Object DGCNN: 3D Object Detection using Dynamic Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

3D object detection often involves complicated training and testing pipelines, which require substantial domain knowledge about individual datasets. Inspired by recent non-maximum suppression-free 2D object detection models, we propose a 3D object detection architecture on point clouds. Our method models 3D object detection as message passing on a dynamic graph, generalizing the DGCNN framework to predict a set of objects. In our construction, we remove the necessity of post-processing via object confidence aggregation or non-maximum suppression. To facilitate object detection from sparse point clouds, we also propose a set-to-set distillation approach customized to 3D detection. This approach aligns the outputs of the teacher model and the student model in a permutation-invariant fashion, significantly simplifying knowledge distillation for the 3D detection task. Our method achieves state-of-the-art performance on autonomous driving benchmarks. We also provide abundant analysis of the detection model and distillation framework.

Methods for 3D object detection have progressed rapidly, yielding deployable autonomous driving perception systems. Following common practice in 2D vision, 3D object detection often employs complex training and testing pipelines including many post-processing operations to achieve superior performance. These operations are typically non-parallelizable and inefficient even with modern deep learning frameworks, implying a steep trade-off between between efficiency and effectiveness.

Modern methods usually employ two stages [1, 2], including a region proposal network [3] that can introduce significant training overhead. Subsequent efforts simplify this pipeline for 3D object detection. PointPillars [4] introduces a one-stage anchor-based design, simplfying training. PillarOD [5] and CenterPoint [6] improve the one-stage model by making per-pillar predictions, that is, one prediction per point on the ground plane. They assign ground-truth bounding boxes to multiple outputs while training to ease optimization. However, they predict redundant boxes, which can overlap in the same positions; extra boxes are eliminated *a posteriori* using non-maximum suppression (NMS). It remains elusive to remove hand-designed components like NMS in training and testing.

We introduce Object DGCNN, a streamlined architecture for 3D object detection from point clouds. Like DETR for 2D object detection [7], we predict a set of bounding boxes from the raw data, enabling an NMS-free pipeline that achieves real-time performance. A critical new component is to treat each object query as a point in a set whose embedding is learned using DGCNN [8]. Compared to the self-attention module [9] in DETR, DGCNN leverages a *sparse* set of object relations, which reflects the real object distribution in the scene. In contrast to PointPillars [4], PillarOD [5], and CenterPoint [6], our method *does not* require post-processing.

We also provide a knowledge distillation approach customized to 3D object detection. Existing methods typically distill dense feature maps from a teacher model to a student model, whose training objective does not necessarily capture 3D object detection performance [10]. In contrast, we propose set-to-set distillation training that aligns the outputs of the teacher and the student in a permutation-

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

invariant fashion. This process is enabled by the unified Object DGCNN architecture. In addition to obtaining better performance, through this process our model can benefit from privileged information (e.g., dense point clouds) only available at training time.

Contributions. We summarize our key contributions as follows:

- We propose a post-processing-free 3D object detection model achieving state-of-the-art performance. To our knowledge, this is the first NMS-free 3D object detector.
- We generalize DGCNN to model objects as a point set. The DGCNN module outperforms its self-attention counterpart thanks to its sparse structure.
- We propose a set-to-set distillation method for 3D object detection. In our construction, knowledge distillation on object detection simply penalizes differences between the outputs of the teacher model and the student model.
- We show our model can use privileged information (such as dense point clouds) that is naturally available at training time to improve the model performance at inference time.
- We will release our code to promote reproducibility and future research.

1 Related Work

2D object detection. Object recognition research has been transitioning from models with hand-crafted components to models with limited post-processing. One-stage detectors [11, 12, 13] remove the complicated region proposal networks in two-stage objectors [3, 14], yielding more efficient training and testing. Anchor-free methods [15, 16] further simplify the one-stage pipeline by shifting from per-anchor prediction to per-pixel prediction. However, these methods still make dense predictions and rely on NMS to reduce redundancy. To alleviate this issue, DETR [7] formulates object detection as a set-to-set prediction problem. It introduces a set-to-set loss that implicitly penalizes redundant boxes, removing the necessity of post-processing. To accelerate convergence, Deformable DETR [17] proposes deformable self-attention and streamlines the optimization process. Our method also formulates 3D object detection as set prediction, but with a customized design for 3D.

3D object detection. VoxelNet [18] generalizes one-stage object detection to 3D. It uses 3D dense convolutions to learn representations on voxelized point clouds, which is too inefficient to capture fine-grained features. To address that, PIXOR [19] and PointPillars [4] project points to a birds-eye view (BEV) and operate on 2D feature maps; PointNet [20] aggregates features within each BEV pixel. We use a variant of PointPillars [4] for 3D detection (§3). These methods are efficient but drop information along the vertical axis. To accompany the BEV projection, MVF [21] adds a spherical projection. PillarOd [5] and CenterPoint [6] use pillar-centric object detection, making predictions per BEV pixel (pillar) rather than per anchor. These anchor-free methods simplify 3D object detection while maintaining efficiency. Beyond SSD-style [11] one-stage models, Complex-YOLO [22] extends YOLO to 3D for real-time perception. PointRCNN [23] employs a two-stage architecture for high-quality detection. To improve representations of two-stage models, PVRCNN [2] proposes a point-voxel feature set abstraction layer to leverage the flexible receptive fields of PointNet-based networks. Unlike works on point clouds, LaserNet [24] operates on raw range scans with comparable performance. [25, 26, 27] combine point clouds with camera images. Frustum-PointNet [28] leverages 2D object detectors to form a frustum crop of points and then uses PointNet to aggregate features. [29] describes an end-to-end learnable architecture that exploits continuous convolutions to fuse feature maps. VoteNet [30, 31] generalizes Hough voting [32] for 3D object detection in point clouds. DOPS [33] extends VoteNet and predicts 3D object shapes. In addition to visual input, [34] shows that high-definition (HD) maps can boost performance of 3D object detectors. [35] argues that multi-tasking can learn better representations than single-tasking. Beyond supervised learning, [36] learns a perception model for unknown classes.

DGCNN. DGCNN [8] pioneered learning point cloud representations via dynamic graphs. It models point clouds as connected graphs, which are dynamically built using k-nearest neighbors in the latent space. DGCNN learns per-point features through message passing. However, it operates on point clouds for single object recognition and semantic segmentation. One of our key contributions is to generalize DGCNN to model scene-level object relations for 3D detection.

Knowledge distillation (KD). KD compresses knowledge from an ensemble of models into a single smaller model [37]. [38] generalizes this idea and combines it with deep learning. KD transfers knowledge from a teacher model to a student model by minimizing a loss, in which the target is the

distribution of class probabilities induced by the teacher. [39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50] improve knowledge distillation for classification. Beyond image classification, KD has been extended to improve object detection. [51] leverages FitNets for object detection, addressing obstacles such as class imbalance, loss instability, and feature distribution mismatch. [52] distills between region proposals, accelerating training with added instability. To address this issue, [53] uses fine-grained representation imitation using object masks. [54] uses KD to tackle a continual learning problem.

Privileged information. [55] introduces the framework of learning with privileged information in the context of support vector machines (SVMs), wherein additional information is accessible during training but not testing. [56] unifies KD and learning using privileged information theoretically. [57] identifies practical applications, e.g., transferring knowledge from localized data to non-localized data, from high resolution to low resolution, from color images to edge images, and from regular images to distorted images. To mediate uncertainty and improve training efficiency, [58] makes the variance of Dropout [59] a function of privileged information. We extend these methods to 3D data, in which privileged information consists of dense point clouds aggregated from LiDAR sequences.

2 Overview

Our target application of object detection differs from the recognition and segmentation tasks considered for DGCNN. Our point clouds typically contain too many points to apply DGCNN and its peers directly to the entire scene. Moreover, the size of our output set, a small set of bounding boxes, differs from the size of our input set, a huge set of points in $\mathbb{R}^3$.

Following state-of-the-art in large-scale object detection, our pipeline learns a grid-based intermediate representation to capture local features (§3). We test two standard learning-based methods for collecting local point cloud features on a birds-eye view (BEV) grid. While in principle it might be possible to avoid grids entirely in our pipeline, this BEV representation is far more efficient and—as observed in previous work—is sufficient to find objects reliably in autonomous driving, where there is likely only one object above any given grid cell on the ground plane.

Our main architecture contribution is the Object DGCNN pipeline (§4), which transitions from this BEV grid of features to a *set* of object bounding boxes. Object DGCNN draws inspiration from the DGCNN architecture; its layers alternate between local feature transformations and k-nearest neighbor aggregation to capture relationships between objects. Unlike conventional DGCNN, however, Object DGCNN incorporates features from the BEV grid in each of its layers; each layer incorporates several queries into the BEV to refine object position estimates. The output of Object DGCNN is a *set* of objects in the scene. We use a permutation-invariant loss (10) to measure divergence from the ground truth set of objects.

The pipeline above does not require hand-designed post-processing like NMS; our output boxes are usable directly for object detection. Beyond simplifying the object detection pipeline, this allows us to propose object detection-specific distillation procedures (§6.3) that further improve performance. These use one network to train another, e.g., to train a network operating on sparse point clouds to output features that imitate those learned by a network trained on denser, more detailed point clouds.

3 Local Features

We begin with a point cloud $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_N\} \subset \mathbb{R}^3$ with per-point features $\mathcal{F} = \{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_i, \ldots, \boldsymbol{f}_N\} \subset \mathbb{R}^K$, ground-truth bounding boxes $\mathcal{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_j, \ldots, \boldsymbol{b}_M\} \subset \mathbb{R}^9$, and categorical labels $\mathcal{C} = \{c_j, \ldots, c_j, \ldots, c_M\} \subset \mathbb{Z}$. Each $\boldsymbol{b}_j$ contains position, size, heading angle, and velocity in the birds-eye view (BEV); our architecture aims to predict these boxes and their labels from the point cloud and its features.

As an initial step, modern 3D object detection models scatter points into either BEV pillars or 3D voxels and then use convolutional neural networks to extract features on a grid. This strategy accelerates object detection for large point clouds. We test two neural network architectures for BEV feature extraction, detailed below.

PointPillars [4] maps sparse point clouds onto a dense BEV pillar map on which 2D convolutions can be applied. Suppose $F_P(i)$ returns the points in pillar i, that is, the set of points in a vertical column above point i on the ground. When collecting features from points to pillars, multiple points can fall

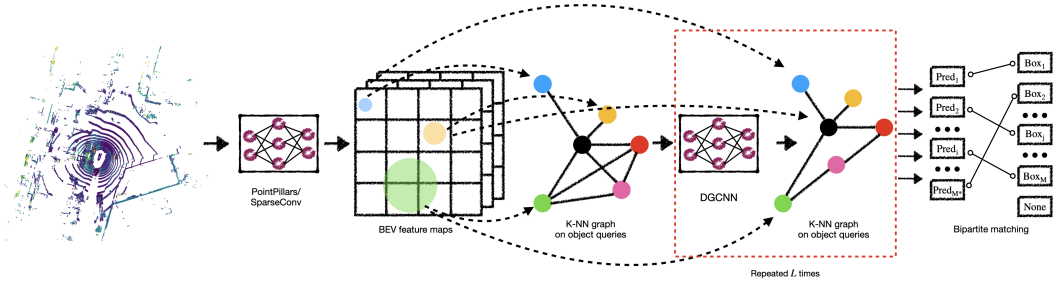


Figure 1: Overview. Point cloud features are learned in BEV, followed by L DGCNNs to model object relations. We predict a set of bounding boxes and compute loss in a one-to-one manner.

into the same pillar. In this case, PointNet [20] (PN) is used to obtain pillar-wise features:

$$\boldsymbol{f}_i^{\text{pillar}} = \text{PN}(\{\boldsymbol{f}_j | \boldsymbol{x}_j \in F_P(\boldsymbol{p}_i)\}), \tag{1}$$

where $\boldsymbol{f}_i^{\text{pillar}}$ is the feature of pillar $\boldsymbol{p}_i$. We set the features for empty pillars to $\mathbf{0}$. This results in a dense 2D grid $\mathcal{F}^{\text{pillar}} \subset \mathbb{R}^{H^{\text{p}} \times W^{\text{p}} \times C^{\text{p}}}$, where $H^{\text{p}}, W^{\text{p}}$ and C^{p} are the height, width, number of channels of this 2D pillar map, respectively. Multiple stacked convolutional layers further embed the pillar features to the final feature map $\mathcal{F}^{\text{d}} \subset \mathbb{R}^{H^{\text{d}} \times W^{\text{d}} \times C^{\text{d}}}$.

An alternative BEV embedding is SparseConv [60]. If $F_V(i)$ returns the set of points in voxel i, SparseConv collects point-wise features into voxel-wise features by

$$\boldsymbol{f}_i^{\text{voxel}} = \text{PN}(\{\boldsymbol{f}_j | \boldsymbol{x}_j \in F_V(i)\}), \tag{2}$$

where $\boldsymbol{f}_i^{\text{voxel}}$ contains the features of voxel i. In contrast to PointPillars, SparseConv conducts 3D sparse convolutions to refine the voxel-wise features. Finally, we compress these sparse voxels to a BEV 2D grid by filling empty voxels with zeros and averaging along the z-axis. For ease of notation, we also denote the resulting 2D grid $\mathcal{F}^{\text{d}} \subset \mathbb{R}^{H^{\text{d}} \times W^{\text{d}} \times C^{\text{d}}}$.

4 Object DGCNN

After obtaining the BEV features $\mathcal{F}^{\text{d}}$ using one of the architectures above, we predict a set of bounding boxes as well as a label for each box. The key difference between our architecture and most recent 3D object detection methods is that ours produces a *set* of bounding boxes rather than a box per grid cell followed by NMS, as in [5, 6]. Hence, we need to transition from a grid of per-pillar features to an unordered set of objects; we detail our approach below. We address two key issues: prediction of the bounding boxes and evaluation of the loss.

Desiderata. Object DGCNN uses a DGCNN-inspired architecture but incorporates grid-based BEV features, built on the philosophy that local features (§3) are reasonable to store on a dense grid, but object predictions are better modeled using sets. Hence, we require a new architecture and set-to-set loss that encourage bounding box diversity.

Object DGCNN uses L layers that follow a series of set-based computations to produce bounding box predictions from the BEV feature maps. Each layer employs the following steps (Figure 1):

1. predict a set of query points and attention weights;
2. collect BEV features from keypoints determined by the queries; and
3. model object-object interactions via DGCNN.

Each layer results in a more refined set of bounding box predictions, one per query. At the end of these layers, we match the prediction set with the ground-truth set in a one-to-one fashion and evaluate a set-to-set object detection loss.

Single layer. Inspired by DETR [7], each layer $\ell \in \{0, \dots, L-1\}$ of Object DGCNN operates on a set of *object queries* $\mathcal{Q}_\ell = \{\boldsymbol{q}_{\ell 1}, \dots, \boldsymbol{q}_{\ell M^*}\} \subset \mathbb{R}^Q$, producing a new set $\mathcal{Q}_{\ell+1}$. Although queries are fully learnable, our intuition is that they represent progressively refined object positions.

The initial set of object queries $\mathcal{Q}_0$ is learned jointly with the neural network weights, yielding a dataset-specific prior. Beyond this fixed initial set, below we detail how to incorporate scene information to obtain $\mathcal{Q}_{\ell+1}$ from $\mathcal{Q}_\ell$ using an approach inspired by DGCNN [8] and deformable self-attention [17]. For notational convenience, we drop the ℓ subscript.

Starting from each query $\boldsymbol{q}_i$ (or, without the index dropped, $\boldsymbol{q}_{\ell i}$), we decode a reference point $\boldsymbol{p}_i \in \mathbb{R}^2$, a set of offsets $\{\boldsymbol{\delta}_{i0}, \ldots, \boldsymbol{\delta}_{iK}\} \subset \mathbb{R}^2$, and a set of attention weights $\{w_{i0}, \ldots, w_{iK}\} \subset \mathbb{R}$:

$$\begin{aligned} &\boldsymbol{p}_i = \Phi_{\text{ref}}(\boldsymbol{q}_i), \qquad \{\boldsymbol{\delta}_i^0, \ldots, \boldsymbol{\delta}_i^k, \ldots \boldsymbol{\delta}_i^K\} = \Phi_{\text{neighbor}}(\boldsymbol{q}_i), \\ &\{w_i^0, \ldots, w_i^k, \ldots w_i^K\} = \Phi_{\text{atten}}(\boldsymbol{q}_i), \end{aligned} \tag{3}$$

where Φ_{ref}, Φ_{neighbor}, and Φ_{atten} are shared neural networks among the queries. We think of $\boldsymbol{p}_i$ as a hypothesis for the center of the i-th object; the $\boldsymbol{\delta}$'s represent the positions of K informative points relative to the position of the object that determine its geometry.

Next, we collect a BEV feature $\boldsymbol{f}_{ik}$ associated to each neighbor point $\boldsymbol{p}_{ik} = \boldsymbol{p}_i + \boldsymbol{\delta}_{ik}$:

$$\boldsymbol{f}_{ik} = f_{\text{bilinear}}(\mathcal{F}^d, \boldsymbol{p}_i + \boldsymbol{\delta}_{ik}), \tag{4}$$

where f_{bilinear} bilinearly interpolates the BEV feature map $\mathcal{F}^d$. Note this step is the interaction between our set-based architecture manipulating query points $\boldsymbol{q}_i$ and the grid-based feature map $\mathcal{F}^d$. We then aggregate a single object query feature $\boldsymbol{f}_i^o$ from the $\boldsymbol{f}_{ik}$s:

$$\boldsymbol{f}_i^o = \sum_k \frac{e^{w_{ik}}}{\sum_k e^{w_{ik}}} \boldsymbol{f}_{ik}. \tag{5}$$

This generates scene-aware features; each object query "attends" to a certain area in the scene.

In the current layer ℓ, the queries have not yet interacted with each other. To incorporate neighborhood information in object detection estimates, we use DGCNN-style operations to model a sparse set of relations. We construct a graph between the queries using a nearest neighbor search in feature space. In particular, we connect each query feature $\boldsymbol{f}_i^o$ to its 16 nearest neighbors as ablated in Table 7. Identically to DGCNN, we learn a feature per edge e_{ij} and then aggregate back to the vertices i to produce the new set of object queries. In detail, we write:

$$\boldsymbol{q}_{(\ell+1)i} = \max_{\text{edges } e_{ij}} \Phi_{\text{edge}}(\boldsymbol{f}_i^o, \boldsymbol{f}_j^o), \tag{6}$$

where $\max$ denotes a channel-wise maximum and Φ_{edge} is a neural network for computing edge features. This completes our layer for computing $\mathcal{Q}_{\ell+1}$ from $\mathcal{Q}_\ell$. Optionally, we repeat this last step multiple times, in effect applying DGCNN to the features $\boldsymbol{f}_i^o$ to get the point set $\mathcal{Q}_{\ell+1}$.

Set-to-set loss. After L Object DGCNN layers as described above, we are left with a set of M^* queries $\mathcal{Q}_L$ used to predict our bounding boxes. For each query $\boldsymbol{q}_{Li}$, we use a classification network to predict a categorical label $\hat{c}_i$ and a regression network to predict bounding box parameters $\hat{\boldsymbol{b}_i}$. Our final task is to assign the predictions to the ground-truth boxes and compute a set-to-set loss.

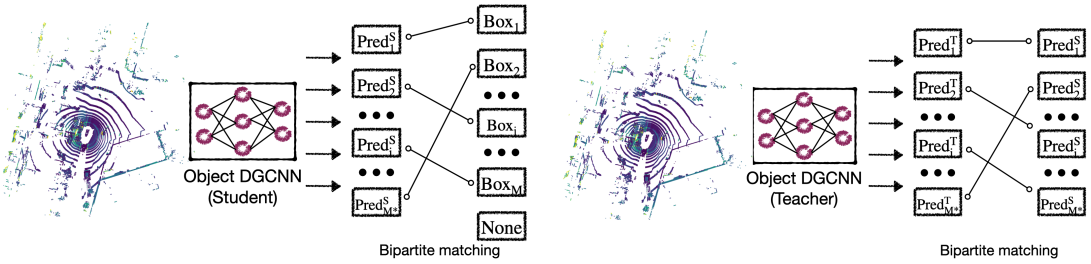
Most object detection models minimize a loss $\mathcal{L}_{\text{od}}$ given by

$$\mathcal{L}_{\text{od}} = \sum_{j=1}^{\hat{M}} -\log \hat{p}_{\hat{\sigma}(j)}(\hat{c}_j) + 1_{\{c_{\hat{\sigma}(j)} \neq \varnothing\}} \mathcal{L}_{\text{box}}(\hat{\boldsymbol{b}}_j, \boldsymbol{b}_{\hat{\sigma}(j)}), \tag{7}$$

where $\hat{M} = H^{\text{d}} * W^{\text{d}}$, $\hat{\sigma}(*)$ returns the corresponding index of the ground-truth bounding box, $\hat{p}_{\hat{\sigma}(j)}(c_j)$ is the probability of class $c_{\hat{\sigma}(j)}$ for the prediction with index $\sigma(j)$, $\varnothing$ denotes an invalid box, and $\mathcal{L}_{\text{box}}$ is typically the $\mathcal{L}_1$ distance. Different matchings $\hat{\sigma}$ yield different optimization landscapes and hence different prediction models. Pillar-OD [5] and CenterPoint [6] employ a simple $\hat{\sigma}$ to determine the ground-truth box used to evaluate the box predicted at BEV pixel j:

$$\hat{\sigma}_{\text{overlap}}(j) = \begin{cases} j^{'}, & \text{if } \boldsymbol{b}_{j^{'}} \text{ overlaps with BEV pixel } j; \\ \varnothing, & \text{otherwise.} \end{cases} \tag{8}$$

This strategy can assign a box to multiple nearby BEV pixels. This one-to-many assignment provides dense supervision for the object detector and eases optimization. Since the training objective



(a) Student network (b) Teacher network (fixed)

Figure 2: The set-to-set distillation pipeline. The student network is trained with the ground-truth supervision as well as with the supervision from a fixed teacher network.

encourages each BEV pixel to predict the same surrounding box, however, redundant boxes are inevitable. So, NMS is usually required to remove redundant boxes at inference time.

Rather than performing dense predictions in the BEV, we make per-query predictions. Typically, M^* is much larger than the number of ground-truth boxes M. To account for this difference, we pad the set of ground-truth boxes with $\varnothing$s (no object) up to M^*. Following [7], we use an objective built on an optimal matching between these two sets. We define the optimal bipartite matching as

$$\sigma^* = \arg\min_{\sigma \in \mathcal{P}} \sum_{j=1}^{M} -1_{\{c_j=\varnothing\}} \hat{p}_{\sigma(j)}(c_j) + 1_{\{c_j=\varnothing\}} \mathcal{L}_{\text{box}}(\boldsymbol{b}_j, \hat{\boldsymbol{b}}_{\sigma(j)}), \tag{9}$$

where $\mathcal{P}$ denotes the set of permutations, $\hat{p}_{\sigma(j)}(c_j)$ is the probability of class c_j for the prediction with index $\sigma(j)$, and $\mathcal{L}_{\text{box}}$ is the $\mathcal{L}_1$ loss for bounding box parameters. We use the Hungarian algorithm [61] to solve this assignment problem, as in [62, 7]. Our final set-to-set loss adapts (7):

$$\mathcal{L}_{\text{sup}} = \sum_{j=1}^{N} -\log \hat{p}_{\sigma^*(j)}(c_j) + 1_{\{c_j=\varnothing\}} \mathcal{L}_{\text{box}}(\boldsymbol{b}_j, \hat{\boldsymbol{b}}_{\sigma^*(j)}). \tag{10}$$

5 Distillation

Object DGCNN enables a new set-to-set knowledge distillation (KD) pipeline. KD usually involves a teacher model $\mathcal{T}$ and a student model $\mathcal{S}$. The common practice is to align the outputs of the student with those of the teacher using $\mathcal{L}_2$ distance or KL-divergence. In past 3D object detection methods, since final performance heavily relies on NMS and the predictions are post-processed to be a smaller set, distilling the teacher to the student is neither efficient nor effective. Since our set-based detection model is NMS-free, we can easily distill the information between models with homogeneous detection heads (per-query object detection head in our case). First, we train a teacher $\mathcal{T}$ using the method above with the loss in (10). Then, we train a student $\mathcal{S}$ with supervision given by $\mathcal{T}$ and the ground-truth. The class label and box parameters predicted by the teacher for each object query are $c_j^{\mathcal{T}}$ and $\boldsymbol{b}_j^{\mathcal{T}}$, respectively. The corresponding student outputs are $c_j^{\mathcal{S}}$ and $\boldsymbol{b}_j^{\mathcal{S}}$. We find an optimal matching between the output set of the teacher and that of the student:

$$\sigma_d^* = \arg\min_{\sigma_d \in \mathcal{P}} \sum_{j}^{N} -\log p_{\sigma_d(j)}(c_j^{\mathcal{T}}) + \mathcal{L}_{\text{box}}(\boldsymbol{b}_j^{\mathcal{T}}, \boldsymbol{b}_{\sigma_d(j)}^{\mathcal{S}}). \tag{11}$$

Then, the optimal matching's KD loss is given by

$$\mathcal{L}_{\text{distill}} = \sum_{j}^{N} -\log \hat{p}_{\sigma_d^*(j)}(c_j^{\mathcal{T}}) + \mathcal{L}_{\text{box}}(\boldsymbol{b}_j^{\mathcal{T}}, \boldsymbol{b}_{\sigma_d^*(j)}^{\mathcal{S}}). \tag{12}$$

So the overall loss during KD is $\mathcal{L} = \alpha\mathcal{L}_{\text{sup}} + \beta\mathcal{L}_{\text{distill}}$, where α and β balance the supervised loss and distillation loss. In practice, we use $\alpha = \beta = 1$.

Table 1: Comparisons to recent works. Our method is robust to whether to use NMS. $*$: implementations with the same PointPillars backbone. ‡: implementations with the same SparseConv backbone.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓	NMS
PointPillars [4]	53.3	40.0	-	-	-	-	-	✓
SSN [63]	54.83	41.56	-	-	-	-	-	✓
FreeAnchor [64]	55.3	43.7	-	-	-	-	-	✓
RegNetX-400MF-SECFPN [65]	55.2	41.2	-	-	-	-	-	✓
Pillar-OD [5]	56.84	44.41	-	-	-	-	-	✓
CenterPoint (pillar) [6] $*$	59.56	47.48	**31.27**	**25.81**	**33.78**	32.25	20.20	✓
CenterPoint (pillar) [6] $*$	55.08	40.27	35.14	26.44	36.75	32.66	19.55	
CenterPoint (voxel) [6] ‡	**64.19**	**54.99**	29.83	**25.71**	32.56	**26.08**	**18.89**	✓
CenterPoint (voxel) [6] ‡	57.00	45.32	31.66	27.14	40.47	37.23	20.14	
Ours (pillar) $*$	**60.54**	**48.71**	32.54	26.15	33.95	**26.60**	**18.73**	✓
Ours (pillar) $*$	**60.54**	48.67	32.54	26.15	33.95	**26.60**	**18.73**	
Ours (voxel) ‡	64.18	54.87	29.75	25.77	**32.40**	26.11	18.91	✓
Ours (voxel) ‡	64.17	54.96	**29.74**	25.78	32.42	26.16	18.94	

Table 2: Comparisons of different distillation approaches.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
Baseline (without distillation)	41.32	37.78	44.99	27.65	64.43	144.26	39.30
Feature distill (voxel→pillar)	41.89	38.12	44.87	27.78	64.12	143.78	39.19
Pseudo labels (voxel→pillar)	39.39	36.56	45.25	28.12	65.23	144.56	40.13
Set-to-set distill (voxel→pillar)	42.21	39.01	43.97	27.73	64.00	143.75	39.20

6 Experiments

We present our experiments in four parts. We introduce the dataset, metrics, implementation, and optimization details in §6.1. Then, we demonstrate performance on the nuScenes dataset [66] in §6.2. We present knowledge distillation results in §6.3. Finally, we provide ablation studies in §6.4.

6.1 Training & testing procedures

Dataset. We experiment on the nuScenes dataset [66]. nuScenes provides rich annotations and diverse scenes. It has 1K short sequences captured in Boston and Singapore with 700, 150, 150 sequences for training, validation, and testing, respectively. Each sequence is $\sim$20s and contains 400 frames. This dataset provides annotation every 0.5s, leading to 28K, 6K, 6K annotated frames for training, validation, and testing. nuScenes uses 32-beam LiDAR, producing 30K points per frame. Following common practice, we use calibrated vehicle pose information to aggregate every 9 non-key frames to key frames, so each annotated frame has $\sim$300K points. The annotations include 23 classes with a long-tail distribution, of which 10 classes are included in the benchmark.

Metrics. The major metrics are mean average precision (mAP) and the nuScenes detection score (NDS). In addition, we use a set of true positive metrics (TP metrics), which include average translation error (ATE), average scale error (ASE), average orientation error (AOE), average velocity error (AVE), and average attribute error (AAE). These metrics are computed in the physical unit.

Model architecture. Our model consists of three parts: a point-based feature extractor, a DGCNN to encode object queries and to connect the point cloud features to object queries, and a detection head to output the categorical label and bounding box parameters. We experiment with PointPillars [4] and SparseConv [60] as feature extractors. The three blocks of the PointPillars backbone have $[3, 5, 5]$ convolutional layers, with dimensions $[64, 128, 256]$ and strides $[2, 2, 2]$; the input features are downsampled to 1/2, 1/4, 1/8 of the original feature map. For SparseConv, we use four blocks of $[3, 3, 3, 2]$ 3D sparse convolutional layers, with dimensions $[16, 32, 64, 128]$ and strides $[2, 2, 2, 1]$; the input features are downsampled to 1/2, 1/4, 1/8, 1/8 of the original feature map. For SparseConv, we transform the features into BEV by collapsing the z-axis. Both backbones use two deformable self-attention [17] layers with dimensions $[256, 256]$ to transform the BEV features. Then, we use two DGCNNs to encode the object queries. Each DGCNN [8] contains two EdgeConv layers with dimensions $[256, 256]$, both with 16 nearest neighbors. For each object query, we predict four points

Table 3: Comparisons of self-distillation versus baselines.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
Baseline (without distillation)	41.32	37.78	44.99	27.65	64.43	144.26	39.30
Self-distillation (pillar→pillar)	42.12	38.89	44.01	27.78	64.01	144.01	39.21
Baseline (without distillation)	58.46	48.01	31.33	25.99	33.89	32.29	20.37
Self-distillation (voxel→voxel)	59.55	49.84	31.17	25.77	33.73	32.22	20.22

Table 4: Self-distillation with privileged information.

Method	NDS ↑	mAP ↑	mATE ↓	mASE ↓	mAOE ↓	mAVE ↓	mAAE ↓
Sparse→sparse (pillar)	42.12	38.89	44.01	27.78	64.01	144.01	39.21
Dense→sparse (pillar)	42.79	39.10	43.89	27.77	64.01	143.97	39.11
Sparse→sparse (voxel)	59.55	49.84	31.17	25.77	33.73	32.22	20.22
Dense→sparse (voxel)	59.89	50.12	31.11	25.76	33.70	32.19	20.11

in the BEV to obtain and aggregate the BEV features. The final feature for this object query is the weighted sum of features of these four BEV points. The final detection head takes the features of each object query and predicts class label and bounding box parameters w.r.t. the reference point.

Training & inference. We use AdamW [67] to train the model. The weight decay for AdamW is 10^{-2}. Following a cyclic schedule [68], the learning rate is initially 10^{-4} and gradually increased to 10^{-3}, which is finally decreased to 10^{-8}. The model is initialized with a pre-trained PointPillars network on the same dataset. We train for 20 epochs on 8 RTX 3090 GPUs. During inference, we take the top 100 objects with highest classification scores as the final predictions.We *do not* use any post-processing such as NMS. For evaluation, we use the toolkit provided with the nuScenes dataset.

6.2 Object DGCNN

We compare to top-performing methods on the nuScenes dataset in Table 1. PointPillars [4] is an anchor-based method with reasonable trade-off between performance and efficiency. FreeAnchor [64] extends PointPillars by learning how to assign anchors to the ground-truth. RegNetX-400MF-SECFPN [65] uses neural architecture search (NAS) to learn a flexible neural network for 3D detection; it is essentially a variant of PointPillars with an enhanced backbone network. Different from anchor-based methods, Pillar-OD [5] makes predictions per pillar, alleviating the class imbalance issue caused by anchors. CenterPoint [6] exploits similar detection heads, with better performance using better training scheduling and data augmentation. For these methods, we use re-implementations in MMDetection3D [69], which match the performances in the original papers.

We mainly compare to CenterPoint with both PointPillars and SparseConv backbones, denoted as "voxel" and "pillar" respectively. Our method outperforms other methods significantly and achieves comparable results compared to CenterPoint with NMS. Without NMS, the performance of CenterPoint drops considerably while our method is unaffected by NMS. This finding verifies the DGCNN implicitly models object relations and removes redundant boxes.

6.3 Set-to-set distillation

In this section, we present experiments involving our set-to-set distillation pipeline. To limit computation time, we train each model over a shorter period of time (1 epoch versus 20 epochs in the previous section). We conduct three types of distillation. First, we distill a teacher model with a SparseConv backbone to a student model with a PointPillars backbone (denoted as "voxel→pillar"). This aligns with the common knowledge distillation setup for classification. We compare to feature-based distillation and pseudo label based methods. The objective of feature-based distillation is to align the middle-level features of the teacher model and the student model while the pseudo label based methods generate pseudo training examples with the pre-trained teacher networks. As Table 2 shows, our set-to-set distillation achieves better performance, confirming that distilling the last stage of the object detection model is more effective than distilling feature maps.

Second, we perform self-distillation [49] (denoted as "voxel→voxel" and "pillar→pillar"), where the teacher and the student are identical and take the same point clouds as input. As Table 3 shows, even when the teacher network and the student network have the same capacity, self-distillation still introduces a performance boost. This finding is consistent with the results in [49].

Finally, we try distillation with privileged information [56], where the teacher gets access to privileged information but the student does not. Following [10], the teacher takes dense point clouds, and the student takes sparse point clouds (denoted as "dense→sparse"). The goal is for the student model to learn the same representations as the teacher model without knowing the dense inputs. In Table 4, we compare this setup with self-distillation, where the difference is the teacher model and the student model take the same sparse point clouds in self-distillation. The student achieves better performance when the teacher takes dense point clouds. The result suggests that set-to-set knowledge distillation is an effective approach to transfer insight from privileged information.

6.4 Ablation

We provide ablation studies on different components of our model to verify assorted design choices. First, we study the improvements of DGCNN over its counterpart, multi-head self-attention [9]. The multi-head self-attention has 8 heads with embedding dimension 256 and LayerNorm [70], following common usage. The DGCNN has two EdgeConv layers with dimensions $[256, 256]$. The number of neighbors K in EdgeConv is 16. In principle, DGCNN is a sparse version of multi-head self-attention; the sparse structure reduces overhead in back-propagation and leads to sharper "attention maps" as well as faster convergence.

Table 5: DGCNN versus multi-head self-attention.

Metric \ Method	Multi-head self-attention	DGCNN
NDS	39.89	**41.32**
mAP	36.35	**37.81**

Table 6: Models with different # DGCNNs.

Metric \ # layers	1	2	3	4	5	6
NDS	35.91	39.75	41.15	41.26	41.07	**41.32**
mAP	32.32	36.54	37.25	37.75	37.78	**37.81**

Table 7: The number of neighbors in DGCNN.

Metric \ # neigbhors	1	4	8	16	32	64
NDS	40.21	40.45	40.51	**41.32**	40.17	39.80
mAP	36.81	37.15	37.46	**37.81**	37.12	36.70

Table 5 shows the comparisons: DGCNN consistently outperforms multi-head self-attention. This aligns with our hypothesis: objects are distributed sparsely in the scene, so dense interactions among objects are neither efficient nor effective. Furthermore, we study the effect of number of neighbors in DGCNN. When it is 1, The model reduces to an architecture without object interaction. As we increase the number, it approaches multi-head self-attention. As shown in Table 7, the sweet spot is 16, which appears to balance object interactions and sparsity.

We also investigate improvements introduced when more DGCNNs are stacked in Table 6. This result suggests it is beneficial to incorporate multiple DGCNNs to model the dynamic object relations.

7 Conclusion

Object DGCNN is a highly-efficient 3D object detector for point clouds. It is able to learn object interactions via dynamic graphs and is optimized through a set-to-set loss, leading to NMS-free detection. The success of Object DGCNN indicates that many post-processing operations in 3D object detection are likely unnecessary and can be replaced with suitable neural network modules. Moreover, we introduce a set-to-set knowledge distillation pipeline enabled by the Object DGCNN. This new pipeline significantly simplifies knowledge distillation for 3D object detection and may be applicable to other tasks like 3D model compression. Beyond the direct usage of our model, our experiments suggest several future directions to address current limitations. For example, our method is initialized with a pre-trained backbone network. Training the model from scratch remains elusive due to the sparse set-to-set supervision; solving this issue may yield improved generalization as in [71]. Furthermore, studying 3D-specific feature extractors will improve the speed and generalizability of 3D object detection. Finally, the large amount of unlabeled data available at training time can serve as another type of privileged information to apply self-supervised learning to 3D domains through set-to-set distillation.

Potential impact. Our method aims to improve the object detection pipeline, which is crucial for the safety of autonomous driving systems. One potential negative impact of our work is that it still lacks theoretical guarantees, similar to many deep learning methods. Future work to improve applicability in this domain might consider challenges of *explainability* and *transparency*.

References

[1] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[2] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NeurIPS)*, 2015.

[4] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[5] Yue Wang, Alireza Fathi, Abhijit Kundu, David Ross, Caroline Pantofaru, Thomas Funkhouser, and Justin Solomon. Pillar-based object detection for autonomous driving. In *The European Conference on Computer Vision*, 2020.

[6] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. *CVPR*, 2021.

[7] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[8] Yue Wang, Yongbin Sun, Sanjay E. Sarma Ziwei Liu, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38:146, 2019.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[10] Yue Wang, Alireza Fathi, Jiajun Wu, Thomas Funkhouser, and Justin Solomon. Multi-frame to single-frame: Knowledge distillation for 3d object detection. In *The Workshop on Perception for Autonomous Driving at the European Conference on Computer Vision*, 2020.

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *The European Conference on Computer Vision (ECCV)*, 2016.

[12] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *The International Conference on Computer Vision (ICCV)*, 2017.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *The International Conference on Computer Vision (ICCV)*, 2017.

[15] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019.

[16] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *European Conference on Computer Vision (ECCV)*, September 2018.

[17] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable {detr}: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2021.

[18] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[19] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[20] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[21] 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 *The Conference on Robot Learning (CoRL)*, 2019.

[22] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Groß. Complex-yolo: Real-time 3d object detection on point clouds. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[23] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[24] 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 *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[25] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation. In *The International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[26] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[27] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[28] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[29] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *The European Conference on Computer Vision (ECCV)*, 2018.

[30] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *The International Conference on Computer Vision*, 2019.

[31] Charles R Qi, Xinlei Chen, Or Litany, and Leonidas J Guibas. Imvotenet: Boosting 3d object detection in point clouds with image votes. *arXiv preprint arXiv:2001.10692*, 2020.

[32] Paul V.C. Hough. Machine analysis of bubble chamber pictures. *The International Conference in High Energy Accelerators and Instrumentation*, 1959.

[33] Mahyar Najibi, Guangda Lai, Abhijit Kundu, Zhichao Lu, Vivek Rathod, Tom Funkhouser, Caroline Pantofaru, David Ross, Larry S. Davis, and Alireza Fathi. Dops: Learning to detect 3d objects and predict their 3d shapes. *ArXiv*, abs/2004.01170, 2020.

[34] Bin Yang, Ming Liang, and Raquel Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In *The Conference on Robot Learning (CORL)*, 2018.

[35] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[36] Kelvin Wong, Shenlong Wang, Mengye Ren, Ming Liang, and Raquel Urtasun. Identifying unknown instances for autonomous driving. In *The Conference on Robot Learning (CORL)*, 2019.

[37] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *SIGKDD*, 2006.

[38] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NeurIPS Deep Learning and Representation Learning Workshop*, 2015.

[39] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *The International Conference on Learning Representations (ICLR)*, 2015.

[40] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *The International Conference on Learning Representations (ICLR)*, 2017.

[41] Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation. In *The International Conference on Computer Vision (ICCV)*, pages 1365–1374, 2019.

[42] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3962–3971, 2019.

[43] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[44] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive representation distillation. In *The International Conference on Learning Representations (ICLR)*, 2020.

[45] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *ArXiv*, abs/1807.03748, 2018.

[46] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

[47] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[48] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

[49] Tommaso Furlanello, Zachary Chase Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born-again neural networks. In *ICML*, 2018.

[50] Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc V. Le. Bam! born-again multi-task networks for natural language understanding. In *ACL*, 2019.

[51] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *Neural Information Processing Systems (NeurIPS)*, 2017.

[52] Quanquan Li, Shengying Jin, and Junjie Yan. Mimicking very efficient network for object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[53] Tao Wang, Li Yuan, Xiaopeng Zhang, and Jiashi Feng. Distilling object detectors with fine-grained feature imitation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[54] Xialei Liu, Hao Yang, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Continual universal object detection. *ArXiv*, abs/2002.05347, 2020.

[55] Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 2009.

[56] David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. In *International Conference on Learning Representations (ICLR)*, 2016.

[57] Jong-Chyi Su and Subhransu Maji. Adapting models to signal degradation using distillation. In *British Machine Vision Conference (BMVC)*, 2017.

[58] John Lambert, Ozan Sener, and Silvio Savarese. Deep learning under privileged information using heteroscedastic dropout. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[59] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[60] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. 2018.

[61] H. W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[62] Russell Stewart, Mykhaylo Andriluka, and Andrew Y. Ng. End-to-end people detection in crowded scenes. In *2016 IEEE Conference on Computer Vision and Pattern Recognition CVPR*, pages 2325–2333. IEEE Computer Society, 2016.

[63] Xinge Zhu, Yuexin Ma, Tai Wang, Yan Xu, Jianping Shi, and Dahua Lin. Ssn: Shape signature networks for multi-class object detection from point clouds. In *Proceedings of the European Conference on Computer Vision*, 2020.

[64] Xiaosong Zhang, Fang Wan, Chang Liu, Rongrong Ji, and Qixiang Ye. FreeAnchor: Learning to match anchors for visual object detection. In *Neural Information Processing Systems*, 2019.

[65] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. 2020.

[66] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[67] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[68] Leslie N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[69] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. `https://github.com/open-mmlab/mmdetection3d`, 2020.

[70] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, 2016.

[71] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *arXiv preprint arXiv:1811.08883*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]