Back to Reality: Learning Data-Efficient 3D Object Detector with Shape Guidance (Supplementary Material)

Xiuwei Xu, *Student Member, IEEE*, Ziwei Wang, *Student Member, IEEE*, Jie Zhou, *Senior Member, IEEE*, and Jiwen Lu, *Senior Member, IEEE*

Appendix A Overview

This supplementary material is organized as follows:

- Appendix B details the Approach section in the main paper.
- Appendix C details our augmentation strategy for small objects during training.
- Appendix D shows more statistics of ScanNet-md40 and Matterport3D-md40.
- Appendix E details the position-level annotation.

Appendix B Approach Details

In this section, we show the details in our BackToReality approach, which is divided into shape-guided **label enhancement** and virtual2real **domain adaptation**.

B.1 Label Enhancement

We show the exact definitions of some concepts appeared in Section 3.2.1 of the main paper as below.

Shape Properties: The MER is computed in XY plane, which is the minimum rectangle enclosing all the points of the object template. The SSH is the height of the largest surface on which other objects can stand. The CSS is a boolean value, indicating whether the supporting surface is similar with the MER (i.e. we can use the MER to approximate the supporting surface if CSS is true).

In order to calculate MER, we use the OpenCV [1] toolbox to calculate the MER of 2D point set. As OpenCV cannot be directly utilized to process point clouds, we first project the object templates to XY plane to acquire 2D point sets. Then we calculate the MER of a point set $S=\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$ as below:

$$(x,y,l,w,\theta)=\text{minAreaRect}(1000*S) \quad (1)$$

$$MER=(x,y,\frac{l}{1000},\frac{w}{1000},\theta) \quad (2)$$

where minAreaRect is a function in OpenCV, which takes integer 2D point set as input and returns a rectangle, and rectangle is represented by a quintuple $(x,y,length,width,\theta)$, which indicates the center coordinate, length, width and rotation angle of a rectangle. $1000*S$ means that we multiply all the coordinates in S by 1000 and then convert the coordinates from float to integer, which can reduce the rounding error.

To compute SSH, we first utilize Open3D [2] to get the normals of each point from point cloud. Then if the normal of a point is almost vertical (i.e. the normal's length along Z-axis is greater than 0.88), we record the coordinate of this point. After traversing all the points, we have recorded a list of coordinates. We sort the list according to the Z coordinate in ascending order, and the list of sorted Z coordinate is named as l_z. Then get a slice of l_z from index $\lfloor\frac{4}{5}len_z\rfloor$ to $\lfloor\frac{9}{10}len_z\rfloor$, where len_z denotes the length of l_z. SSH can be calculated by averaging this slice. Note that this algorithm suppose the supporter has a large supporting surface on its top, and it can tolerate 10% points higher than this surface.

To calculate CSS, we collect points which satisfy $SSH-\frac{1}{10}h<z<SSH+\frac{1}{10}h$ from the given object template, where h is the height of this object template. Then we project these points to XY plane and name them supporter points P_S. If P_S can almost fill the MER, the CSS is set to be $True$. To analyze the compactness, we use K-means algorithm to divide P_S into 2 clusters: P_{S1} and P_{S2}. Then we calculate the area of convex hull of P_{S1} and P_{S2}. The area is computed by using OpenCV:

$$A=\frac{\text{contourArea}(\text{convexHull}(1000*P))}{1000000} \quad (3)$$

where contourArea and convexHull are functions in OpenCV, P is a 2D point set and A is the area of P. The areas for P_{S1} and P_{S2} are A_1 and A_2 respectively. So we can compute CSS as below:

$$CSS=\begin{cases}True, & A_1+A_2>0.9*l*w\\ False, & otherwise\end{cases} \quad (4)$$

- *The authors are with the Beijing National Research Center for Information Science and Technology (BNRist), Department of Automation, Tsinghua University, Beijing 100084, China. E-mail: xxw21@@mails.tsinghua.edu.cn, wang-zw18@mails.tsinghua.edu.cn, jzhou@tsinghua.edu.cn, lujiwen@tsinghua.edu.cn. (Corresponding author: Jiwen Lu)*

where l and w are the length and width of the MER of this object template.

Segment Properties: Next we provide the definitions of horizontal segment, the area of segment and the height of segment.

For a segment, we define z as the Z coordinate of all the points on it. Then if $|maximum(z) - median(z)| < 0.2$ or $|minimum(z) - median(z)| < 0.2$, we consider this segment is horizontal. To calculate the area of segment, we directly utilize (3) and take all points on the segment as input (ignore the Z coordinates of points). To compute the height of a segment, we follow the same procedure as computing SSH: we first calculate the normals and pick out points with normals that are almost vertical, and then we pick out the Z coordinates of these points and acquire a list l_z. The segment's height is defined as the mean of l_z.

B.2 Domain Adaptation

We first provide detailed definition of L_3. Then we show the architectures of our center refinement module and the two discriminators.

For weakly-supervised training, as only objects' centers and semantic classes are available, we set L_3 as a simpler version of L_2:

$$L_3 = L_f + L_i,\ L_f = L_s + L_o + L_c \tag{5}$$

L_f is used to supervise the final prediction, where L_s and L_o are the cross entropy losses for semantic labels and objectness scores, and L_c is defined as:

$$L_c = \sum_i max(||C_{gi} - C_i||_2 - \lambda S_{gi}, 0) \tag{6}$$

which denotes the hinge loss for centers. C_i is the i-th predicted center, C_{gi} is the nearest ground-truth center to C_i, and S_{gi} indicates the average size for the semantic class of this object. We set $\lambda = 0.05$ to approximate the labeling error of centers. For L_i, we only make use of the center coordinates to weakly supervise the intermediate process of training. For example, in VoteNet [3], the detection module predicts votes from the semantic features and aggregate them to generate object proposals, in which voting coordinates are the intermediate variables need to be supervised. Here we utilize the Chamfer Distance between the voting coordinates and the ground-truth center coordinates to supervise the voting. In GroupFree3D [4], the detection module utilize KPS to sample the semantic features and generate initial object proposals, where the sampled points require supervision. Originally the KPS operation requires us to sample the nearest k points to the object center from the point cloud belong to this object. However, we weaken this requirement and sample the nearest k points without any constraints.

For the center refinement module, we adopt the Set Abstraction (SA) layer [5] to extract feature from the local KNN graph. Then a MLP is utilized to predict center offset from the feature. The SA layer first concatenates the relative coordinates between the center and its neighbors to the features of the neighbors, which is followed by a shared MLP ($MLP(256, 128)$[1]) and a channel-wise max-pooling layer. The pooled feature contains the local information of the center, which is then concatenated with the one-hot vector of the center's semantic class (we name the feature after concatenation as center feature). We utilize another MLP ($MLP(64, 3)$) to predict the center offset from the center feature. For the global and proposal discriminators, we show their architectures in Figure 1.

1. Numbers in bracket are output layer sizes. Batchnorm is used for all layers with ReLU except for the final prediction layer in $MLP(64, 3)$.

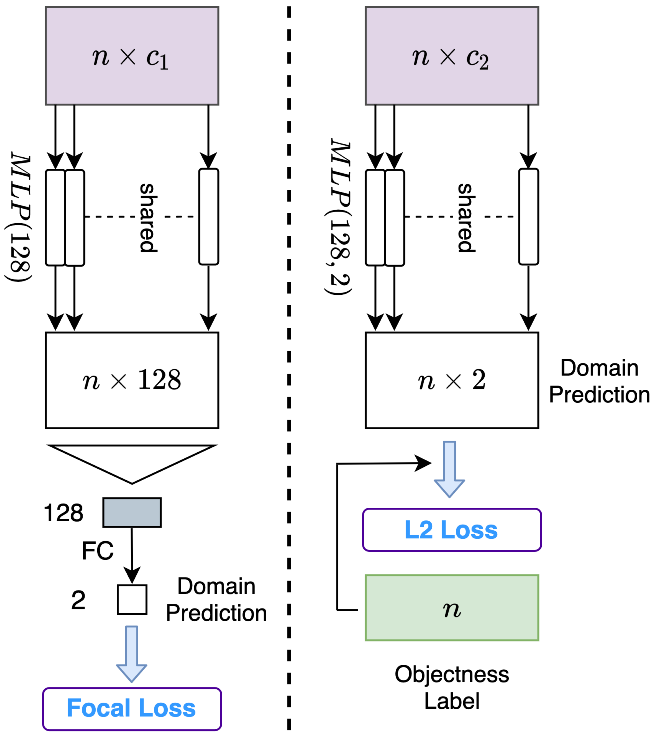

Fig. 1. Architecture of the global and proposal discriminators. (Global on the left, proposal on the right.)

Appendix C Augmentation Strategy

As the number of scenes which contain small objects[2] and the probability of small objects being sampled are relatively smaller than others, it is difficult for the detector to learn how to locate small objects in complex scenes. Therefore we utilize an augmentation strategy similar to [6] to handle the problem.

During trianing, we oversample the virtual scenes which contain small objects twice in each epoch. We further copy-paste small objects to the oversampled virtual scenes: for each small object, we copy it with a probability of 0.75 and paste it randomly in the scene (the pasted center must be in the axis-aligned bounding box of the whole scene). Then we apply gravity and collision contraints and control the densities of these added small objects as mentioned in the virtual scene generation method.

Apart from small objects, we also consider the scarce objects[3], as the number of them is relatively small and thus the detector is not sufficiently trained on these categories. We add the scarce objects to the oversampled virtual scenes to expand the number of them. We first decide how many objects of each scarce category we should add according to Table 2 in the main paper, where we set 40, 70, 15, 55 and 50 for bathtub, bench, dresser, laptop and wardrobe respectively. Then we choose scenes which are suitable

2. Small objects are {bottle, cup, keyboard}.
3. Scarce objects are {bathtub, bench, dresser, laptop, wardrobe}.

TABLE 1
Number of objects in each category in the training set and validation set of ScanNet and Matterport3D, and average number of points of objects in each category in the real scenes and the virtual scenes. N_{object}, N_{point}, # TR, # VD, # R and # V refer to number of objects, number of points, training set, validation set, real scenes and virtual scenes respectively.

	Property	Split	Bath-tub	Bed	Bench	Book-shelf	Bottle	Chair	Cup	Cur-tain	Desk	Door	Dresser	Key-board	Lamp	Laptop	Monitor	Night-stand	Plant	Sofa	Stool	Table	Toilet	Ward-robe
ScanNet	N_{object}	# TR	113	308	58	786	234	4357	132	408	551	2028	174	193	376	86	574	190	293	406	315	1526	201	98
		# VD	31	81	21	234	41	1368	34	95	127	467	43	53	83	25	191	34	50	97	51	407	58	19
	N_{point}	# R	2941	3905	1015	2679	101	726	66	2919	1525	1110	1274	74	272	173	370	700	792	2718	525	1282	1445	2762
		# V	6891	8683	4097	6258	162	2135	91	5495	5004	6048	2703	480	609	343	939	1088	1249	7250	1391	5421	3716	6105
Matterport3D	N_{object}	# TR	94	200	163	–	–	2159	–	532	159	1815	116	–	–	–	–	205	–	257	252	1136	204	–
		# VD	13	25	14	–	–	271	–	82	15	283	17	–	–	–	–	34	–	37	43	189	25	–
	N_{point}	# R	3089	3670	1145	–	–	731	–	2146	1436	1864	1474	–	–	–	–	737	–	2014	455	1093	1499	–
		# V	4819	7300	3199	–	–	1480	–	4988	3283	5775	3838	–	–	–	–	943	–	4824	678	2607	2125	–

for adding these objects by calculating the value of correlation between scenes and scarce categories as below:

$$Corr(s,c) = \sum_{i=1}^{22} l_{s_i}(v_{c_i} - r) \tag{7}$$

where s indicates a scene and c denotes a scarce category. l_s is a 22-dimensional boolean vector where l_{s_i} indicates whether there is an object of the i-th category in s. v_c is a 22-dimensional vector which indicates the correlation between c and other categories:

$$v_{c_i} = \begin{cases} \frac{Num(i,Index(c))}{Num(Index(c))}, & i \neq Index(c) \\ 0, & i = Index(c) \end{cases} \tag{8}$$

where $Num(...)$ is a function, whose input is a set of indexs of category and output is the number of scenes which contain objects in all the input categories. The larger v_{c_i}, the stronger the correlation between c and the i-th category. As we hope the highly correlated scenes for c do not contain too many categories with low v_{c_i}, we introduce a penalty term r to reduce the value of $Corr(s,c)$ when there are a large number of categories weakly correlated to c in s. We set $r = 0.25$ in our experiments.

Appendix D
Statistics of Datasets

We show number of objects in each category in the training set and validation set of ScanNet and Matterport3D, and average number of points of objects in each category in the real scenes and the virtual scenes in Table 1.

Appendix E
Position-level Annotation

In this work, we generate the position-level annotations by randomly jittering the original centers according to the labeling error. The labeling time and error are estimated by a user study.

Labeling tool: We develop the labeling tool based on MeshLab [7]. As shown in Figure 2 (a), we first generate dense meshgrids (the black points) surrounding the scene which help us to fit 3D lines in the following steps. Then for an object to be annotated, we first select a view and crop its 2D center with MeshLab. This operation will crop a small proportion of points along the ray between the focal point and the 2D center, from which we can fit a 3D line. This process is shown in Figure 2 (b). Next we need to select a point on this 3D line. A simple way to achieve this is to change a view and follow the same procedure to fit another line. Then we can compute the closest point to the second line on the first line, which is the annotated object center. Below we detail our labeling tool and the user study.

TABLE 2
User study: the average labeling time and error rate for some categories.

	bed	door	table	toilet	keyboard
Time (s)	3.8	2.9	4.4	5.3	2.6
Error rate (%)	7.7	9.2	5.6	10.2	7.0

Labeling strategy: There needs to be a space angle larger than $45°$ between the two perspectives in order to ensure the labeling error is small. For objects which can be seen in BEV (most objects), we label the first line vertically, which is fast and accurate.

User study: We hire 5 annotators to annotate 10 scenes, each annotator focusing on a specified category. We choose bed, door, table, toilet and keyboard as the 5 categories. After annotating, we compute the average labeling time and error rate[4] for each category, as shown in Table 2. This statistics support the reported 10% error and 5 secs annotation time. We also note that toilets are harder to label. This is because they are usually located at the corner of a room, which is hard for annotators to select proper perspective. A more user-friendly UI design will further reduce the annotating time and error.

References

[1] "Opencv," [EB/OL], https://opencv.org/. 1
[2] "Open3d: A modern library for 3d data processing," [EB/OL], http://www.open3d.org/. 1
[3] Qi, Charles R. and Litany, Or and He, Kaiming and Guibas, Leonidas J., "Deep hough voting for 3d object detection in point clouds," in *ICCV*, 2019, pp. 9277–9286. 2
[4] Liu, Ze and Zhang, Zheng and Cao, Yue and Hu, Han and Tong, Xin, "Group-free 3d object detection via transformers," *arXiv preprint arXiv:2104.00678*, 2021. 2
[5] Qi, Charles Ruizhongtai and Yi, Li and Su, Hao and Guibas, Leonidas J, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space." 2
[6] Kisantal, Mate and Wojna, Zbigniew and Murawski, Jakub and Naruniec, Jacek and Cho, Kyunghyun, "Augmentation for small object detection," *arXiv preprint arXiv:1902.07296*, 2019. 2

4. The error rate is defined as the ratio of the distance from the annotated center to the real center to the doubled diagonal length of the object.

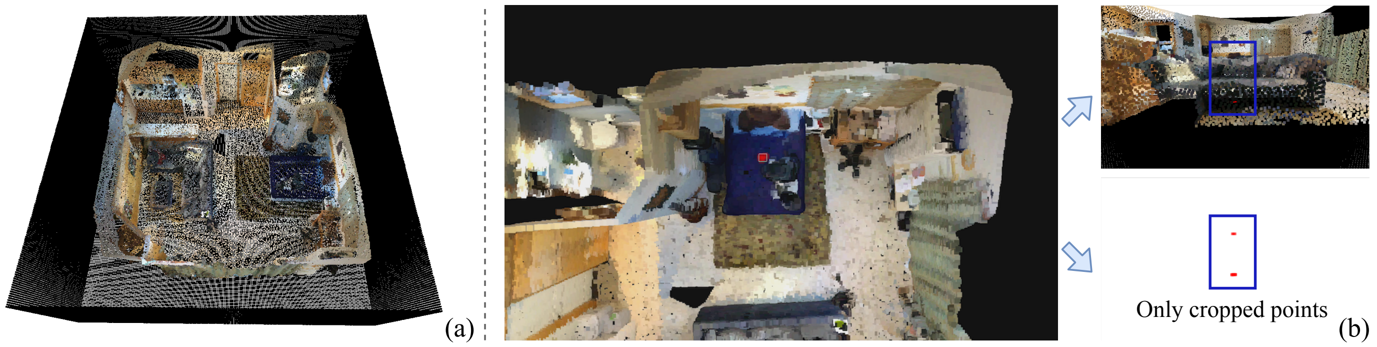

Fig. 2. Demonstration of our labeling process. (a) The labeling tool. We generate dense meshgrids surrounding the scene to help fitting 3D lines. (b) We choose a view to crop the 2D center of object. The cropped points are then used to fit a 3D line. We can repeat this operation and compute the center of object with the two lines.

[7] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *EICC*, V. Scarano, R. D. Chiara, and U. Erra, Eds. The Eurographics Association, 2008. 3