameters.

To compute the corner loss, we firstly construct $NS \times NH$ "anchor" boxes from all size templates and heading angle bins. The anchor boxes are then translated to the estimated box center. We denote the anchor box corners as P_k^{ij}, where i, j, k are indices for the size class, heading class, and (predefined) corner order, respectively. To avoid large penalty from flipped heading estimation, we further compute distances to corners (P_k^{**}) from the flipped ground truth box and use the minimum of the original and flipped cases. δ_{ij}, which is one for the ground truth size/heading class and zero else wise, is a two-dimensional mask used to select the distance term we care about.

5. Experiments

Experiments are divided into three parts[1]. First we compare with state-of-the-art methods for 3D object detection on KITTI [10] and SUN-RGBD [33] (Sec 5.1). Second, we provide in-depth analysis to validate our design choices (Sec 5.2). Last, we show qualitative results and discuss the strengths and limitations of our methods (Sec 5.3).

5.1. Comparing with state-of-the-art Methods

We evaluate our 3D object detector on KITTI [11] and SUN-RGBD [33] benchmarks for 3D object detection. On both tasks we have achieved significantly better results compared with state-of-the-art methods.

KITTI Tab. 1 shows the performance of our 3D detector on the KITTI *test* set. We outperform previous state-of-the-art methods by a large margin. While MV3D [6] uses multi-view feature aggregation and sophisticated multi-sensor fusion strategy, our method based on the PointNet [25] (v1) and PointNet++ [27] (v2) backbone is much cleaner in design. While out of the scope for this work, we expect that sensor fusion (esp. aggregation of image feature for 3D detection) could further improve our results.

We also show our method's performance on 3D object localization (bird's eye view) in Tab. 2. In the 3D localization task bounding boxes are projected to bird's eye view plane and IoU is evaluated on oriented 2D boxes. Again, our method significantly outperforms previous works which include DoBEM [42] and MV3D [6] that use CNNs on projected LiDAR images, as well as 3D FCN [17] that uses 3D CNNs on voxelized point cloud.

[1]Details on network architectures, training parameters as well as more experiments are included in the supplementary material.

Method	Cars			Pedestrians			Cyclists		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
DoBEM [42]	7.42	6.95	13.45	-	-	-	-	-	-
MV3D [6]	71.09	62.35	55.12	-	-	-	-	-	-
Ours (v1)	80.62	64.70	56.07	50.88	41.55	38.04	69.36	53.50	52.88
Ours (v2)	**81.20**	**70.39**	**62.19**	**51.21**	**44.89**	**40.23**	**71.96**	**56.77**	**50.39**

Table 1. **3D object detection** *3D* AP on KITTI *test* set. DoBEM [42] and MV3D [6] (previous state of the art) are based on 2D CNNs with bird's eye view LiDAR image. Our method, without sensor fusion or multi-view aggregation, outperforms those methods by large margins on all categories and data subsets. 3D bounding box IoU threshold is 70% for cars and 50% for pedestrians and cyclists.

Method	Cars			Pedestrians			Cyclists		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
DoBEM [42]	36.49	36.95	38.10	-	-	-	-	-	-
3D FCN [17]	69.94	62.54	55.94	-	-	-	-	-	-
MV3D [6]	86.02	76.90	68.49	-	-	-	-	-	-
Ours (v1)	87.28	77.09	67.90	55.26	47.56	42.57	73.42	59.87	52.88
Ours (v2)	**88.70**	**84.00**	**75.33**	**58.09**	**50.22**	**47.20**	**75.38**	**61.96**	**54.68**

Table 2. **3D object localization** AP (bird's eye view) on KITTI *test* set. 3D FCN [17] uses 3D CNNs on voxelized point cloud and is far from real-time. MV3D [6] is the previous state of the art. Our method significantly outperforms those methods on all categories and data subsets. Bird's eye view 2D bounding box IoU threshold is 70% for cars and 50% for pedestrians and cyclists.

Method	Easy	Moderate	Hard
Mono3D [4]	2.53	2.31	2.31
3DOP [5]	6.55	5.07	4.10
VeloFCN [17]	15.20	13.66	15.98
MV3D (LiDAR) [6]	71.19	56.60	55.30
MV3D [6]	71.29	62.68	56.56
Ours (v1)	83.26	69.28	62.56
Ours (v2)	**83.76**	**70.92**	**63.65**

Table 3. **3D object detection** AP on KITTI *val* set (cars only).

Method	Easy	Moderate	Hard
Mono3D [4]	5.22	5.19	4.13
3DOP [5]	12.63	9.49	7.59
VeloFCN [17]	40.14	32.08	30.47
MV3D (LiDAR) [6]	86.18	77.32	76.33
MV3D [6]	86.55	78.10	**76.67**
Ours (v1)	87.82	82.44	74.77
Ours (v2)	**88.16**	**84.02**	76.44

Table 4. **3D object localization** AP on KITTI *val* set (cars only).

The output of our network is visualized in Fig. 6 where we observe accurate 3D instance segmentation and box prediction even under very challenging cases. We defer more discussions on success and failure case patterns to Sec. 5.3. We also report performance on KITTI val set (the same split as in [6]) in Tab. 3 and Tab. 4 (for cars) to support comparison with more published works, and in Tab. 5 (for pedestrians and cyclists) for reference.

SUN-RGBD Most previous 3D detection works specialize either on outdoor LiDAR scans where objects are well separated in space and the point cloud is sparse (so that it's feasible for bird's eye projection), or on indoor depth maps that are regular images with dense pixel values such that image CNNs can be easily applied. However, methods designed for bird's eye view may be incapable for indoor rooms where multiple objects often exist together in vertical space. On the other hand, indoor focused methods could find it hard to apply to sparse and large-scale point cloud from LiDAR scans.

Benchmark	Easy	Moderate	Hard
Pedestrian (3D Detection)	70.00	61.32	53.59
Pedestrian (Bird's Eye View)	72.38	66.39	59.57
Cyclist (3D Detection)	77.15	56.49	53.37
Cyclist (Bird's Eye View)	81.82	60.03	56.32

Table 5. Performance on KITTI *val* set for pedestrians and cyclists. Model evaluated is Ours (v2).

In contrast, our frustum-based PointNet is a generic framework for both outdoor and indoor 3D object detection. By applying the same pipeline we used for KITTI data set, we've achieved state-of-the-art performance on SUN-RGBD benchmark (Tab. 6) with significantly higher mAP as well as much faster (10x-1000x) inference speed.

5.2. Architecture Design Analysis

In this section we provide analysis and ablation experiments to validate our design choices.

Experiment setup. Unless otherwise noted, all experiments in this section are based on our v1 model on KITTI data using train/val split as in [6]. To decouple the influence of 2D detectors, we use ground truth 2D boxes for region proposals and use 3D box estimation accuracy (IoU threshold 0.7) as the evaluation metric. We will only focus on the car category which has the most training examples.

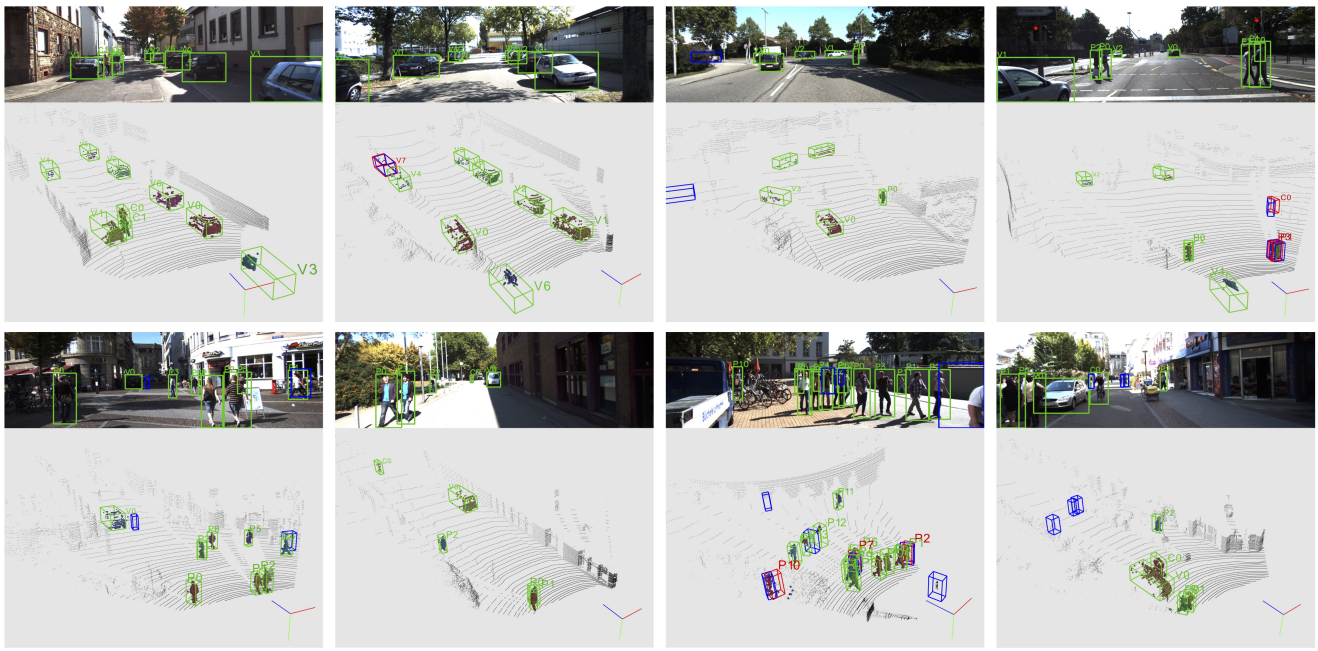

Figure 6. **Visualizations of Frustum PointNet results on KITTI val set** (best viewed in color with zoom in). These results are based on PointNet++ models [27], running at 5 fps and achieving test set *3D* AP of 70.39, 44.89 and 56.77 for car, pedestrian and cyclist, respectively. 3D instance masks on point cloud are shown in color. True positive detection boxes are in green, while false positive boxes are in red and groundtruth boxes in blue are shown for false positive and false negative cases. Digit and letter beside each box denote instance id and semantic class, with "v" for cars, "p" for pedestrian and "c" for cyclist. See Sec. 5.3 for more discussion on the results.

	bathtub	bed	bookshelf	chair	desk	dresser	nightstand	sofa	table	toilet	Runtime	mAP
DSS [35]	44.2	78.8	11.9	61.2	20.5	6.4	15.4	53.5	50.3	78.9	19.55s	42.1
COG [30]	**58.3**	63.7	31.8	62.2	**45.2**	15.5	27.4	51.0	**51.3**	70.1	10-30min	47.6
2D-driven [16]	43.5	64.5	31.4	48.3	27.9	25.9	41.9	50.4	37.0	80.4	4.15s	45.1
Ours (v1)	43.3	**81.1**	**33.3**	**64.2**	24.7	**32.0**	**58.1**	**61.1**	51.1	**90.9**	0.12s	**54.0**

Table 6. **3D object detection AP on SUN-RGBD val set.** Evaluation metric is average precision with 3D IoU threshold 0.25 as proposed by [33]. Note that both COG [30] and 2D-driven [16] use room layout context to boost performance while ours and DSS [35] not. Compared with previous state-of-the-arts our method is 6.4% to 11.9% better in mAP as well as one to three orders of magnitude faster.

Comparing with alternative approaches for 3D detection. In this part we evaluate a few CNN-based baseline approaches as well as ablated versions and variants of our pipelines using 2D masks. In the first row of Tab. 7, we show 3D box estimation results from two CNN-based networks. The baseline methods trained VGG [32] models on ground truth boxes of RGB-D images and adopt the same box parameter and loss functions as our main method. While the model in the first row directly estimates box location and parameters from vanilla RGB-D image patch, the other one (second row) uses a FCN trained from the COCO dataset for 2D mask estimation (as that in Mask-RCNN [14]) and only uses features from the masked region for prediction. The depth values are also translated by subtracting the median depth within the 2D mask. However, both CNN baselines get far worse results compared to our main method.

To understand why CNN baselines underperform, we visualize a typical 2D mask prediction in Fig. 7. While the estimated 2D mask appears in high quality on an RGB image, there are still lots of clutter and foreground points in the 2D mask. In comparison, our 3D instance segmentation gets much cleaner result, which greatly eases the next module in finer localization and bounding box regression.

In the third row of Tab. 7, we experiment with an ablated version of frustum PointNet that has no 3D instance segmentation module. Not surprisingly, the model gets much worse results than our main method, which indicates the critical effect of our 3D instance segmentation module. In the fourth row, instead of 3D segmentation we use point clouds from 2D masked depth maps (Fig. 7) for 3D box estimation. However, since a 2D mask is not able to cleanly segment the 3D object, the performance is more than 12% worse than that with the 3D segmentation (our main method in the fifth row). On the other hand, a combined usage of 2D and 3D masks – applying 3D segmentation on point cloud

network arch.	mask	depth representation	accuracy
ConvNet	-	image	18.3
ConvNet	2D	image	27.4
PointNet	-	point cloud	33.5
PointNet	2D	point cloud	61.6
PointNet	3D	point cloud	**74.3**
PointNet	2D+3D	point cloud	70.0

Table 7. **Comparing 2D and 3D approaches.** 2D mask is from FCN on RGB image patch. 3D mask is from PointNet on frustum point cloud. 2D+3D mask is 3D mask generated by PointNet on point cloud poped up from 2D masked depth map.

frustum rot.	mask centralize	t-net	accuracy
-	-	-	12.5
√	-	-	48.1
-	√	-	64.6
√	√	-	71.5
√	√	√	**74.3**

Table 8. **Effects of point cloud normalization.** Metric is 3D box estimation accuracy with IoU=0.7.

loss type	regularization	accuracy
regression only	-	62.9
cls-reg	-	71.8
cls-reg (normalized)	-	72.2
cls-reg (normalized)	corner loss	**74.3**

Table 9. **Effects of 3D box loss formulations**. Metric is 3D box estimation accuracy with IoU=0.7.

from 2D masked depth map – also shows slightly worse results than our main method probably due to the accumulated error from inaccurate 2D mask predictions.

Effects of point cloud normalization. As shown in Fig. 4, our frustum PointNet takes a few key coordinate transformations to canonicalize the point cloud for more effective learning. Tab. 8 shows how each normalization step helps for 3D detection. We see that both frustum rotation (such that frustum points have more similar XYZ distributions) and mask centroid subtraction (such that object points have smaller and more canonical XYZ) are critical. In addition, extra alignment of object point cloud to object center by T-Net also contributes significantly to the performance.

Effects of regression loss formulation and corner loss. In Tab. 9 we compare different loss options and show that a combination of "cls-reg" loss (the classification and residual regression approach for heading and size regression) and a regularizing corner loss achieves the best result.

The naive baseline using regression loss only (first row) achieves unsatisfactory result because the regression target is large in range (object size from 0.2m to 5m). In comparison, the cls-reg loss and a normalized version (residual normalized by heading bin size or template shape size) of it achieve much better performance. At last row we show that a regularizing corner loss further helps optimization.

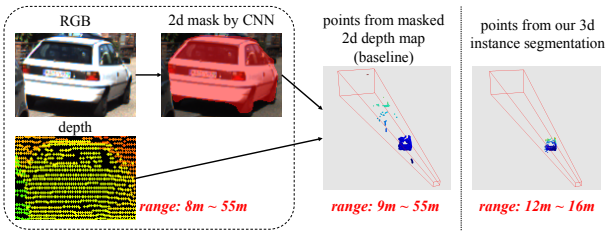

Figure 7. **Comparisons between 2D and 3D masks.** We show a typical 2D region proposal from KITTI val set with both 2D (on RGB image) and 3D (on frustum point cloud) instance segmentation results. The red numbers denote depth ranges of points.

5.3. Qualitative Results and Discussion

In Fig. 6 we visualize representative outputs of our frustum PointNet model. We see that for simple cases of non-occluded objects in reasonable distance (so we get enough number of points), our model outputs remarkably accurate 3D instance segmentation mask and 3D bounding boxes. Second, we are surprised to find that our model can even predict correctly posed *amodal* 3D box from partial data (e.g. parallel parked cars) with few points. Even humans find it very difficult to annotate such results with point cloud data only. Third, in some cases that seem very challenging in images with lots of nearby or even overlapping 2D boxes, when converted to 3D space, the localization becomes much easier (e.g. P11 in second row third column).

On the other hand, we do observe several failure patterns, which indicate possible directions for future efforts. The *first* common mistake is due to inaccurate pose and size estimation in a sparse point cloud (sometimes less than 5 points). We think image features could greatly help esp. since we have access to high resolution image patch even for far-away objects. The *second* type of challenge is when there are multiple instances from the same category in a frustum (like two persons standing by). Since our current pipeline assumes a single object of interest in each frustum, it may get confused when multiple instances appear and thus outputs mixed segmentation results. This problem could potentially be mitigated if we are able to propose multiple 3D bounding boxes within each frustum. *Thirdly*, sometimes our 2D detector misses objects due to dark lighting or strong occlusion. Since our frustum proposals are based on region proposal, no 3D object will be detected given no 2D detection. However, our 3D instance segmentation and amodal 3D box estimation PointNets are not restricted to RGB view proposals. As shown in the supplementary, the same framework can also be extended to 3D regions proposed in bird's eye view.

Acknowledgement The authors wish to thank the support of Nuro Inc., ONR MURI grant N00014-13-1-0341, NSF grants DMS-1546206 and IIS-1528025, a Samsung GRO award, and gifts from Adobe, Amazon, and Apple.

References

[1] Kitti 3d object detection benchmark leader board. http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d. Accessed: 2017-11-14 12PM. 2

[2] Kitti bird's eye view object detection benchmark leader board. http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=bev. Accessed: 2017-11-14 12PM. 2

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 14

[4] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2156, 2016. 2, 6, 11

[5] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object class detection. In *Advances in Neural Information Processing Systems*, pages 424–432, 2015. 6

[6] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE CVPR*, 2017. 2, 5, 6, 11, 12, 13

[7] Z. Deng and L. J. Latecki. Amodal detection of 3d objects: Inferring 3d bounding boxes from 2d ones in rgb-depth images. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017. 2

[8] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1355–1361. IEEE, 2017. 1, 2

[9] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017. 12

[10] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 5

[11] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 5

[12] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 12

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014. 1

[14] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017. 1, 3, 7

[15] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *NIPS 2015*. 4

[16] J. Lahoud and B. Ghanem. 2d-driven 3d object detection in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4622–4630, 2017. 2, 7

[17] B. Li. 3d fully convolutional network for vehicle detection in point cloud. *arXiv preprint arXiv:1611.08069*, 2016. 2, 5, 6

[18] B. Li, T. Zhang, and T. Xia. Vehicle detection from 3d lidar using fully convolutional network. *arXiv preprint arXiv:1608.07916*, 2016. 2, 13

[19] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. Fpnn: Field probing neural networks for 3d data. *arXiv preprint arXiv:1605.06240*, 2016. 2

[20] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. *arXiv preprint arXiv:1612.03144*, 2016. 3, 12

[21] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017. 12

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 12

[23] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2015. 1, 2

[24] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka. 3d bounding box estimation using deep learning and geometry. *arXiv preprint arXiv:1612.00496*, 2016. 2, 5

[25] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 1, 2, 4, 5, 10, 11, 13

[26] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2016. 1, 2

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 1, 2, 4, 5, 7, 10, 11, 13, 14

[28] J. Ren, X. Chen, J. Liu, W. Sun, J. Pang, Q. Yan, Y.-W. Tai, and L. Xu. Accurate single stage detector using recurrent rolling convolution. In *CVPR*, 2017. 13

[29] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 2, 5, 12

[30] Z. Ren and E. B. Sudderth. Three-dimensional object detection and layout prediction using clouds of oriented gradients. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1525–1533, 2016. 2, 7, 12

[31] G. Riegler, A. O. Ulusoys, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. *arXiv preprint arXiv:1611.05009*, 2016. 2

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 7, 12, 13

[33] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 567–576, 2015. 1, 5, 7, 12

[34] S. Song and J. Xiao. Sliding shapes for 3d object detection in depth images. In *Computer Vision–ECCV 2014*, pages 634–651. Springer, 2014. 2

[35] S. Song and J. Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 808–816, 2016. 2, 7

[36] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015. 1

[37] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *arXiv preprint arXiv:1707.02968*, 1, 2017. 14

[38] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. *Proceedings of the Robotics: Science and Systems, Rome, Italy*, 1317, 2015. 2

[39] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017. 2

[40] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015. 1, 2

[41] Y. Xiang, W. Choi, Y. Lin, and S. Savarese. Data-driven 3d voxel patterns for object category recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1903–1911, 2015. 2

[42] S.-L. Yu, T. Westfechtel, R. Hamada, K. Ohno, and S. Tadokoro. Vehicle detection and localization on birds eye view elevation images using convolutional neural network. *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2017. 5, 6

A. Overview

This document provides additional technical details, extra analysis experiments, more quantitative results and qualitative test results to the main paper.

In Sec.B we provide more details on network architectures of PointNets and training parameters while Sec. C explains more about our 2D detector. Sec. D shows how our framework can be extended to bird's eye view (BV) proposals and how combining BV and RGB proposals can further improve detection performance. Then Sec. E presents results from more analysis experiments. At last, Sec. F shows more visualization results for 3D detection on SUN-RGBD dataset.

B. Details on Frustum PointNets (Sec 4.2, 4.3)

B.1. Network Architectures

We adopt similar network architectures as in the original works of PointNet [25] and PointNet++ [27] for our v1 and v2 models respectively. What is different is that we add an extra link for class one-hot vector such that instance segmentation and bounding box estimation can leverage semantics predicted from RGB images. The detailed network architectures are shown in Fig. 8.

For v1 model our architecture involves point embedding layers (as shared MLP on each point independently), a max pooling layer and per-point classification multi-layer perceptron (MLP) based on aggregated information from global feature and each point as well as an one-hot class vector. Note that we do not use the transformer networks as in [25] because frustum points are viewpoint based (not complete point cloud as in [25]) and are already normalized by frustum rotation. In addition to XYZ , we also leverage LiDAR intensity as a fourth channel.

For v2 model we use set abstraction layers for hierarchical feature learning in point clouds. In addition, because LiDAR point cloud gets increasingly sparse as it gets farther, feature learning has to be robust to those density variations. Therefore we used a robust type of set abstraction layers – multi-scale grouping (MSG) layers as introduced in [27] for the segmentation network. With hierarchical features and learned robustness to varying densities, our v2 model shows superior performance than v1 model in both segmentation and box estimation.

B.2. Data Augmentation and Training

Data augmentation Data augmentation plays an important role in preventing model overfitting. Our augmentation involves two branches: one is 2D box augmentation and the other is frustum point cloud augmentation.

We use ground truth 2D boxes to generate frustum point clouds for Frustum PointNets training and augment the 2D boxes by random translation and scaling. Specifically, we

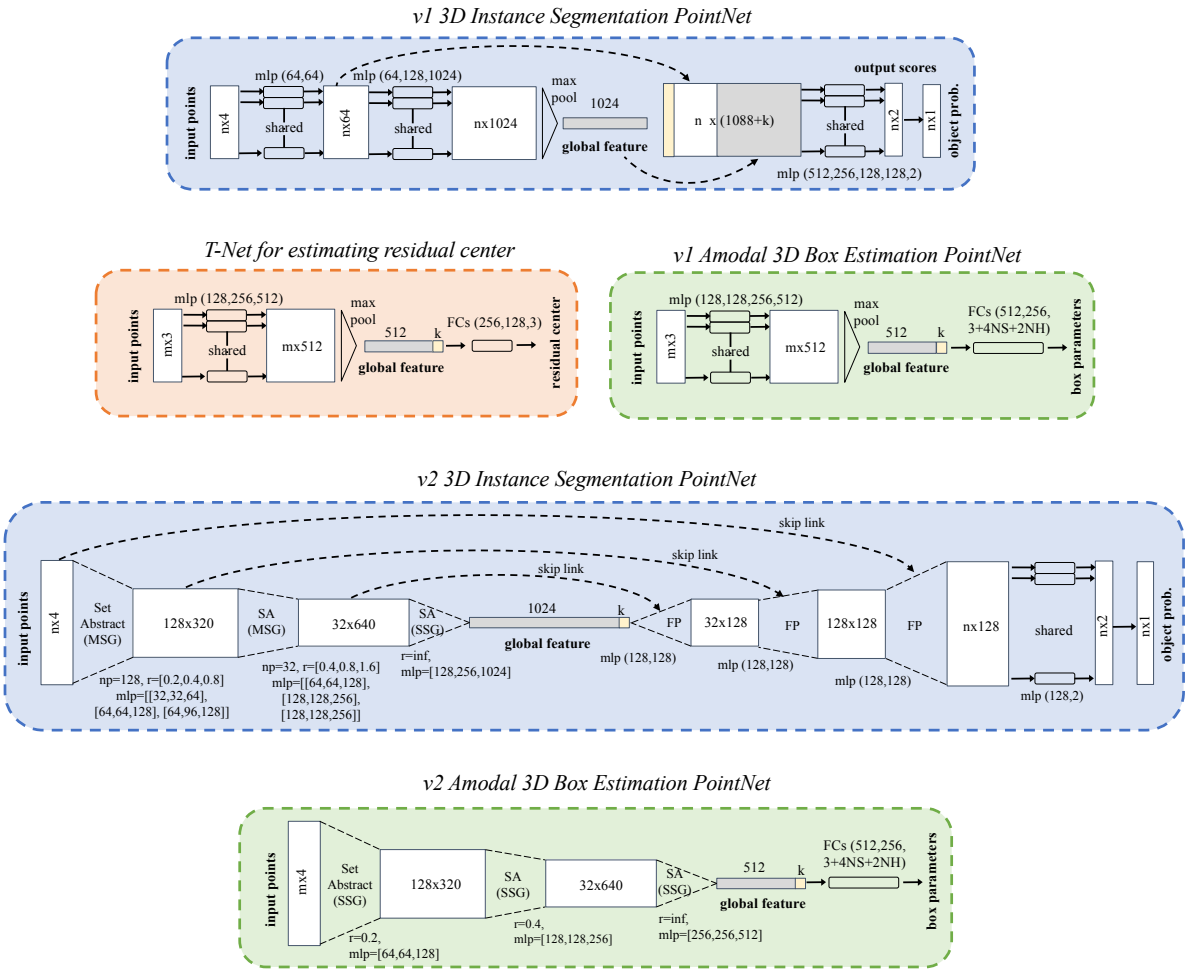

Figure 8. **Network architectures for Frustum PointNets.** v1 models are based on PointNet [25]. v2 models are based on PointNet++ [27] set abstraction (SA) and feature propagation (FP) layers. The architecture for residual center estimation T-Net is shared for Ours (v1) and Ours (v2). The colors (blue for segmentaiton nets, red for T-Net and green for box estimation nets) of the network background indicate the coordinate system of the input point cloud. Segmentation nets operate in frustum coordinate, T-Net processes points in mask coordinate while box estimation nets take points in object coordinate. The small yellow square (or bar) concatenated with global features is class one-hot vector that tells the predicted category of the underlying object.

firstly compute the 2D box height (h) and width (w) and translate the 2D box center by random distances sampled from Uniform$[-0.1w, 0.1w]$ and Uniform$[-0.1h, 0.1h]$ in u,v directions respectively. The height and width are also augmented by two random scaling factor sampled from Uniform$[0.9, 1.1]$.

We augment each frustum point cloud by three ways. First, we randomly sample a subset of points from the frustum point cloud on the fly (1,024 for KITTI and 2,048 for SUN-RGBD). For object points segmented from our predicted 3D mask, we randomly sample 512 points from it (if there are less than 512 points we will randomly resample to make up for the number). Second, we randomly flip the frustum point cloud (after rotating the frustum to the center) along the YZ plane in camera coordinate (Z is forward, Y is pointing down). Thirdly, we perturb the points by shifting the entire frustum point cloud in Z-axis direction such that the depth of points is augmented. Together with all data augmentation, we modify the ground truth labels for 3D mask and headings correspondingly.

KITTI Training The object detection benchmark in KITTI provides synchronized RGB images and LiDAR point clouds with ground truth amodal 2D and 3D box annotations for vehicles, pedestrians and cyclists. The training set contains 7,481 frames and an undisclosed test set contains 7,581 frames. In our own experiments (except those for test sets), we follow [4, 6] to split the official training set to a train set of 3,717 frames and a val set of 3769 frames such that frames in train/val sets belong to different

video clips. For models evaluated on the test set we train our model on our own train/val split where around 80% of the training data is used such that the model can achieve better generalization by seeing more examples.

To get ground truth for 3D instance segmentation we simply consider all points that fall into the ground truth 3D bounding box as object points. Although there are sometimes false labels from ground points or points from other closeby objects (e.g. a person standing by), the auto-labeled segmentation ground truth is in general acceptable.

For both of our v1 and v2 models, we use Adam optimizer with starting learning rate 0.001, with step-wise decay (by half) in every 60k iterations. For all trainable layers except the last classification or regression ones, we use batch normalization with a start decay rate of 0.5 and gradually decay the decay rate to 0.99 (step-wise decay with rate 0.5 in every 20k iterations). We use batch size 32 for v1 models and batch size 24 for v2 models. All three PointNets are trained end-to-end.

Trained on a single GTX 1080 GPU, it takes around one day to train a v1 model (all three nets) for 200 epochs while it takes around three days for a v2 model. We picked the early stopped (200 epochs) snapshot models for evaluation.

SUN-RGBD Training The data set consists of 10,355 RGB-D images captured from various depth sensors for indoor scenes (bedrooms, dining rooms etc.). We follow the same train/val splits as [33, 30] for experiments. The data augmentation and optimization parameters are the same as that in KITTI.

As to auto-labeling of instance segmentation mask, however, data quality is much lower than that in KITTI because of strong occlusions and tight arrangement of objects in indoor scenes (see Fig. 11 for some examples). Nonetheless we still consider all points within the ground truth boxes as object points for our training. For 3D segmentation we get only a 82.7% accuracy compared to around 90% in KITTI. Due to the heavy noise in segmentation mask label, we choose to only train and evaluate on v1 models that has more strength in global feature learning than v2 ones. For future works, we think higher quality in 3D mask labels can greatly help the instance segmentation network training.

C. Details on RGB Detector (Sec 4.1)

For 2D RGB image detector, we use the encoder-decoder structure (e.g. DSSD [9], FPN [20]) to generate region proposals from multiple feature maps using focal loss [21] and use Fast R-CNN [12] to predict final 2D detection bounding boxes from the region proposals.

To make the detector faster, we take the reduced VGG [32] base network architecture from SSD [22], sample half of the channels per layer and change all max pooling layers to convolution layers with 3×3 kernel size and stride of 2. Then we fine-tune it on ImageNet CLS-LOC dataset for 400k iterations with batch size of 260 on 10 GPUs. The resulting base network architecture has about 66.7% top-1 classification accuracy on the CLS-LOC validation dataset and only needs about 1.2ms to process a 224×224 image on a NVIDIA GTX 1080.

We then add the feature pyramid layers [20] from conv3_3, conv4_3, conv5_3, and fc7, which are used to predict region proposals with scales of 16, 32, 64, 128 respectively. We also add an extra convolutional layer (conv8) which halves the fc7 feature map size, and use it to predict proposals with scale of 256. We use 5 different aspect ratios $\{\frac{1}{3}, \frac{1}{2}, 1, 2, 3\}$ for all layers except that we ignore $\{\frac{1}{3}, 3\}$ for conv3_3. Following SSD, we also use normalization layer on conv3_3, conv4_3, and conv5_3 and initialize the norm 40. For Fast R-CNN part, we extract features from conv3_3, conv5_3, and conv8 for each region proposal and concatenate all the features to predict class scores and further adjust the proposals. We train this detector from COCO dataset with 384×384 input image and have achieved 35.5 mAP on the COCO minival dataset, with only 10ms processing time for a 384×384 image on a single GPU.

Finally, we fine-tune the detector on car, people, and bicycle from COCO dataset, and have achieved 48.5, 44.1, and 40.1 for these three classes on COCO. We take this model and further fine-tune it on car, pedestrian, and cyclist from KITTI dataset. The final model takes about 30ms to process a 384×1280 image. To increase the recall of the detector, we also do detection from the center crop of the image besides the full image, and then merge the detections using non-maximum suppression.

Tab. 10 shows our detector's AP (2D) on KITTI test set. Our detector has achieved competitive or better results than current leading players on KITTI leader board. We've also reported our AP (2D) on val set in Tab. 11 for reference.

D. Bird's Eye View PointNets (Sec 5.3)

In this section, we show that our 3D detection framework can also be extended to using bird's eye view proposals, which adds another orthogonal proposal source to achieve better overall 3D detection performance. We evaluate the results of car detection using LiDAR bird's eye view only proposals + point net (Ours(BV)), and combine frustum point net and bird's eye view point net using 3D non-maximum suppression (NMS) (Ours(Frustum + BV)). The results are shown in Table 12.

Bird's Eye View Proposal Similar to MV3D [6] we use point features such as height, intensity and density, and train the bird's eye view 2D proposal net using the standard Faster-RCNN [29] structure. The net outputs axis-aligned 2D bounding boxes in the bird's eye view. In detail, we

Method	Cars			Pedestrians			Cyclists		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
SWC	**90.82**	90.05	80.59	87.06	**78.65**	73.92	**86.02**	**77.58**	**68.44**
RRC [28]	90.61	**90.22**	**87.44**	84.14	75.33	70.39	84.96	76.47	65.46
Ours	90.78	90.00	80.80	**87.81**	77.25	**74.46**	84.90	72.25	65.14

Table 10. **2D object detection** AP on KITTI *test* set. Evaluation IoU threshold is 0.7. SWC is the first place winner on KITTI leader board for pedestrians and cyclists at the time of submission. Our 2D results are based on a CNN model on monocular RGB images.

Subset	Easy	Moderate	Hard
AP (2D) for cars	96.48	90.31	87.63

Table 11. **Our 2D object detection** AP on KITTI *val* set.

discretize the projected point clouds into 2D grids with resolution of 0.1 meter and with the depth and width range 0 60 meters, which gives us the 600×600 input size. For each cell, we take the intensity and the density of the highest point and divide the heights into 7 bins with the height of the highest point in each bin, which gives us 9 channels in total. In Faster R-CNN, we use the VGG-16 [32] with 3 anchor scales $(16, 32, 48)$ and 3 aspect ratios $(\frac{1}{2}, 1, 2)$. We train RPN and Fast R-CNN together using the approximate joint training.

To combine 3D detection boxes from frustum PointNets and the bird's eye view PointNets, we use 3D NMS with IoU threshold 0.8. We also apply a weight (0.5) to 3D boxes from BV PointNets since it is a weaker detector compared with our frustum one.

Bird's Eye View (BV) PointNets Similar to Frustum PointNets that take point cloud in frustum, segment point cloud and estimate amodal bounding box, we can apply PointNets to points in bird's eye view regions. Since bird's eye view is based on orthogonal projection, the 3D space specified by a BV 2D box is a 3D cuboid (cut by minimum and maximum height) instead of a frustum.

Results Tab. 12 (Ours BV) shows the APs we get by using bird's eye view proposals only (without and RGB information). We compare with two previous LiDAR only methods (VeloFCN [18] and MV3D (BV+FV) [6]) and show that our BV proposal based detector greatly outperforms VeloFCN on all cases and outperforms MV3D (BV+FV) on moderate and hard cases by a significant margin.

More importantly, we show in the last row of Tab. 12 that bird's eye view and RGB view proposals can be combined to achieve an even better performance (*3.8%* AP improvement on hard cases). Fig. 9 gives an intuitive explanation of why bird's eye view proposals could help. In the sample frame shown: while our 2D detector misses some highly occluded cars (Fig. 9: left RGB image), bird's eye view based RPN successfully detects them (Fig. 9: blue arrows in right LiDAR image).

Method	Easy	Moderate	Hard
VeloFCN [18]	15.20	13.66	15.98
MV3D [6] (BV+FV)	71.19	56.60	55.30
Ours (BV)	69.50	62.30	59.73
Ours (Frustum)	**83.76**	**70.92**	63.65
Ours (Frustum + BV)	**83.76**	70.91	**67.47**

Table 12. **3D object detection** AP on KITTI *val* set. By using both proposals from RGB view (frustum) and bird's eye view (BV), we see a significant improvement in 3D AP (*3.82%*) on hard cases compared with our frustum only method. Ours (Frustum) here is the Ours (v2) in the main paper using PointNet++ architectures.

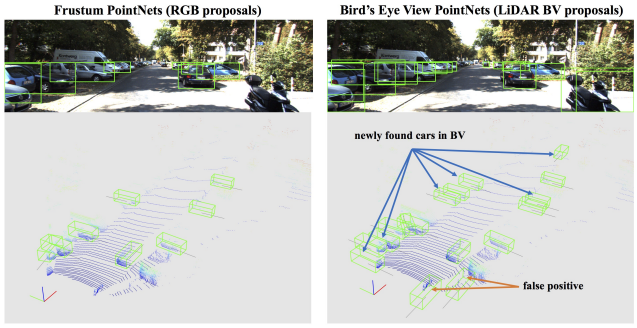

Figure 9. **Comparing Frustum PointNets and BV PointNets.** This is a scene with lots of parallel parking cars (sample 5595 from val set). *Left* column shows 2D boxes from our 2D detector in image and 3D boxes from our Frustum PointNets in point cloud. *Right* column shows 3D boxes from BV PointNets in point cloud and the 2D boxes (projected from the 3D detection boxes) in image. Note that 2D detection boxes from Ours (Frustum) that have box height less than 25 pixels or contain no LiDAR points in the frustum are not shown in the image.

E. More Experiments (Sec 5.2)

E.1. Effects of PointNet Architectures

Table 13 compares PointNet [25] (v1) and PointNet++ [27] (v2) architectures for instance segmentation and amodal box estimation. The v2 model outperforms v1 model on both tasks because 1) v2 model learns hierarchical features that are richer and more generalizable; 2) v2 model uses multi-scale feature learning that adapts to varying point densities. Note that the ours (v1) model corresponds to first row of Table 13 while the ours (v2) links to the last row.

seg net	box net	seg acc.	box acc.
v1	v1	90.6	74.3
v2	v1	**91.0**	74.7
v1	v2	90.6	76.0
v2	v2	**91.0**	**77.1**

Table 13. **Effects of PointNet architectures.** Metric is 3D box estimation accuracy with IoU=0.7.

E.2. Effects of Training Data Size

Recently [37] observed linear improvement in performance of deep learning models with exponential growth of data set size. In our Frustum PointNets we observe similar trend (Fig. 10). This trend indicates a promising performance potential of our methods with larger datasets.

We train three separate group of Frustum PointNets on three sets of training data and then evaluate the model on a fixed validation set (1929 samples). The three data points in Fig. 10 represent training set sizes of 1388, 2776, 5552 samples (0.185x, 0.371x, 0.742x of the entire trainval set) respectively. We augment the training data such that the total amount of samples are the same for each of the three cases (20x, 10x and 5x augmentation respectively). The training set and validation set are chosen such that they don't share frames from the same video clips.

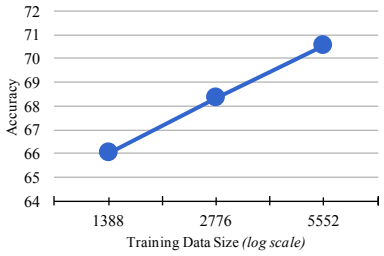

Figure 10. **Effects of training data size.** Evaluation metric is 3D box estimation accuracy (IoU threshold 0.7). We see a clear trend of linear improvement in accuracy with exponential growth of training data size.

E.3. Runtime and Model Size

In Table 14, we show decomposed runtime cost (inference time) for our frustum PointNets (v1 and v2). The evaluation is based on TensorFlow [3] with a NVIDIA GTX 1080 and a single CPU core. While for v1 model frustum proposal (with CNN and backprojection) takes the majority time, for v2 model since a PointNet++ [27] model with multi-scale grouping is used, computation bottleneck shifts to instance segmentation. Note that we merge batch normalization and FC/convolution layers for faster inference (since they are both linear operation with multiply and sum), which results in close to 50% speedup for inference.

CNN model has size 28 MB. v1 PointNets have size 19MB. v2 PointNets have size 22MB. The total size is therefore 47MB for v1 model and 50MB for v2 model.

Model	Frustum Proposal	3D Seg	Box Est.	Total
v1	60 ms	18 ms	10 ms	88 ms
v2	60 ms	88 ms	19 ms	167 ms

Table 14. **3D detector runtime.** Thirty-two region proposals used for frustum-based PointNets. 1,024 points are used for instance segmentation and 512 points are used for box estimation.

F. Visualizations for SUN-RGBD (Sec 5.1)

In Fig. 11 we visualize some representative detection results on SUN-RGBD data. We can see that compared with KITTI LiDAR data, depth images can be popped up to much more dense point clouds. However even with such dense point cloud, strong occlusions of indoor objects as well as the tight arrangement present new challenges for detection in indoor scenes.

In Fig. 12 we report the 3D AP curves of our Frustum PointNets on SUN-RGBD val set. 2D detection APs of our RGB detector are also provided in Tab. 11 for reference.

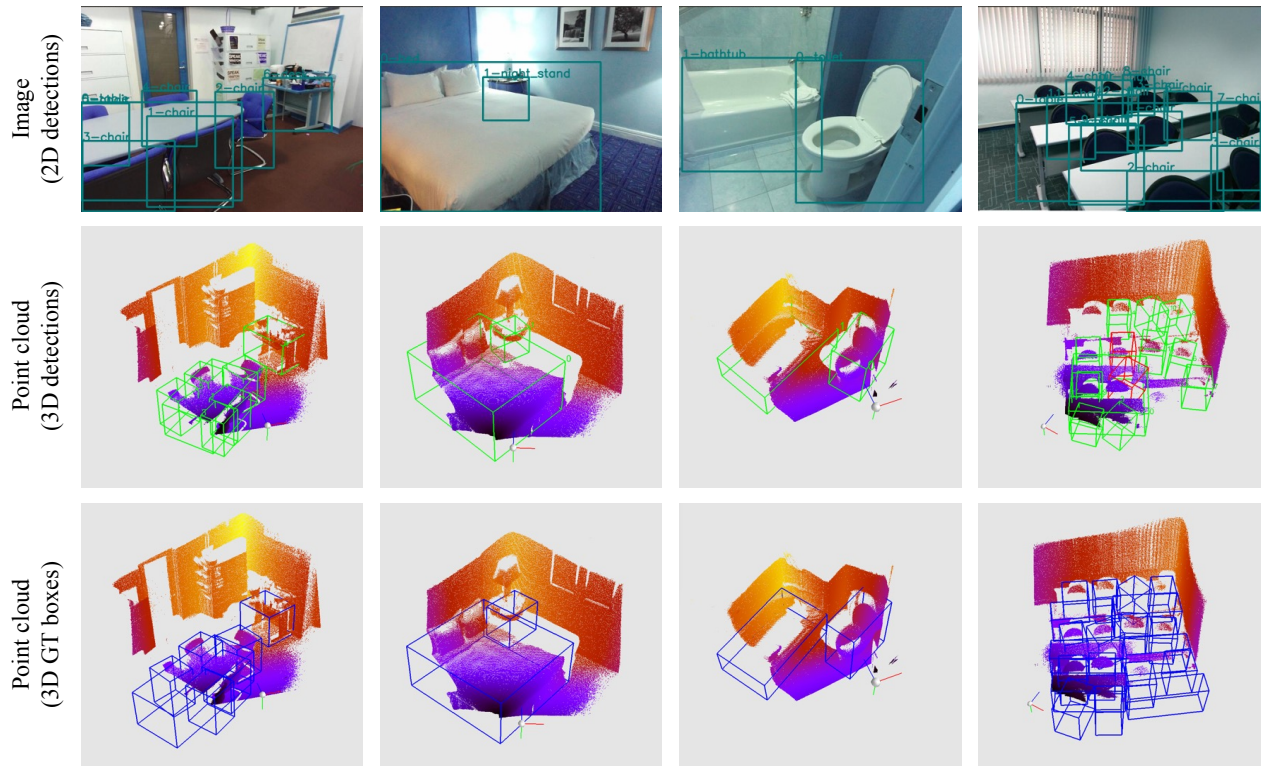

Figure 11. **Visualization of Frustum PointNets results on SUN-RGBD val set.** *First row:* RGB image with 2D detection boxes. *Second row:* point cloud popped up from depth map and predicted amodal 3D bounding boxes (the numbers beside boxes correspond to 2D boxes on images). Green boxes are true positive. Red boxes are false positives. False negatives are not visualized. *Third row:* point cloud popped up from depth map and ground truth amodal 3D bounding boxes.

Category	bathtub	bed	bookshelf	chair	desk	dresser	nightstand	sofa	table	toilet	mean
AP (2D)	81.3	56.7	67.2	64.1	77.8	33.3	37.2	57.4	49.9	43.5	50.3
AP (3D)	43.3	81.1	33.3	64.2	24.7	32.0	58.1	61.1	51.1	90.9	54.0

Table 15. **2D and 3D object detection** AP on SUN-RGBD val set. 2D IoU threshold is 0.5. Note that on some categories we get higher 3D AP (displayed in the table as well, the same results as in main paper) than 2D AP because our network is able to recover 3D geometry from very partial scan and is also due to a more loose 3D IoU threshold (0.25) in SUN-RGBD 3D AP evaluation.

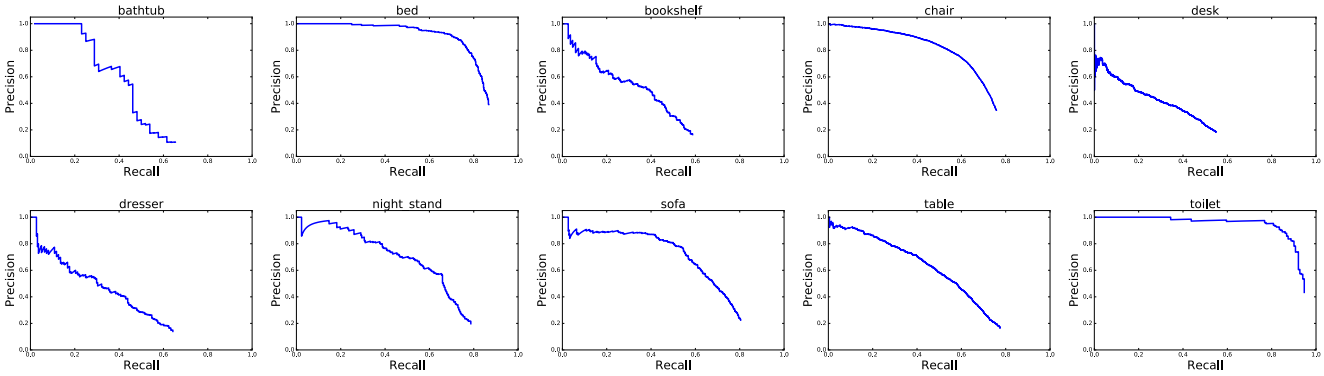

Figure 12. **Precision recall (PR) curves** for 3D object detection on SUN-RGBD val set.