DOPS: Learning to Detect 3D Objects and Predict their 3D Shapes

Mahyar Najibi[1] Guangda Lai[2] Abhijit Kundu[2] Zhichao Lu[2] Vivek Rathod[2]
Thomas Funkhouser[2] Caroline Pantofaru[2] David Ross[2] Larry S. Davis[1] Alireza Fathi[2]

[1]University of Maryland [2]Google

Abstract

We propose DOPS, a fast single-stage 3D object detection method for LIDAR data. Previous methods often make domain-specific design decisions, for example projecting points into a bird-eye view image in autonomous driving scenarios. In contrast, we propose a general-purpose method that works on both indoor and outdoor scenes. The core novelty of our method is a fast, single-pass architecture that both detects objects in 3D and estimates their shapes. 3D bounding box parameters are estimated in one pass for every point, aggregated through graph convolutions, and fed into a branch of the network that predicts latent codes representing the shape of each detected object. The latent shape space and shape decoder are learned on a synthetic dataset and then used as supervision for the end-to-end training of the 3D object detection pipeline. Thus our model is able to extract shapes without access to ground-truth shape information in the target dataset. During experiments, we find that our proposed method achieves state-of-the-art results by ∼5% on object detection in ScanNet scenes, and it gets top results by 3.4% in the Waymo Open Dataset, while reproducing the shapes of detected cars.

1. Introduction

There has been great progress in recent years on 3D object detection for robotics and autonomous driving applications. Previous work on 3D object detection takes one of these following approaches: (a) projecting LIDAR points to 2D bird's-eye view and performing 2D detection on the projected image, (b) performing 2D detection on images and using a frustum to overlap that with the point cloud, or (c) using a two-stage approach where points are first grouped together and then an object is predicted for each group.

Each of these approaches come with their own drawbacks. Projecting LIDAR to a bird's-eye view image sacrifices geometric details which may be critical in cluttered indoor environments. The frustum based approaches are strictly dependent on the 2D detector and will miss an object entirely if it is not detected in 2D. Finally, the two-stage methods introduce additional hyperparameters and design choices which require tuning and adapting for each domain separately. Furthermore, we believe grouping the points is a harder task than predicting 3D objects. Solving the former to predict the latter will result in an unnecessary upper-bound that limits the accuracy of 3D object detection.

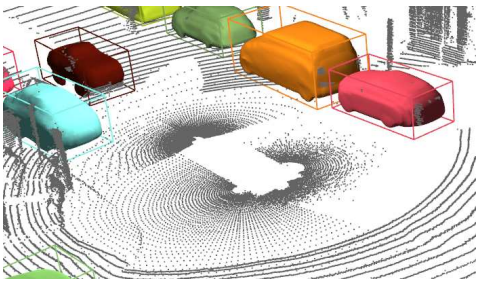

Figure 1: A sample output of our object detection pipeline.

In this paper, we propose a single-stage 3D object detection method that outperforms previous approaches. We predict 3D object properties for every point while allowing the information to flow in the 3D adjacency graph of predictions. This way, we do not make hard grouping decisions while at the same time let the information to propagate from each point to its neighborhood.

In addition to predicting 3D bounding boxes, our pipeline can also output the reconstructed 3D object shapes as depicted in Figure 1. Even though there have been various approaches proposed for predicting 3D bounding boxes, predicting the 3D shapes and extents of objects remains largely under-explored. The main challenges in predicting the 3D shapes of objects are sparsity in LIDAR scans, predominant partial occlusion, and lack of ground-truth 3D shape annotations. In this work, we address these challenges by proposing a novel weakly-supervised approach.

Our proposed solution for shape prediction consists of two steps. First, we learn 3D object shape priors using an external 3D CAD-model dataset by training an encoder that maps an object shape into an embedding representation and

a decoder that recovers the 3D shape of an object given its embedding vector. Then, we augment our 3D object detection network to predict a shape embedding for each object such that its corresponding decoded shape best fits the points observed on the surface of that object. Using this as an additional constraint, we train a network that learns to detect objects, predict their semantic labels, and their 3D shapes.

To summarize, our main contributions are as follows. First, we propose a single-stage 3D object detection method that achieves state-of-the-art results on both indoor and outdoor point cloud datasets. While previous methods often make certain design choices (*e.g*. projection to a bird-eye view image) based on the problem domain, we show the possibility of having a generic pipeline that aggregates per-point predictions with graph convolutions. By forming better consensus predictions in an end-to-end hybrid network, our approach outperforms previous works in both indoor and outdoor settings while running at a speed of 12ms per frame. Second, in addition to 3D bounding boxes, our model is also able to jointly predict the 3D shapes of the objects efficiently. Third, we introduce a training approach that does not require ground-truth 3D shape annotations in the target dataset (which is not available in large-scale self-driving car datasets). Instead, our method learns a shape prior from a dataset of CAD models and transfers that knowledge to the real-world self-driving car setup.

2. Related Works

2.1. 3D Object Detection

3D object detection has been studied extensively. In this paper, we focus on applications such as autonomous driving, where the input is a collection of 3D points captured by a LIDAR range sensor. Processing this type of data using neural networks introduces new challenges. Most notably, unlike images, the input is highly sparse, making it inefficient to uniformly process all locations in the 3D space.

To deal with this problem, PointNet [30, 31] directly consumes the 3D coordinates of the sparse points and processes the point cloud as a set of unordered points. FoldingNet [40], AtlasNet [12], 3D Point Capsule Net [44], and PointWeb [43] improve the representation by incorporating the spatial relationships among the points into the encoding process. For the task of 3D object detection, various methods rely on PointNets for processing the point cloud data. To name a few, Frustum PointNets [29] uses these networks for the final refinement of the object proposals and PointRCNN [33] employs PointNets for the task of proposal generation. VoteNet [28] deploys PointNet++ to directly predict bounding boxes from points in a two-stage voting scheme.

Projecting the point cloud data to a 2D space and using 2D convolutions is an alternative approach for reducing the computation. Bird's-eye view (BEV), front view, native range view, and learned projections are among such 2D projections. PIXOR [39], Complex YOLO [35], and Complexer YOLO [34] generate 3D bounding boxes in a single stage based on the projected BEV representation. Chen *et al*. [3] and Liang *et al*. [20] use a BEV representation and fuse its extracted information with RGB images to improve the detection performance. VeloFCN [18] projects the points to the front view and uses 2D convolutions for 3D bounding box generation. Recently, LaserNet [25] shows that it is possible to achieve state-of-the-art results while processing the more compact native range view representation. PointPillars [17], on the other hand, learns this 2D projection by training a PointNet to summarize the information of points that lie inside vertical pillars in the 3D space.

Voxelization followed by 3D convolutions is also applied to point cloud-based object detection [46]. However, 3D convolution is computationally expensive, especially when the input has a high spatial resolution. Sparse 3D convolution [7, 9, 10] is shown to be effective in solving this problem. Our backbone in this paper uses voxelization with sparse convolutions to process the point cloud.

Modeling auxiliary tasks is also studied in the literature. Fast and Furious [22] performs detection, tracking, and motion forecasting using a single network. HDNET [38] estimates high-definition maps from LIDAR sweeps and uses the geometric features to improve 3D detection. Liang *et al*. [19] performs 2D detection, 3D detection, ground estimation, and depth completion. Likewise, our system predicts the 3D shape of the objects from incomplete point clouds besides detecting the objects.

2.2. 3D Shape Prediction for Object Detection

For 3D object detection from images, 3D-RCNN [15] recovers the 3D shape of the objects by estimating the pose of known shapes. A render and compare loss with 2D segmentation annotation is used as supervision. Instead of using known shapes, Mesh R-CNN [8] first predicts a coarse voxelized shape followed by a refinement step. The 3D ground-truth information is assumed to be given. For semantic segmentation, [16] improved the generalization of unseen categories by estimating the shape of the detected objects. For 3D detection, GSPN [42] learns a generative model to predict 3D points on objects and uses them for proposal generation. ROI-10D [23] annotates ground-truth shapes offline and adds a new branch for shape prediction. In contrast, our approach does not need 3D shape ground-truth annotations in the target dataset. We use the recently proposed explicit shape modeling [27, 24, 32] to learn a function for representing a shape prior. This prior is then used as a weakly supervised signal when training the shape prediction branch on the target dataset.

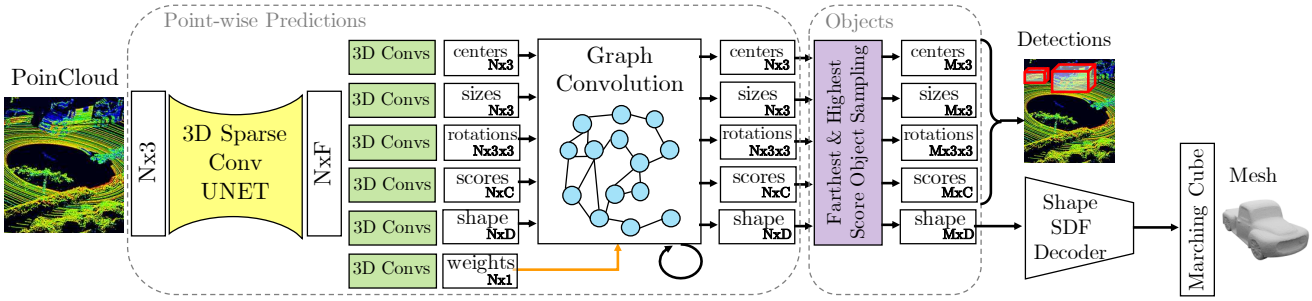

Figure 2: Object Detection Pipeline. After voxelization, a 3D sparse U-Net [11] is used to extract features from each voxel. Then two blocks of sparse convolutions predict object properties per voxel. These features are then propagated back to the points and passed through a graph convolution module. Finally, a "farthest & highest score object sampling" layer followed by NMS outputs the per-object properties including the 3D shape.

3. Approach

The overall architecture of our model is depicted in Figure 2. The model consists of four parts: The first one consumes a point cloud and predicts per point object attributes and shape embedding. The second component builds a graph on top of these per-point predictions and uses graph convolutions to transfer information across the predictions. The third component proposes the final 3D boxes and their attributes by iteratively sampling high scoring boxes which are farthest from the already selected ones. Finally, the fourth component decodes the predicted shape embeddings into SDF values which we convert to 3D meshes using the Marching Cubes algorithm [21].

3.1. Per Point 3D Object Prediction

Given a point cloud of size $N \times I$ consisting of N points with I-dimensional input features (*e.g.* positions, colors, intensities, normals), first, a 3D encoder-decoder network predicts 3D object attributes (center, size, rotation matrix, and semantic logits) and the shape embedding for every point.

We use *SparseConvNet* [11] as backbone to generate per-point features $\{f_i \in \mathbb{R}^F\}_{i=1}^N$. Each of the object attributes and the shape embedding vector are computed by applying two layers of 3D sparse convolutions on the extracted $N \times F$ features.

Box Prediction Loss: We represent a 3D object box by three properties: size (length, width, height), center location (c_x, c_y, c_z), and a 3x3 rotation matrix. Given these predictions, we use a differentiable function to compute the eight 3D corners of each predicted box. We apply a Huber loss on the distance between predicted and the ground-truth corners. The loss will automatically propagate back to the size, center and rotation variables.

To compute the rotation matrix, our network predicts 6 parameters: (cos_x, sin_x, cos_y, sin_y, cos_z, sin_z). Then we formulate the rotation matrix as $R = R_x \times R_y \times R_z$.

The benefit of using this loss in comparison to separate losses for rotation, center, and size is that we do not need to tune the relative scale among multiple losses. Our box corner loss propagates back to all and minimizes the predicted corner errors. We define the per-point box corner regression loss as

$$\mathcal{L}_{\text{corner}}(P, G) = \frac{1}{8 \times \sum_{i=1}^{N} \mathbb{1}(x_i)} \sum_{i=1}^{N} \mathbb{1}(x_i) \sum_{j=1}^{8} \left\| p_i^{(j)} - g_i^{(j)} \right\|_H \quad (1)$$

where $|| \cdot ||_H$ is the *Huber*-loss (*i.e.* smooth L_1-loss), and $\mathbb{1}(.)$ is binary function indicating whether a point x_i is on an object surface. P and G are the sets of predicted and ground-truth corners in which $p_i^{(j)}$ represents the j'th predicted corner for point i, and $g_i^{(j)}$ denotes the corresponding ground-truth corner.

Dynamic Classification Loss: Every point in the point cloud predicts a 3D bounding box. The box prediction loss forces each point to predict the box that it belongs to. Some of the points make more accurate box predictions than others. Thus we design a classification loss that classifies points that make accurate predictions as positive and others as negative. During the training stage, at each iteration, we compute the IoU overlap between predicted boxes and ground-truth matches and classify the points that have an IoU more than 70% as positive and the rest as negative. This loss gives us a few percent improvements in comparison to regular classification loss (where we would label points that fall inside an object as positive and points outside as negative). We use a softmax loss for classification.

3.2. Object Proposal Consolidation

Each point predicts its object center, size, and rotation matrix. We create a graph where the points are the nodes, and each point is connected to its K nearest neighbors in

the center space. In other words, each point is connected to those with similar center predictions. We perform a few layers of graph convolution to consolidate the per-point object predictions. A weight value is estimated per point by the network which determines the significance of the vote a point casts in comparison to its neighbors. We update each object attribute predicted by points as follows:

$$a_x = \frac{\Sigma_{y \in \mathcal{N}_x} w_y . a_y}{\Sigma_{y \in \mathcal{N}_x} w_y} \tag{2}$$

where a_x is an object attribute (*e.g.* object length) predicted for point x, $\mathcal{N}_x$ is the set of neighbors of x in the predicted center space, and w_y is the weight predicted for point y.

We apply the bounding box prediction loss both before and after the graph convolution step to let the network learn a set of weights that make final predictions more accurate. In this way, instead of directly applying a loss on the predicted point weights, the network automatically learns to assign larger weights to more confident points.

3.3. Proposing Boxes

Our network predicts a 3D object box and a semantic score for every point. During the training stage, we apply the losses directly to the per point predictions. However, during the evaluation, we need to use a box proposal mechanism that can reduce the hundreds of thousands of box predictions into a few accurate box proposals. We can greedily pick boxes with high semantic scores. However, we also want to encourage spatial diversity in the locations of the proposed boxes. For this reason, we compute the distance between each predicted box center and all previously selected boxes and choose the one that is far from the already picked points (similar to the heuristic used by KMeans++ initialization [1]). More precisely, at step t, given predicted boxes for previous steps $\mathcal{B}_{1:t-1}$, we select a seed point as follows:

$$b_t = \arg \max_{b \notin \mathcal{B}_{1:t-1}} [\log(s_b) + \alpha \log(D(b, \mathcal{B}_{1:t-1})]$$

where

$$D(b, \mathcal{B}_{1:t-1}) = \min_{b' \in \mathcal{B}_{1:t-1}} \|b - b'\|$$

and s_b represents the foreground semantic score of box b. Selecting boxes with high foreground semantic score guarantees high precision, and selecting diverse boxes guarantees high recall. Note that our sampling strategy is different from the non-maximum suppression algorithm. In NMS, boxes that have a high IoU are suppressed and are not redeemable, while in our algorithm, we can tune the balance between confidence and diversity.

3.4. Shape Prediction

To predict shapes, first, we learn a shape prior function from an external synthetic 3D dataset of CAD models as discussed in Section 3.4.1. Then we deploy our learned prior to recover 3D shapes from the embedding space predicted by the object detection pipeline.

3.4.1 Modeling the Shape Prior

There are various ways to represent a shape prior. For our application, given that a shape embedding vector should be predicted for each point in the point cloud, the representation needs to be compact. We use an encoder-decoder architecture with a compact bottleneck to model the shape prior. The general framework is depicted in Figure 3.

The shape encoder consumes the point cloud of an object after data augmentation techniques (*e.g.* random cropping) and then outputs a compact shape embedding vector. The point cloud representation of the object is first voxelized and then forwarded through an encoder network. The network consists of three convolutional blocks, each having two 3D sparse convolution layers intervened by *BatchNorm* and *ReLU* layers (not shown in the figure for simplicity). The spatial resolution of the feature maps is reduced by a factor of two after each convolutional block. Finally, a fully-connected layer followed by a global average pooling layer output the embedding vector of the input shape.

For shape decoding, we represent the shape as a level set of an implicit function [24, 32, 27]. That is, the shape is modeled as the level set zero of a signed distance field (SDF) function over a unit hyper-cube. Following [24], we rely on Conditional Batch Normalization[5, 6] layers to condition the decoder on the predicted embedding vector. The input to the decoder is a batch of 3D coordinates of the query points. After five conditional blocks, a fully connected layer followed by a *tanh* function predicts the signed distance of each query from the surface of the object in a canonical viewpoint.

During training, we sample some query points close to the object surface and some uniformly in the unit hypercube surrounding the object to predict their SDF values. However, as suggested in [32], we regress towards discrete label values to capture more detail near the surface boundaries. More precisely, given a batch of training queries $Q = \{q_i\}_{i=1}^N \in \mathbb{R}^{3 \times N}$, their corresponding ground-truth signed distance values $S = \{s_i\}_{i=1}^N \in \mathbb{R}^N$, and their predicted embedding vectors $E = \{e_i\}_{i=1}^N \in \mathbb{R}^{D \times N}$, the loss is defined as:

$$\mathcal{L}(Q, S, E|f) = \frac{1}{N} \sum_{i=1}^{N} \|f(q_i|e_i) - sign(s_i)\|^2 \tag{3}$$

where $f(.)$ is the conditional decoder function and $sign(.)$ is the sign function.

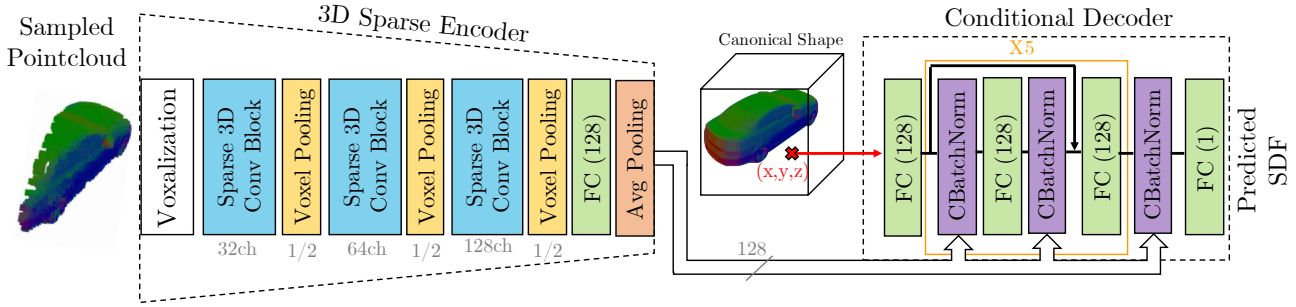

Figure 3: Shape Prior Network Architecture. The encoder consumes the point cloud representation of an object after augmentations (*e.g.* random cropping) and outputs a compact embedding vector. The decoder consists of Conditional Batch Norm [5] layers which are conditioned on the predicted embeddings. The input to the decoder is a batch of 3D point coordinates and the output is the predicted signed distance of each point to the object surface.

3.4.2 Training the Shape Prediction Branch

Although there is no ground-truth 3D shape annotation available in detection datasets collected for applications such as autonomous driving, once trained, the learned prior model can be deployed to enforce shape constraints. That is, for each object in the incomplete point cloud, we expect that most of the observed points in its bounding box lie on its surface.

To predict the shape embedding, we add a branch to the object detection pipeline to predict a D-dimensional vector per point. The shape embeddings for all points belonging to an object is then averaged pool to form its shape representation. To enforce the constraints, we freeze the 3D decoder in Figure 3 and discard the encoder. Conditioned on the predicted shape embedding and given some shape queries per object, the frozen shape decoder should be able to predict the signed distances.

To define the queries, for each object present in the point cloud, we subtract the object center and rotate the queries to match the canonical orientation used during training the shape prior network. Then, the queries are projected into a unit hyper-cube. We also pre-process them by removing points on the ground and augmenting the symmetrical points (if the object is symmetrical). Finally, as the shape prior is trained with discrete sign labels, we sample some number of queries on the ray connecting the object center to each of the observed points and assign -1/+1 labels to inside/outside queries respectively (in this paper we sample two points with distance $\delta = 0.1$ to each observed point along the rays). During training, we also optimize the loss defined in Eq. 3 for objects with a reasonable number of points observed (*i.e.* minimum of 500 points in this paper.)

3.5. Achieving Real-Time Speed

Our 3D sparse feature extractor with 30 3D sparse convolution layers, 7 3D sparse pooling layers, and 7 3D sparse un-pooling layers achieves a speed of $12ms$ per frame on Waymo Open dataset (with around 200k input points per frame). Here we describe the implementation details of our Tensorflow sparse GPU operations.

We use CUDA to implement the submanifold sparse convolution [11] and sparse pooling GPU operations in TensorFlow. Since the input to the convolution operation is sparse, we need a mechanism to get all the neighbors for each non-empty voxel. We implemented a hashmap to do that, where the keys are the XYZ indices of the voxels, and the values are the indices of the corresponding voxels in the input voxel array. We use an optimized spatial hash function[37]. Our experiments on the Waymo Open dataset shows that with a load factor of 0.42, the average collision rate is 0.18. We precompute the neighbor indices for all non-empty voxels and reuse them in one or more subsequent convolution operations. We use various CUDA techniques to speed up the computation (*e.g.* partitioning and caching the filter in shared memory and using bit operations).

Both 3D sparse max pooling and 3D sparse average pooling operations are implemented in CUDA. Since each voxel needs to be looked up only once during pooling, we do not reuse the convolution hashmap that can introduce redundant lookups. Instead, we compute the pooled XYZ indices and use them as the key to building a new "hashmultimap"(multiple voxels can be pooled together thus having the same key), and shuffle the voxels based on the keys. Our experiments show that this approach is more than 10X faster than the radix sort provided by the CUB library. Furthermore, since our pooling operation does not rely on the original XYZ indices, it has the ability to handle duplicate input indices. This allows us to use the same operation for voxelizing the point cloud, which is the most expensive pooling operation in the network. Our implementation is around 20X faster than a well-designed implementation with pre-existing TensorFlow operations.

4. Experiments

4.1. Experimental Setup

For our object detection backbone, we use an encoder-decoder UNET with sparse 3D convolutions. The encoder consists of 6 blocks of 3D sparse convolutions, each of which having two 3D sparse convolutions inside. Going deeper, we increase the number of channels gradually (*i.e.* 64, 96, 128, 160, 192, 224, 256 channels). We also apply a 3D sparse pooling operation after each block to reduce the spatial resolution of the feature maps. For the decoder, we use the same structure but in the reverse order and replace the 3D sparse pooling layers with unpooling operations. Two 3D sparse convolutions with 256 channels connect the encoder and decoder and form the bottleneck. Models are trained on 20 GPUs with a batch size of 6 scenes per each. We use stochastic gradient descent with an initial learning rate of 0.3 and drop the learning rate every 10K iterations by the factors of [1.0, 0.3, 0.1, 0.01, 0.001, 0.0001]. We use a weight decay of 5×10^{-4} and stop training when the loss plateaus. We use random rotations of (-10, 10) degrees along the z-axis and random scaling of (0.9, 1.1) for data augmentation.

The 3D sparse encoder in our shape prior network consists of three convolutional blocks with two 3D sparse convolutions in each. We use an embedding size of 128 dimensions and set $((32, 64), (64, 128), (128, 128))$ as the number of channels in the 3D convolutional layers. We downsample the feature maps by a factor of 2 after each block. A global average pooling, followed by a fully-connected layer outputs the predicted embedding. Our shape decoder consists of five conditional blocks with two 128 dimensional fully connected layers intervened by conditional batch normalization layers. A *tanh* function maps predictions to [-1, +1]. We train our model with an initial learning rate of 0.1 with the same step-wise learning rate schedule used for training the detection pipeline.

4.2. Datasets

ScanNetV2 [4] is a dataset of 3D reconstructed meshes of around 1.5K indoor scenes with both 3D instance and semantic segmentation annotations. The meshes are reconstructed from RGB-D videos that are captured in various indoor environments. Following the setup in [28], we sample vertices from the reconstructed meshes as our input point clouds and since ScanNetV2 does not provide amodal or oriented bounding box annotations, we predict axis-aligned bounding boxes instead, as in [28, 14].

Waymo Open Dataset [26, 45] is a large scale self-driving car dataset, recently released for benchmarking 3D object detection. The dataset captures multiple major cities in the U.S., under a variety of weather conditions and across different times of the day. The dataset contains a total of 1000 sequences, where each sequence consists of around 200 frames that are 100 ms apart. The training split consists of 798 sequences containing 4.81M vehicle boxes. The validation split consists of 202 sequences with the same duration and sampling frequency, containing 1.25M vehicle boxes. The effective annotation radius in the Waymo Open dataset is 75m for all object classes. For our experiments, we evaluate 3D object detection metrics for vehicles and predict 3D shapes for them.

	Input	mAP@0.25	mAP@0.5
DSS [36, 14]	Geo + RGB	15.2	6.8
MRCNN 2D-3D [13, 14]	Geo + RGB	17.3	10.5
F-PointNet [29, 14]	Geo + RGB	19.8	10.8
GSPN [41]	Geo + RGB	30.6	17.7
3D-SIS [14]	Geo + 1 view	35.1	18.7
3D-SIS [14]	Geo + 3 views	36.6	19.0
3D-SIS [14]	Geo + 5 views	40.2	22.5
3D-SIS [14]	Geo only	25.4	14.6
DeepVote[28]	Geo only	58.6	33.5
DOPS (ours)	Geo only	**63.7**	**38.2**

Table 1: 3D object detection results on ScanNetV2 validation set. We report results for other approaches as appeared in the original papers or provided by the authors.

4.3. Object Detection on ScanNetV2

We present our object detection results on the ScanNetV2 dataset in Table 1. For this dataset, we follow [28, 14] and predict axis-aligned bounding boxes. Although we only use the available geometric information, we also compare the proposed method with approaches that use the available RGB images and different viewpoints. Our approach noticeably improves the state-of-the-art by 3% and 4.6% with respect to mAP@0.25 and mAP@0.5 metrics. We also report our per-category results on the ScanNetV2 dataset in Table 2. Figure 7 shows our qualitative results.

4.4. Object Detection on Waymo Open

We achieve an mAP of **56.4**% at IOU of 0.7. This is while StarNet [26] achieves an mAP of 53.0%. Note that [45] also reports 3D object detection results on the Waymo open dataset. However, their results are not directly comparable to ours since they fuse 2D networks applied to multiple views in addition to a 3D network. Since our detection pipeline consists of different parts, we also perform our ablation studies on this dataset. Table 3 shows the contribution of each component of the system on its overall performance. Each column shows the performance when a single component of the system is excluded and the rest remain the same. Removing graph convolution over the predictions on the neighborhood graph reduces the detection performance

	Bathtub	Bed	Book Shelf	Cabinet	Chair	Counter	Curtain	Desk	Door	Other	Picture	Refrig.	Shower Curtain	Sink	Sofa	Table	Toilet	Window	Overall Score
mAP@0.25	86.6	83.3	41.0	53.2	91.6	51.9	53.9	73.7	54.8	59.2	26.3	49.2	64.7	71.3	82.6	60.5	98.0	45.2	63.7
mAP@0.5	71.0	70.2	21.4	25.2	75.8	9.5	24.4	39.4	27.8	35.0	12.3	33.7	17.3	35.7	54.8	41.2	80.6	12.1	38.2

Table 2: Per-category results on ScanNetV2. We report mAP at IoU of 25% and 50%.

by $\sim 2\%$, showing its importance. Replacing the dynamic classification loss with a regular classification loss drops the performance by 3.3%. Finally, if instead of the farthest and highest object sampling, one directly deploys NMS to form the objects, the performance drops by 0.7%. We also noticed that shape prediction does not have a noticeable impact on the detection precision. We believe the main reason is that the Waymo Open dataset has manually labeled bounding boxes for object detection, but no ground-truth shape annotations. As a result, the shape predictions are supervised only with noisy, partial, and sparse LIDAR data, which provides a relatively weaker training signal.

	DOPS (ours)	w/o Graph Convolution	w/o Dynamic Cls Loss	w/o Farthest & Highest Sampling
mAP@0.7	**56.4**	54.5	53.1	55.7

Table 3: The contribution of each compontent on the overal accuracy on the Waymo Open Dataset.

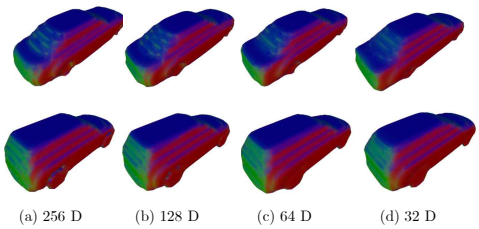

Figure 4: Shapes recovered in ShapeNet dataset from the learned embedding by Marching Cube [21] on a 100^3 SDF volume. Our prior network captures the shape information even using a low-dimensional embedding vector.

4.5. 3D Shape Prediction on Waymo Open

To model shape, we first learn a prior from the synthetic ShapeNet dataset [2]. Figure 4 shows shapes recovered from the compact embedding vectors predicted for CAD models in ShapeNet. Each row represents one shape and columns show the results for different embedding dimensions. We use marching cube [21] with a resolution of 100 points per side on SDF values predicted by our decoder for a uniform hyper-cube surrounding the object. As can be seen, the decoder can recover the extent of the object from the predicted embedding vector, even when the dimensionality of the embedding space is low.

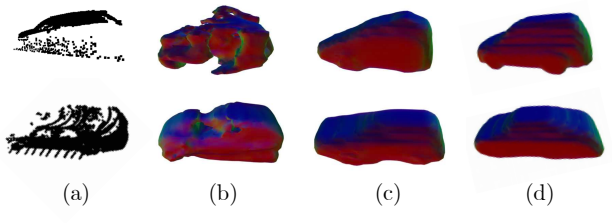

Figure 5: Ablations for shape fitting on observed points in Waymo dataset. (a) Observed points. (b) Enforcing the decoder to predict zero SDFs only for the observed points. (c) Adding two points along the rays passing through the observations and object center inside/outside the object with a distance of $\delta = 0.5$. (d) Decreasing δ to 0.1.

Once trained on the ShapeNet dataset, we freeze the decoder and use it to recover shapes from the observed points in the real-world scenes captured by the LIDAR sensors. However, compared to the synthetic CAD models, LIDAR points are incomplete, noisy, and the distribution of the observed points can be different from the clean synthetic datasets. Consequently, we found proper preprocessing and data augmentation techniques crucial. Noticeably, ShapeNet contains dense annotations even for surfaces inside the objects. However, when it comes to autonomous driving datasets, only a sparse set of points on the surface of the object is observed. We remove internal points when training on the ShapeNet dataset and empirically noticed that this step improves convergence and shape prediction quality. Moreover, the LIDAR sensor frequently captures points on the ground while this does not happen in ShapeNet. We also remove points on the ground based on the coordinate frame of each object.

Given a set of N observed points in the point cloud, a predicted encoding vector, and a frozen decoder, it is possible to enforce N weakly supervised constraints to recover the shapes. The points which are observed should lie on the surface of the object with a high probability. That is, the frozen decoder conditioned on the predicted embedding should predict a zero SDF value for these points. However, this set of constraints is not enough for reliably recovering the shape. Figure 5b shows the case when a shape is fitted to a set of of points observed from an object in the Waymo Open dataset, shown in 5a. As can be seen, the decoder is able to fit a complex surface to the points. This is while the

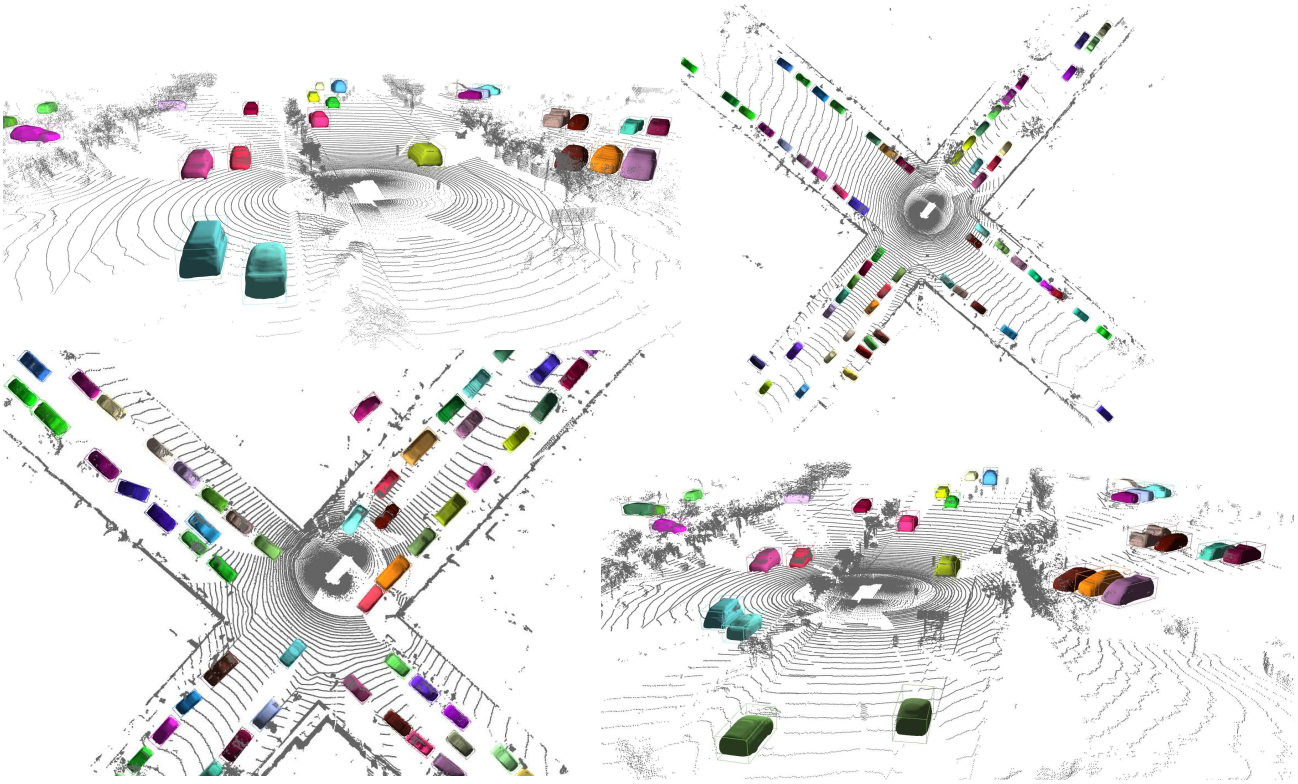

Figure 6: Qualitative results of 3D object detection and 3D shape prediction.

Figure 7: Qualitative results for the axis aligned object dtection on the ScanNet dataset.

shape almost perfectly passes through the observed points.

Instead, we augment points with additional ones sampled along the ray connecting the observations to the object center. For each observed point, we add two points on this ray inside and outside the object with distance δ from the surface and assign labels -1/+1 to them respectively. Figures 5c, and 5d show the shape fitting when we set δ to 0.5 and 0.1 respectively. As can be seen, this augmentation technique is crucial and sampling closer points increases the quality of the recovered shape.

Finally, Figure 6 presents our end-to-end shape prediction results. Note that the car shapes fit the point cloud and are not simply copies of examples from a database.

5. Conclusions

We propose DOPS, a single-stage object detection system which operates on point cloud data. DOPS directly predicts object properties for each point. Instead of grouping points before prediction, a graph convolution module is deployed to aggregate the information across neighboring points. For a more accurate localization, it also outputs a 3D mesh using a shape prior learned on a synthetic dataset of CAD models. We show state-of-the-art results for on 3D object detection datasets for both indoor and outdoor scenes. Topics for future work include detection and tracking over time, semi-supervised training of shape priors, and extending shape models to handle non-rigid objects.

References

[1] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. *Proc. symposium on discrete algorithms*, 2007. 4

[2] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, and H. Su. Shapenet: An information-rich 3d model repository. In *arXiv:1512.03012*, 2015. 7

[3] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017. 2

[4] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 6

[5] Harm De Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C Courville. Modulating early visual processing by language. In *Advances in Neural Information Processing Systems*, pages 6594–6604, 2017. 4, 5

[6] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016. 4

[7] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1355–1361. IEEE, 2017. 2

[8] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn. *arXiv preprint arXiv:1906.02739*, 2019. 2

[9] Ben Graham. Sparse 3d convolutional neural networks. In Gary K. L. Tam Xianghua Xie, Mark W. Jones, editor, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 150.1–150.9. BMVA Press, September 2015. 2

[10] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018. 2

[11] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017. 3, 5

[12] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Atlasnet: A papier-m\^ ach\'e approach to learning 3d surface generation. *arXiv preprint arXiv:1802.05384*, 2018. 2

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017. 6

[14] Ji Hou, Angela Dai, and Matthias Nießner. 3d-sis: 3d semantic instance segmentation of rgb-d scans. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2019. 6

[15] Abhijit Kundu, Yin Li, and James M Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3559–3568, 2018. 2

[16] Weicheng Kuo, Anelia Angelova, Jitendra Malik, and Tsungyi Lin. Shapemask: Learning to segment novel objects by refining shape priors. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. 2

[17] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019. 2

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

[19] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7345–7353, 2019. 2

[20] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 641–656, 2018. 2

[21] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987. 3, 7

[22] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3569–3577, 2018. 2

[23] Fabian Manhardt, Wadim Kehl, and Adrien Gaidon. Roi-10d: Monocular lifting of 2d detection to 6d pose and metric shape. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2069–2078, 2019. 2

[24] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 2, 4

[25] Gregory P Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K Wellington. Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12677–12686, 2019. 2

[26] Jiquan Ngiam, Benjamin Caine, Wei Han, Brandon Yang, Yuning Chai, Pei Sun, Yin Zhou, Xi Yi, Ouais Alsharif, Patrick Nguyen, Zhifeng Chen, Jonathon Shlens, and Vijay Vasudevan. Starnet: Targeted computation for object detection in point clouds. In *arXiv:1908.11069*, 2019. 6

[27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *arXiv preprint arXiv:1901.05103*, 2019. 2, 4

[28] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019. 2, 6

[29] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018. 2, 6

[30] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 2

[31] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017. 2

[32] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. *arXiv preprint arXiv:1905.05172*, 2019. 2, 4

[33] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019. 2

[34] Martin Simon, Karl Amende, Andrea Kraus, Jens Honer, Timo Samann, Hauke Kaulbersch, Stefan Milz, and Horst Michael Gross. Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. 2

[35] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *European Conference on Computer Vision*, pages 197–209. Springer, 2018. 2

[36] Shuran Song and Jianxiong 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. 6

[37] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. Optimized spatial hashing for collision detection of deformable objects. In *Vmv*, volume 3, pages 47–54, 2003. 5

[38] Bin Yang, Ming Liang, and Raquel Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In *Conference on Robot Learning*, pages 146–155, 2018. 2

[39] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018. 2

[40] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 206–215, 2018. 2

[41] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas Guibas. Gspn: Generative shape proposal network for 3d instance segmentation in point cloud. *arXiv preprint arXiv:1812.03320*, 2018. 6

[42] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas J Guibas. Gspn: Generative shape proposal network for 3d instance segmentation in point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3947–3956, 2019. 2

[43] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5565–5573, 2019. 2

[44] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3d point capsule networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1009–1018, 2019. 2

[45] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *Conference on Robot Learning (CoRL)*, 2019. 6

[46] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018. 2