Finding Your (3D) Center:
3D Object Detection Using a Learned Loss

David Griffiths*[0000−0002−8582−138X], Jan Boehm[0000−0003−2190−0449], and
Tobias Ritschel

University College London, London, UK
{david.griffiths.16, j.boehm, t.ritschel}@ucl.ac.uk

Abstract. Massive semantically labeled datasets are readily available for
2D images, however, are much harder to achieve for 3D scenes. Objects in
3D repositories like ShapeNet are labeled, but regrettably only in isolation,
so without context. 3D scenes can be acquired by range scanners on city-
level scale, but much fewer with semantic labels. Addressing this disparity,
we introduce a new optimization procedure, which allows training for 3D
detection with raw 3D scans while using as little as 5 % of the object
labels and still achieve comparable performance. Our optimization uses
two networks. A *scene network* maps an entire 3D scene to a set of 3D
object centers. As we assume the scene not to be labeled by centers, no
classic loss, such as Chamfer can be used to train it. Instead, we use
another network to emulate the loss. This *loss network* is trained on a
small labeled subset and maps a non-centered 3D object in the presence
of distractions to its own center. This function is very similar – and
hence can be used instead of – the gradient the supervised loss would
provide. Our evaluation documents competitive fidelity at a much lower
level of supervision, respectively higher quality at comparable supervision.
Supplementary material can be found at: **dgriffiths3.github.io**

Keywords: 3D learning; 3D point clouds; 3D object detection; Unsu-
pervised

1 Introduction

We can reason about one 3D chair as we do about a 2D chair image, however,
we cannot yet machine-understand a point cloud of a 3D room as we would do
for a 2D room image. For 2D images, massive amounts of manual human labeling
have enabled amazing state-of-the-art object detectors [10, 19, 26, 34]. We also
have massive repositories of clean 3D objects [3] which we can classify thanks to
deep 3D point processing [25]. But we do not have, despite commendable efforts
[6, 28], and probably might never have, 3D scene labeling at the extent of 2D
images. We hence argue that progress in 3D understanding even more critically
depends on reducing the amount of supervision required.

---

* Corresponding author.

While general unsupervised detection is an elusive goal, we suggest taking a shortcut: while we do not have labeled 3D scenes, we do have labeled 3D objects. The key idea in this work is to first teach a *loss network* everything that can be learned from seeing snippets of labeled objects. Next, we use this network to learn a *scene network* that explores the relation of objects within scenes, but without any scene labels, i. e., on raw scans.

After reviewing previous work, we will show how this cascade of networks is possible when choosing a slightly more primitive loss than the popular Chamfer loss and we propose two network architectures to implement it. Results show how a state-of-the-art, simple, fast and feed-forward 3D detection network can achieve similar Chamfer distance and mAP@.25 scores to a supervised approach, but with only 5 % of the labels.

2   Previous Work

2D object detection has been addressed by deep-learning based approaches like Fast R-CNN [10], YOLO [26], SSD [19] or the stacked hourglass architecture [21] with great success.

In early work Song and Xiao [29] have extended sliding window-detection to a 3D representation using voxels with templates of Hough features and SVM classifiers. This approach was later extended to deep templates [30]. Both approaches use fully-supervised training on object locations given by bounding boxes. We compare to such a sliding window approach using a point-based deep template. Hou et al. [14] complement a voxel-based approach with color 2D image information which more easily represents finer details.

Karpathy et al. [15] detect objects by over-segmenting the scene and classifying segments as objects based on geometric properties such as compactness, smoothness, etc. Similarly, Chen et al. [4] minimize other features to 3D-detect objects in street scans.

While modern software libraries make voxels simple to work with, they are limited in the spatial extent of scenes they can process, and the detail of the scene they can represent. Qi et al. [24] were first to suggest an approach to work on raw point clouds. 3D object detection in point clouds is investigated by Qi et al. [23] and Engelcke et al. [7] map the scene to votes, then those votes are clustered and each cluster becomes a proposal. The vectors pointing from a seed to a vote are similar to the loss network gradients proposed in our method, but for VoteNet, this is part of the architecture during training and testing while for us these vectors are only part of the training. Finally, VoteNet is trained fully supervised with object positions. The idea of merging 2D images and 3D processing is applicable to point clouds as well, as shown by Ku et al. [16] and Qi et al. [22].

Zhou and Tuzel [35] question the usefulness of points for 3D detection and have suggested to re-sample point clouds to voxels again. Also Chen et al. [5] show how combining point inputs, volume convolutions and point proposals can

lead to good results. For a survey on 3D detection, also discussing trade-offs of points and voxels, see the survey by Griffiths and Boehm [11].

Our architecture is inspired by Fast R-CNN [10, 27], which regresses object proposals in one branch, warps them into a canonical frame and classifies them in the other branch. Recently, Yang et al. [33] have shown how direct mapping of a point cloud to bounding boxes is feasible. Feng et al. [8] train a network with supervision that makes multiple proposals individually and later reasons about their relation. Also, Zhou et al. [34] first work on center points for object representation alone and later regress the 2D bounding box and all other object features from image content around the points. All these works tend to first extract proposals in a learned fashion and then reason about their properties or relations in a second, learned step. We follow this design for the scene network, but drive its learning in an entirely different, unsupervised, way. Finally, all of these works require only one feed-forward point cloud network pass, a strategy we will follow as well.

Unrelated to 2D or 3D detection, Adler and Öktem [1] have proposed to replace the gradient computation in an optimization problem by a neural network. In computer vision, this idea has been used to drive light field [9] or appearance synthesis [20]. We take this a step further and use a network to emulate the gradients in a very particular optimization: the training of another network.

3   Our Approach

We learn two networks: a *scene network* and a *loss network* (Fig. 1). The first (Fig. 1, bottom) is deployed, while the second (Fig. 1, top) is only used in training.
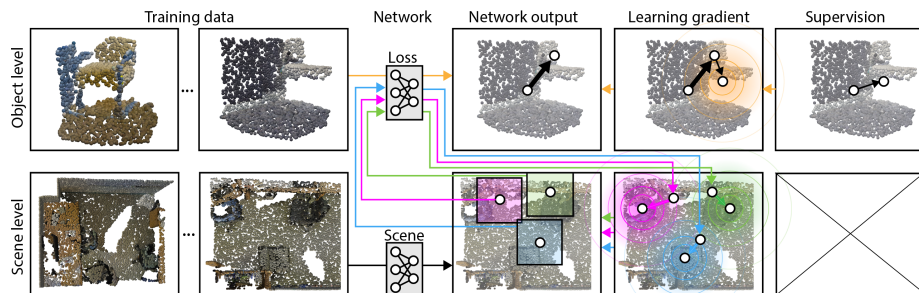

Fig. 1. Our approach proceeds in two steps of training **(row)** with different training data (column one and two), networks (column three), outputs (column four), gradients (column five) and supervision (column six). Object level training **(first row)** data comprises of 3D scene patches with known objects that are not centered. The loss network maps off-center scenes to their center (big black arrow). Its learning follows the gradient of a quadratic potential (orange field) that has the minimum at the offset that would center the object. This offset is the object-level supervision, as seen in the last column. The scene network **(second row)** is trained to map a scene to all object centers, here for three chairs. The gradient to train the scene network is computed by running the loss network from the previous step once for each object (here three times: blue, pink, green). Note, that there is no scene-level supervision (cross).

The *scene network* maps 3D scenes to sets of 3D object centers. The input data is a 3D point cloud. The output is a fixed sized list of 3D object centers. We assume a feed-forward approach, that does not consider any proposals [14, 21, 29, 30] or voting [22, 23], but directly regresses centers from input data [33, 34].

The *loss network* emulates the loss used to train the scene network. The input data is again a 3D point cloud, but this time of a single object, displaced by a random amount and subject to some other distortions. Output is not the scalar loss, but the gradient of a Mean Squared Error loss function.

In the following, we will first describe the training (Sec. 3.1) before looking into the details of both the scene and loss network implementation (Sec. 3.2).

3.1 Training

The key contribution of our approach is a new way of training. We will first look into a classic baseline with scene-level supervision, then introduce a hypothetical oracle that solves almost the same problem and finally show how this problem can be solved without scene-level supervision by our approach.
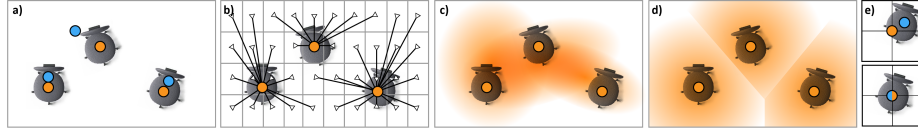

Fig. 2. a) A 2D scene with three chair objects, supervised by centers (orange) and their predictions (blue). **b)** The same scene, with the vector field of the oracle ∇ shown as arrows. **c)** A 2D Slice through a 6D cost function. **d)** A 2D Slice through an alternative cost function, truncated at the Voronoi cell edges. The oracle is the gradient of this. **e)** The simple task of the loss network: given a chair not in the center (top), regress an offset such that it becomes centered.

Supervised Consider learning the parameters θ of a scene network $\mathcal{S}_\theta$ which regresses object centers $\mathcal{S}_\theta(\mathbf{x}_i) = \hat{\mathbf{c}}$ from a scene $\mathbf{x}_i$. The scene is labeled by a set of 3D object centers $\mathbf{c}_i$ (Fig. 2, a). This is achieved by minimizing the expectation

$$\arg \min_\theta \, \mathbb{E}_i[H(\mathcal{S}_\theta(\mathbf{x}_i) - \mathbf{c}_i)], \tag{1}$$

using a two-sided Chamfer loss between the label point set $\mathbf{c}_i$ and a prediction $\hat{\mathbf{c}}_{\mathbf{i}}$

$$H(\hat{\mathbf{c}}, \mathbf{c}) = \mathbb{E}_i[\min_j ||\hat{\mathbf{c}}_i - \mathbf{c}_j||_2^2] + \mathbb{E}_i[\min_j ||\mathbf{c}_i - \hat{\mathbf{c}}_j||_2^2]. \tag{2}$$

Under H, the network is free to report centers in any order, and ensures all network predictions are close to a supervised center (precision) and all supervised centers are close to at least one network prediction (recall) (Fig. 3).

In this work, we assume the box center supervision $\mathbf{c}_i$ to not be accessible. Tackling this, we will first introduce an oracle solving a similar problem.
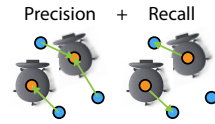

Precision + Recall

Fig. 3. Chamfer loss.

Oracle Consider, instead of supervision, an *oracle* function $\nabla(\mathbf{x})$ which returns for a 3D scene $\mathbf{p}$ *the smallest offset by which we need to move the scene so that the world center falls onto an object center* (Fig. 2, b). Then, learning means to

$$\arg\min_{\theta} \mathbb{E}_{i,j}[||\nabla(\underbrace{\mathbf{x}_i \ominus \mathcal{S}_\theta(\mathbf{x}_i)_j}_{\mathbf{y}_{\theta,i,j}})||_2^2], \tag{3}$$

where $\mathbf{x} \ominus \mathbf{d}$ denotes shifting a scene $\mathbf{x}$ by an offset $\mathbf{d}$. The relation between Eq. 1 and Eq. 3 is intuitive: knowing the centers is very similar to pointing to the nearest center from every location. It is, however, not quite the same. It assures every network prediction would map to a center, but does not assure, that there is a prediction for every center. We will need to deal with this concern later, by assuring space is well covered, so that there are enough predictions such that at least one maps to every center. We will denote a scene i shifted to be centered around object j by a *scene network* with parameters θ as $\mathbf{y}_{\theta,i,j}$.

Every location that maps to itself, i. e., a *fixed point* [31] of ∇, is an object center. Hence, we try to get a scene network that returns the roots of the gradient field of the distance function around each object center (Fig. 2, c):

$$\arg\min_{\theta} \mathbb{E}_{i,j}[||\nabla(\mathbf{y}_{\theta,i,j})||_2^2]. \tag{4}$$

Learned loss The key idea is to emulate this oracle with a *loss* network $\mathcal{L}_\phi$ having parameters ϕ as in

$$\arg\min_{\theta} \mathbb{E}_{i,j}[||\mathcal{L}_\phi(\mathbf{y}_{\theta,i,j})||_2^2]. \tag{5}$$

The loss network does not need to understand any global scene structure, it only locally needs to *center* the scene around the nearest object (Fig. 2, d). This task can be learned by working on local 3D object *patches*, without scene-level supervision. So we can train the loss network on any set of objects $\mathbf{o}_k$, translated by a known offset $\mathbf{d}_k$ using

$$\arg\min_{\phi} \mathbb{E}_k[||\mathbf{d}_k - \mathcal{L}_\phi(\mathbf{o}_k \ominus \mathbf{d}_k)||_2]. \tag{6}$$

As the loss network is local, it is also only ever trained on 3D patches. These can be produced in several different ways: sampling of CAD models, CAD models with simulated noise, by pasting simulated results on random scene pieces, etc. In our experiments, we use a small labeled scene subset to extract objects as follows: we pick a random object center and a 3D box of 1 meter size such that at least point representing an object surface is present in the box. Hence the center of the object is offset by a random, but known $\mathbf{d}_k$ we regress and subject to natural clutter. Note, that the box does not, and does not have to, strictly cover the entire object – which are of different sizes – but has to be just large enough to guess the center. Alg. 1 demonstrates how the *loss network* output can be used to provide *scene network* supervision.

Algorithm 1: $\mathcal{L}$: loss network, $\mathcal{S}$: scene network, k: proposal count, n 3D patch point count, m scene point count.

$\mathcal{L}_\phi : \mathbb{R}^{n\times 3} \rightarrow \mathbb{R}^3$;
$\mathcal{S}_\theta : \mathbb{R}^{m\times 3} \rightarrow \mathbb{R}^{k\times 3}$;
$\mathtt{crop} : \mathbb{R}^{m\times 3} \rightarrow \mathbb{R}^{n\times 3}$;
while *loss training* **do**
    $x = \mathtt{sampleScene}()$;
    $o = \mathtt{randObjectCenter}()$;
    $d = \mathtt{randOffset}()$;
    $p = \mathtt{crop}(x \ominus (o + d))$;
    $\nabla = \frac{\partial}{\partial_\phi} ||\mathcal{L}_\phi(p) - d||_2^2$;
    $\phi = \mathtt{optimizer}(\phi, \nabla)$;
end
 while *scene training* **do**
    $x = \mathtt{sampleScene}()$;
    $c = \mathcal{S}_\theta(x)$;
    for $i = 1 \ldots k$ **do**
        $p = \mathtt{crop}(x \ominus c_i)$;
        $\nabla_i = \mathcal{L}_\phi(p)$;
    end
    $\theta = \mathtt{optimizer}(\theta, \nabla)$;
end

Varying object count The above was assuming the number of objects n_c to be known. It did so when assuming a vector of a known dimension as supervision in Eq. 1 and did so, when assuming the oracle Eq. 3 and its derivations were returning gradient vectors of a fixed size. In our setting this number is unknown. We address this by bounding the number of objects and handling occupancy i.e., a weight indicting if an object is present or not, at two levels.

First, we train an occupancy branch $\mathcal{O}_\phi$ of the loss network that classifies occupancy of a single patch, much like the loss network regresses the center. We define space to be *occupied*, if the 3D patch contains any points belonging to the given objects surface. This branch is trained on the same patches as the loss network plus an equal number of additional 3D patches that do not contain any objects i.e. occupancy is zero.

Second, the occupancy branch is used to support the training of the scene network which has to deal with the fact that the number of actual centers is lower than the maximal number of centers. This is achieved by ignoring the gradients to the scene networks parameters θ if the occupancy network reports the 3D patch about a center to not contain an object of interest. So instead of Eq. 5, we learn

$$\arg\min_\theta \mathbb{E}_{i,j}[\mathcal{O}_\phi(\mathbf{y}_{\theta,i,j})\mathcal{L}_\phi(\mathbf{y}_{\theta,i,j})]. \tag{7}$$

The product in the sum is zero for centers of 3D patches that the loss network thinks, are not occupied and hence should not affect the learning.

Overlap When neither object centers nor their count are known, there is nothing to prevent two network outputs to map to the same center. While such duplicates can to some level be addressed by non-maximum suppression as a (non-differentiable) post-process to testing, we have found it essential to already prevent them (differentiable) from occurring when training the scene network. Without doing so, our training degenerates to a single proposal.

To this end, we avoid *overlap*. Let $v(q_1, q_2)$ be a function that is zero if the bounding boxes of the object in the scene centers do not overlap, one if they are

identical and otherwise be the ratio of intersection. We then optimize

$$\arg\min_{\theta} c_1(\theta) = \mathbb{E}_{i,j,k}\left[\mathcal{O}_\phi(\mathbf{y}_{\theta,i,j})\mathcal{L}_\phi(\mathbf{y}_{\theta,i,j}) + v(\mathbf{y}_{\theta,i,j}, \mathbf{y}_{\theta,i,k})\right]. \qquad (8)$$

We found that in case of a collision instead of mutually repelling all colliding objects, it can be more effective if out of multiple colliding objects, the collision acts on all but one winner object (winner-takes-all). To decide the winner, we again use the gradient magnitude: if multiple objects collide, the one that is already closest to the target i.e., the one with the smallest gradient, remains unaffected ($v = 0$) and takes possession of the target, while all others adapt.

Additional features For other object properties such as size, orientation, class of object, etc. we can proceed in two similar steps. First, we know the object-level property vector $\mathbf{q}$, so we can train a *property branch* denoted $\mathcal{P}_\theta$ that shares parameters θ with the loss network to regresses the property vector from the same displaced 3D patches as in Eq. 6

$$\arg\min_{\phi} \mathbb{E}_k[||\mathbf{q}_k - \mathcal{P}_\phi(\mathbf{o}_k \ominus \mathbf{d}_k)||_1]. \qquad (9)$$

For scene-level learning we extend the scene network by a branch $\mathcal{T}_\theta$ to emulate what the property network had said about the 3D patch at each center, but now with global context and on a scene-level

$$\arg\min_{\theta} c_1(\theta) + \alpha \cdot \mathbb{E}_{i,j}[|\mathcal{T}_\theta(\mathbf{y}_{\theta,i,j}) - \mathcal{P}\phi(\mathbf{y}_{\theta,i,j})|_1]. \qquad (10)$$

For simplicity, we will denote occupancy just as any other object property and assume it to be produced by $\mathcal{T}$ just, that it has a special meaning in training as defined in Eq. 7. We will next detail the architecture of all networks.
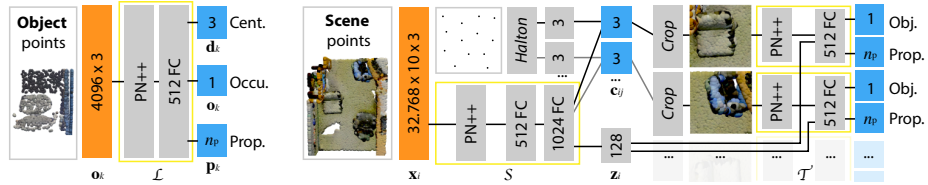

Fig. 4. The object **(left)** and scene **(right)** network. Input denoted orange, output blue, trainable parts yellow, hard-coded parts in italics. Please see Sec. 3.2 for a details.

3.2   Network

Both networks are implemented using PointNet++ [25] optimized using ADAM. We choose particularly simple designs and rather focus on the analysis of changes from the levels of supervision we enable.

Loss and occupancy network The loss network branches $\mathcal{L}$ and $\mathcal{O}$ share parameters ϕ and both map 4,096 3D points to a 3D displacement vector, occupancy and other scalar features (left in Fig. 4).

Scene network The scene network branches $\mathcal{S}$ and $\mathcal{T}$ jointly map a point cloud to a vector of 3D object centers and property vectors (including occupancy), sharing parameters θ. The box branch $\mathcal{S}$ first generates positions, next the scene is cropped around these positions and each 3D patch respectively fed into a small PointNet++ encoder $\mathcal{M}$ to produce crop specific local feature encodings. Finally, we concatenate the global scene latent code $\mathcal{S}_z$ with the respective local latent code $\mathcal{M}_z$ and pass it through the scene property branch $\mathcal{T}$ MLP.

The scene property branch is trained sharing all weights across all instances for all objects. This is intuitive, as deciding that e.g., a chair's orientation is the same for all chairs (the back rest is facing backwards), can at the same time be related to global scene properties (alignment towards a table).

Instead of learning the centers, we learn the residual relative to a uniform coverage of 3D space such that no object is missed during training. The Hammersley pattern [32] assures that, no part of 3D space is left uncovered.

We assume a fixed number of 32,768 input points for one scene. Note, that we do not use color as input, a trivial extension. Each MLP sub-layer is an MLP consisting of 3 fully-connected layers where layer 1 has 512 hidden states and the final layer contains the branch specific output nodes.

Post-process Our scene network returns a set of oriented bounding boxes with occupancy. To reduce this soft answer to a set of detected objects, e.g., to compute mAP metrics, we remove all bounding boxes with occupancy below a threshold τ_o, which we set to 0.9 in all our results.

In the evaluation, the same will be done for our ablations SLIDING and SUPERVISED, just that these also require additional non-maximum suppression (NMS) as they frequently propose boxes that overlap. To construct a final list of detections, we pick the proposal with maximal occupancy and remove any overlapping proposal with IoU $>$ .25 and repeat until no proposals remain.

4   Evaluation

We compare to different variants of our approach under different metrics and with different forms of supervision as well as to other methods.

4.1   Protocol

Data sets

We consider two large-scale sets of 3D scanned scenes: Stanford 2D-3D-S dataset (S3D) [2] and ScanNet [6]. From both we extract, for each scene, the list of object centers and object features for all objects of one class.

We split the dataset in three parts (Fig. 5): First, the test dataset is the official test dataset (pink in Fig. 5). The remaining training data is split into two parts: a labeled, and and unlabeled part. The labeled part (orange in Fig. 5) has all 3D scenes with complete annotations on them. The unlabeled part (blue in Fig. 5) contains only raw 3D point cloud without annotation. Note, that the labeled data is a subset of the unlabeled data, not a different set.

We call the ratio of labeled over unlabeled data the *label ratio*. To more strictly evaluate transfer across data sets, we consider ScanNet completely unlabeled. All single-class results are reported for the class `chair`.

Metrics Effectiveness is measured using the Chamfer distance (less is better) also used as a loss in Eq. 1 and the established mean Average Precision mAP@.25, (more is better) of a x % bounding box overlap test. X is chosen at 25 %.

Fig. 5. Label ratio.

Methods We consider the following *three* methods: SUPERVISED is the supervised approach define by Eq. 1. This method can be trained only on the labeled part of the training set. SLIDING window is an approach that applies our loss network, trained on the labeled data, to a dense regular grid of 3D location in every 3D scene to produce a heat map from which final results are generated by NMS. OURS is our method. The loss network is trained on the labeled data (orange in Fig. 5). The scene network is trained on the unlabeled data (blue in Fig. 5), which includes the labeled data (but without accessing the labels) as a subset.
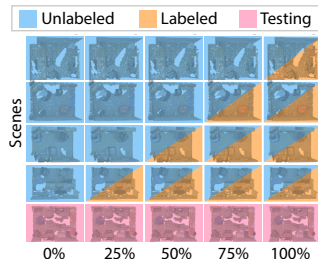
4.2   Results

Effect of supervision The main effect to study is the change of 3D detection quality in respect to the level of supervision. In Tbl. 1, different rows show different label ratios. The columns show Chamfer error and mAP@.25 for the class `chair` trained and tested on S3D.

Table 1. Chamfer error (less is better) and mAP@.25 (more is better) (**columns**), as a function of supervision (**rows**) in units of label ratio on the S3D class `chair`. Right, the supervision-quality-relation plotted as a graph for every method (color).

Ratio	Chamfer error			mAP		
	SUP	SLI	OUR	SUP	SLI	OUR
1 %	1.265	.850	**.554**	.159	**.473**	.366
5 %	.789	.577	**.346**	.352	.562	**.642**
25 %	.772	.579	**.274**	.568	.573	**.735**
50 %	.644	.538	**.232**	.577	.589	**.773**
75 %	.616	.437	**.203**	.656	.592	**.785**
100 %	.557	.434	**.178**	.756	.598	**.803**

We notice that across all levels of supervision, OUR approach performs better in Chamfer error and mAP than SLIDING window using the same object training or SUPERVISED training of the same network. It can further be seen, how all methods improve with more labels. Looking at a condition with only 5 % supervision, OUR method can perform similar to a SUPERVISED method that had 20× the labeling effort invested. At this condition, our detection is an acceptable .642,

which SUPERVISED will only beat when at least 75 % of the dataset is labeled. It could be conjectured, that the scene network does no more than emulating to slide a neural-network object detector across the scene. If this was true, SLIDING would be expected to perform similar or better than OURS, which is not the case. This indicates, that the scene network has indeed learned something not known at object level, something about the relation of the global scene without ever having labels on this level.

Fig. 6 plots the rank distribution (horizontal axis) of Chamfer distances (vertical axis) for different methods (colors) at different levels of supervision (lightness). We see that OUR method performs well across the board, SUPERVISED has a more steep distribution compared to SLIDING, indicating it produces good as well as bad results, while the former is more uniform. In terms of supervision scalability, additional labeling invested into our method

Fig. 6. Error distribution.

(brighter shades of yellow) result in more improvements to the right side of the curve, indicating, additional supervision is reducing high error-responses while already-good answers remain.
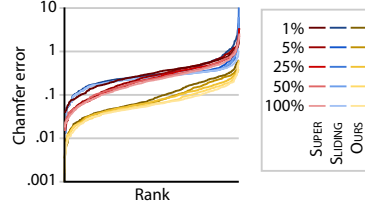
Transfer across data sets So far, we have only considered training and testing on S3D. In Tbl. 2, we look into how much supervision scaling would transfer to another data set, ScanNet. Remember, that we treat ScanNet as unlabeled, and hence, the loss network will be strictly only trained on objects from S3D. The three first rows in Tbl. 2 define the conditions compared here: a loss network always trained on S3D, a scene network trained on either S3D or ScanNet and testing all combinations on both data sets.

Table 2. Transfer across data sets: Different rows show different levels of supervision, different columns indicate different methods and metrics. The plot on the right visualizes all methods in all conditions quantified by two metrics. Training either on S3D or on ScanNet. The metrics again are Chamfer error (also the loss) and mAP@.25. Colors in the plot correspond to different training, dotted/solid to different test data.

Loss:	S3D				S3D					
Scene:	S3D				SCANNET					
Test:	S3D		ScanNet		S3D		ScanNet			
Ratio	Err.	mAP	Err.	mAP	Err.	mAP	Err.	mAP		
1%	0.554	.366	1.753	.112	0.579	.296	0.337	.548		
5%	0.346	.642	0.727	.138	0.466	.463	0.703	.599		
50%	0.232	.773	0.588	.380	0.447	.497	0.258	.645		
100%	0.178	.803	0.789	.384	0.336	.555	0.356	.661		

Column two and three in Tbl. 2 and the dotted violet line in the plot, iterate the scaling of available label data we already see in Tbl. 1 when training and testing on S3D. Columns four and five, show a method trained on S3D but tested

on ScanNet. We find performance to be reduced, probably, as the domain of ScanNet is different from the one of S3D. If we include the unlabeled scenes of ScanNet in the training, as seen in columns six to nine, the quality increases again, to competitive levels, using only S3D labels and 0 % of the labels available for ScanNet.

Table 3. Performance of the loss network for different label ratio (**rows**) on different test data and according to different metrics (**columns**). [1] *Class not present in ScanNet.*

Ratio	Class	#Sce	#Obj	S3D Err.	S3D Acc.	SCANNET Err.	SCANNET Acc.
1 %	chair	11	2400	.0085	.853	.0099	.843
5 %	chair	54	16,000	.0052	.936	.0075	.899
25 %	chair	271	47,200	.0049	.949	.0071	.907
50 %	chair	542	121,191	.0046	.953	.0069	.902
75 %	chair	813	162,000	.0045	.955	.0065	.920
100 %	chair	1084	223,980	.0043	.960	.0068	.911
5 %	table	54	5060	.0078	.921	—[1]	—[1]
5 %	bcase	54	4780	.0093	.819	—[1]	—[1]
5 %	column	54	2780	.0100	.855	—[1]	—[1]

Tbl. 3 further illustrate the loss network: how good are we at finding vectors that point to an object center? We see that the gradient error and the confidence error, both go down moderately with more labels when training and testing on S3D (violet). The fact that not much is decreasing in the loss network, while the scene network keeps improving, indicates the object task can be learned from little data, and less object-level supervision is required than what can be learned on a scene level, still. We further see, that the loss network generalizes between data sets from the fact that it is trained on S3D (violet curve) but when tested on ScanNet (green curve) goes down, too.

Besides seeing how quality scales with the amount of labeled supervision for training the loss network, it is also relevant to ask what happens when the amount of unlabeled training data for the scene network is increased while holding the labeled data fixed. This is analyzed in Tbl. 4. Here we took our loss network and trained it at 5 % label ratio on S3D and tested on ScanNet. Next, the scene network was trained, but on various number of scenes from Scan-Net, which, as we said, is considered unlabeled. The number of scenes changes over

Table 4. Chamfer error and mAP@.25 reported for varying the number of scenes.
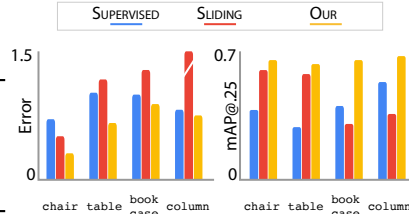
#Sce	OUR Err.	OUR mAP
66	.643	.079
330	.509	.242
1648	.506	.360
3295	.457	.412
4943	.435	.479
6590	.407	.599

columns, resp. along the horizontal axis in the plot. We see that without investing any additional labeling effort, the scene network keeps increasing substantially,

indicting what was learned on a few labeled S3D objects can enable understanding the structure of ScanNet.

Different classes Tbl. 1 was analyzing the main axis of contribution: different levels of supervision but for a single class. This has shown that at around a label ratio of 5 % OUR method performs similar to a SUPERVISED one. Holding the label ration of 5 % fixed and repeating the experiment for other classes, is summarized in Tbl. 5. We see, that the relation between SUPERVISED, SLIDING and OURS is retained across classes.

Table 5. Chamfer error (less is better) and mAP@.25 precision (more is better) **(columns)**, per class **(rows)** at a supervision of 5 % labeling ratio.

	Chamfer error			mAP		
Class	SUP	SLI	OUR	SUP	SLI	OUR
chair	0.789	0.577	.346	.352	.562	.642
table	1.144	1.304	.740	.282	.528	.615
bookcase	1.121	1.427	.979	.370	.298	.640
column	0.900	2.640	.838	.490	.353	.654

Comparison to other work In Tbl. 6 we compare our approach to other methods. Here, we use 20 % of ScanNet V2 for testing and the rest for training. Out of the training data, we train our approach once with 100 % labeled and once with only 5 % labeled. Other methods were trained at 100 % label ratio.

Table 6. Performance (mAP(%) with IoU threshold .25) of different methods **(rows)** on all classes **(columns)** of ScanNet V2. [1] 5 images. [2] Only xyz. [3] Their ablation; similar to our backbone.

Method		cabinet	bed	chair	sofa	table	door	window	bookshelf	picture	counter	desk	curtain	fridge	curtain	toilet	sink	bathtub	other	mAP
3DSIS[1]	[14]	19.8	69.7	66.2	71.8	36.1	30.6	10.9	27.3	0.0	10.0	46.9	14.1	53.8	36.0	87.6	43.0	84.3	16.2	40.2
3DSIS[2]	[14]	12.8	63.1	66.0	46.3	26.9	8.0	2.8	2.3	0.0	6.9	33.3	2.5	10.4	12.2	74.5	22.9	58.7	7.1	25.4
MTML	[17]	32.7	80.7	64.7	68.5	57.1	41.8	39.6	58.8	18.2	0.4	18.0	81.5	44.5	100.0	100.0	44.2	100.0	36.4	54.9
VoteNet	[23]	36.3	87.9	88.7	89.6	58.8	47.3	38.1	44.6	7.8	56.1	71.7	47.2	45.4	57.1	94.9	54.7	92.1	37.2	58.7
BoxNet[3]	[17]						No per-class information available													45.4
3D-BoNet	[33]	58.7	88.7	64.3	80.7	66.1	52.2	61.2	83.6	24.3	55.0	72.4	62.0	51.2	100.0	90.9	75.1	100.0	50.1	68.7
Ours 100 %		43.0	70.8	58.3	16.0	44.6	28.0	13.4	58.2	4.9	69.9	74.0	75.0	36.0	58.9	79.0	47.0	77.9	48.2	50.2
Ours 5 %		38.1	68.9	58.9	88.8	42.5	21.1	9.0	53.2	6.8	53.9	68.0	62.3	26.5	45.6	69.9	40.4	66.9	48.0	48.3

We see that our approach provides competitive performance, both at 100 % of the labels, as well as there is only a small drop when reducing supervision by factor 20×. Our mAP at 100 % of the labels is better than both variants (with and without color) of 3DSIS [14] from 2018 and similar to MTML [17] from 2019. VoteNet [23] and 3D-BoNet [33] are highly specialized architectures from 2019 that have a higher mAP. We have included BoxNet from Qi et al. [23], an

ablation they include as a vanilla 3D detection approach that is similar to what we work with. We achieve similar even slightly better performance, yet at 5 % of the supervision. In some categories, our approach wins over all approaches. We conclude that a simple backbone architecture we use is no contribution and cannot win over specialized ones, but that it also is competitive to the state-of-the-art. We should note here, as we do not carry out Semantic instance segmentation in our network, we did not test on the official test ScanNet benchmark test set. Instead, we reserve 20% of the labeled training scenes for testing.

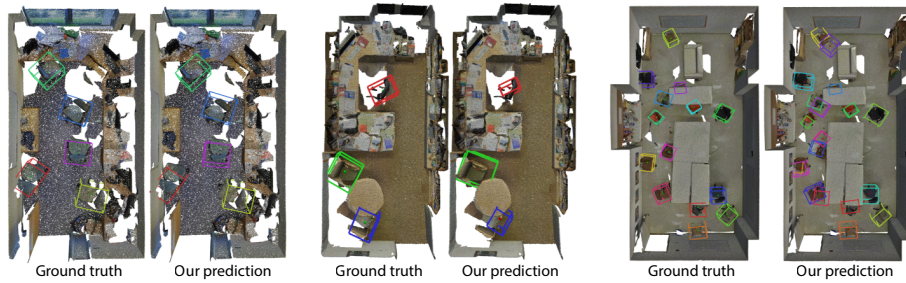**Qualitative results** Fig. 7 shows qualitative example results of our approach.

Ground truth   Our prediction   Ground truth   Our prediction   Ground truth   Our prediction

Fig. 7. Qualitative results of our approach and the ground truth for `chair` on S3D.

Computational efficiency Despite the additional complexity in training, at deployment, out network is a direct and fast forward architecture, mapping a point cloud to bounding boxes. Finding 20 proposals in 32,768 points takes 189 ms, while the supervised takes the same amount of time, with the small overhead of a NMS (190 ms) on a Nvidia RTX 2080Ti. Our CPU implementation of sliding window requires 4.7 s for the same task on a i7-6850K CPU @ 3.60GHz. All results are computed with those settings.

5   Discussion

How can OURS *be better than the* SUPERVISED*?* It is not obvious why at 100 % label ratio in Tbl. 1, the SUPERVISED architecture performs at an mAP of .756 while OURS remains slightly higher at an mAP of .803. This is not just variance of the mAP estimation (computed across many identical objects and scenes).

A possible explanation for this difference is, that our training is no drop-in replacement for supervised training. Instead, it optimizes a different loss (truncation to the nearest object and collision avoidance) that might turn out to be better suited for 3D detection than what it was emulating in the beginning. We, for example, do not require NMS. As our training does not converge without those changes to the architecture, some of the effects observed might be due to differences in architecture and not due to the training. We conjecture future work might consider exploring different losses, involving truncation and collision, even when labels are present.

Why Hammersley? Other work has reasoned about what intermediate points to use when processing point clouds. When voting [23], the argument is, that the centers of bounding boxes are not part of the point set, and hence using a point set that is any subset of the input is not a good solution. While we do not vote, we also have chosen not to use points of the scene as the initial points. We also refrain from using any improved sampling of the surface, such as Poisson disk [13] sampling as we do not seek to cover any particular instance but space in general, covered by scenes uniformly.

How can the scene network be "better" than the loss network? As the loss network is only an approximation to the true loss, one might ask, how a scene network, trained with this loss network, can perform better than the loss network alone, e. g., how can it, consistently (Tbl. 1, 2, 4 and 5), outperform SLIDINGWINDOW?

Let us assume, that a scene network trained by a clean supervision signal can use global scene structure to solve the task. If now the supervision signal would start to be corrupted by noise, recent work has shown for images [18] or point clouds [12], that a neural network trained under noise will converge to a result that is very similar to the clean result: under $\mathcal{L}_2$ it will converge to the mean of the noise, under $\mathcal{L}_1$ to its median, etc. The amount of variance of that noise does not influence the result, what matters is that the noise is unbiased. In our case, this means if we were to have supervision by noisy bounding boxes, that would not change anything, except that the scene network training would converge slower but still to the mean or median of that noise distribution, which is, the correct result. So what was done in our training, by using a network to approximate the loss, means to just introduce another form of noise into the training.

6   Conclusion

We have suggested a novel training procedure to reduce the 3D labeling effort required to solve a 3D detection task. The key is to first learn a loss function on a small labeled local view of the data (objects), which is then used to drive a second learning procedure to capture global relations (scenes). The way to enlightenment here is to "find your center": the simple task of taking any piece of 3D scene and shifting it so it becomes centered around the closest object. Our analysis indicates that the scene network actually understands global scene structure not accessible to a sliding window. Our network achieves state of the art results, executes in a fraction of a second on large point clouds with typically only 5 % of the labeling effort. We have deduced what it means exactly to learn the loss function, the new challenges associated with this problem and proposed several solutions to overcome these challenges.

In future work, other tasks might benefit from similar decoupling of supervision labels and a learned loss, probably across other domains or modalities.

Bibliography

[1] Adler, J., Öktem, O.: Solving ill-posed inverse problems using iterative deep neural networks. Inverse Problems **33**(12) (2017)

[2] Armeni, I., Sax, S., Zamir, A.R., Savarese, S.: Joint 2D-3D-semantic data for indoor scene understanding. arXiv:1702.01105 (2017)

[3] Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al.: Shapenet: An information-rich 3D model repository. arXiv:1512.03012 (2015)

[4] Chen, X., Kundu, K., Zhu, Y., Berneshawi, A.G., Ma, H., Fidler, S., Urtasun, R.: 3D object proposals for accurate object class detection. In: NIPS (2015)

[5] Chen, Y., Liu, S., Shen, X., Jia, J.: Fast point R-CNN. In: ICCV (2019)

[6] Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: Scannet: Richly-annotated 3D reconstructions of indoor scenes. In: CVPR (2017)

[7] Engelcke, M., Rao, D., Wang, D.Z., Tong, C.H., Posner, I.: Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In: ICRA (2017)

[8] Feng, M., Gilani, S.Z., Wang, Y., Zhang, L., Mian, A.: Relation graph network for 3D object detection in point clouds. arXiv:1912.00202 (2019)

[9] Flynn, J., Broxton, M., Debevec, P., DuVall, M., Fyffe, G., Overbeck, R., Snavely, N., Tucker, R.: Deepview: View synthesis with learned gradient descent. In: CVPR (2019)

[10] Girshick, R.: Fast R-CNN. In: ICCV (2015)

[11] Griffiths, D., Boehm, J.: A review on deep learning techniques for 3d sensed data classification. Remote Sensing **11**(12) (2019)

[12] Hermosilla, P., Ritschel, T., Ropinski, T.: Total denoising: Unsupervised learning of 3d point cloud cleaning. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 52–60 (2019)

[13] Hermosilla, P., Ritschel, T., Vázquez, P.P., Vinacua, À., Ropinski, T.: Monte carlo convolution for learning on non-uniformly sampled point clouds. ACM Trans. Graph. (proc. SIGGRAPH Asia) **37**(6), 1–12 (2018)

[14] Hou, J., Dai, A., Nießner, M.: 3D-SIS: 3D semantic instance segmentation of RGB-D scans. In: CVPR (2019)

[15] Karpathy, A., Miller, S., Fei-Fei, L.: Object discovery in 3d scenes via shape analysis. In: ICRA (2013)

[16] Ku, J., Mozifian, M., Lee, J., Harakeh, A., Waslander, S.L.: Joint 3D proposal generation and object detection from view aggregation. In: IROS (2018)

[17] Lahoud, J., Ghanem, B., Pollefeys, M., Oswald, M.R.: 3d instance segmentation via multi-task metric learning. In: ICCV (2019)

[18] Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., Aila, T.: Noise2noise: Learning image restoration without clean data. arXiv:1803.04189 (2018)

[19] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot multibox detector. In: ECCV (2016)

[20] Maximov, M., Leal-Taixe, L., Fritz, M., Ritschel, T.: Deep appearance maps. In: ICCV (2019)

[21] Newell, A., Yang, K., Deng, J.: Stacked hourglass networks for human pose estimation. In: ECCV (2016)

[22] Qi, C.R., Chen, X., Litany, O., Guibas, L.J.: Imvotenet: Boosting 3d object detection in point clouds with image votes. In: arXiv preprint arXiv:2001.10692 (2020)

[23] Qi, C.R., Litany, O., He, K., Guibas, L.J.: Deep hough voting for 3D object detection in point clouds. In: ICCV (2019)

[24] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 652–660 (2017)

[25] Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. arXiv:1706.02413 (2017)

[26] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: CVPR (2016)

[27] Shi, S., Wang, X., Li, H.: PointRCNN: 3D object proposal generation and detection from point cloud. In: CVPR (2019)

[28] Song, S., Lichtenberg, S.P., Xiao, J.: Sun RGB-D: A RGB-D scene understanding benchmark suite. In: CVPR (2015)

[29] Song, S., Xiao, J.: Sliding shapes for 3D object detection in depth images. In: ECCV (2014)

[30] Song, S., Xiao, J.: Deep sliding shapes for amodal 3d object detection in RGB-D images. In: CVPR (2016)

[31] Weisstein, E.: Fixed point (2020), URL `http://mathworld.wolfram.com/FixedPoint.html`

[32] Weisstein, E.: Hammersley point set (2020), URL `http://mathworld.wolfram.com/HammersleyPointSet.html`

[33] Yang, B., Wang, J., Clark, R., Hu, Q., Wang, S., Markham, A., Trigoni, N.: Learning object bounding boxes for 3D instance segmentation on point clouds. arXiv:1906.01140 (2019)

[34] Zhou, X., Wang, D., Krähenbühl, P.: Objects as points. arXiv:1904.07850 (2019)

[35] Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: CVPR (2018)