Supplementary Material for itKD: Interchange Transfer-based Knowledge Distillation for 3D Object Detection

A. Student configuration

Table 1. The number of parameters of the teacher, the student (1/2), and the student (1/4).

Model	Point cloud encoder	Backbone	Head	Total
Teacher	4,608	4,806,400	413,003	5,224,011 (5.2M)
Student (1/2)	4,608	1,212,288	302,411	1,519,307 (1.5M)
Student (1/4)	4,608	308,416	247,115	560,139 (0.6M)

Conventional KD methods for 3D object detection focused on improving performance or reducing latency. However, the main purpose of our method is how to reduce the parameters of 3D object detector. In this respect, we investigate which component of the backbone architecture has most parameters as shown in Table 1. Since the backbone has 4.8M parameters, which occupies about 92% of the 5.2M parameters of the teacher network, We apply channel reduction to each layers of backbone because channel reduction maintains performance better than depth reduction on detection task [2] [4]. Finally, our student (1/4) has 8.7× less parameters and student (1/2) has 3.5× less parameters.

B. Performance of the student $1/2$

Table 2. Comparison with different KD methods in Waymo and nuScenes validation set.

Method	Waymo				nuScenes	
	Vehicle	Pedestrian	Cyclist	Total	NDS	mAP
Teacher [7]	65.11	54.99	60.28	60.13	59.45	48.83
Student	62.19	53.19	56.45	57.28	56.79	45.45
Baseline	63.16	53.81	57.21	58.06	57.95	46.78
FitNet [3]	63.45	54.10	57.31	58.29	57.97	46.78
EOD-KD [1]	62.80	53.70	57.27	57.92	58.07	46.83
TOFD [8]	60.99	52.98	57.44	57.14	57.54	46.10
SE-SSD [9]	63.02	54.21	57.86	58.36	57.30	46.03
Obj. DGCNN [5]	63.07	54.23	57.77	58.36	57.96	46.92
SparseKD [6]	62.57	53.75	58.06	58.13	57.59	46.54
Ours	**63.92**	**54.53**	**58.66**	**59.04**	**58.32**	**47.18**

To verify the generality of our method, we compare the student (1/2) with other KD methods on Waymo and nuScenes validation set. Table 2 shows the mAPH of level2 performance of KD methods on Waymo, and NDS and mAP on nuScenes. Our student (1/2) shows better performance than other methods on both datasets. In conclusion, we confirm that our method has generality regardless of the parameter reduction ratio.

C. Pseudocode

Algorithm 1: PyTorch-style pseudocode for the channel-wise autoencoder

```
# c_t:  Channel size of the teacher's backbone
 output
# c_s:  Channel size of the student's backbone
 output
# c_e:  Channel size of the compressed
 representation
# x_t:  The map-view feature of the teacher
 network
# x_s:  The map-view feature of the student
 network

# Define the channel-wise autoencoder as class
class ChannelWiseAE(nn.Module):
    def __init__(self, c_t, c_s, c_e):
        # Sampling layers to adapt channel size
        self.downs = nn.Conv2d(c_t, c_s, (1, 1))
        self.ups = nn.Conv2d(c_s, c_t, (1, 1))
        # Build encoder layers
        self.encoder =
            nn.Sequential(
                nn.Conv2d(c_t, 128, (1, 1)),
                nn.Conv2d(128, 64, (1, 1)),
                nn.Conv2d(64, c_e, (1, 1)))
        # Build decoder layers
        self.decoder =
            nn.Sequential(
                nn.Conv2d(c_e, 64, (1, 1)),
                nn.Conv2d(64, 128, (1, 1)),
                nn.Conv2d(128, c_t, (1, 1)))
    def forward(self, x_t, x_s):
        # Pass through the autoencoder
        x_s = self.ups(x_s)
        comp_t = self.encoder(x_t)
        comp_s = self.encoder(x_s)
        decomp_t = self.decoder(comp_t)
        decomp_s = self.decoder(comp_s)
        decomp_t = self.downs(decomp_t)
        # Calculate loss values
        comp_repr = F.l1_loss(comp_s, comp_t)
        decomp_s2t = F.l1_loss(s_decode, x_t)
        decomp_t2s = F.l1_loss(t_decode, x_s)
        # Return total loss
        return comp_repr + decomp_s2t + decomp_t2s
```

Algorithm 1 and 2 show PyTorch-style pseudo-code for the channel-wise autoencoder and the head relation-aware self-attention, respectively. The interchange transfer and

Algorithm 2: PyTorch-style pseudocode for the relation-aware self-attention

```
# x_t:  Detection results of the teacher network
# x_s:  Detection results of the student network
# fusion:  1×1 convolution layer for fusion on
  channel dimension
# ind:  Index of objects’ location

# Define the self-attention
def self_attention(x):
    # Calculate attention score
    score = F.softmax(torch.matmul(x.transpose(-2,
     -1), x) / torch.sqrt(x.size(-2)), dim=-2)
    return torch.matmul(x, score)

# Define the head relation-aware self-attention
def relation_aware_self_attention(x):
    # Generate feature sequences
    seq = x.gather(ind)
    # Apply the intra-head relation attention
    for seq_head in seq:
        intra_at1tention.append(self_attention(seq_head))
    intra_attention = torch.cat(inter_attention,
     dim=1)
    # Apply the inter-head relation attention
    inter_attention = (self_attention(seq))
    # Pass through the fusion layer
    attention =
        fusion(torch.cat([intra_attention,
         inter_attention], dim=1))
    return attention

# Apply the relation-aware self-attention
rasa_t = relation_aware_self_attention(x_t)
rasa_s = relation_aware_self_attention(x_s)
# Calculate the attentive head loss
attentive_head = F.l1_loss(rasa_s, rasa_t)
# Return the loss
return attentive_head
```

the compressed representation loss are included in Algorithm 1. Algorithm 2 contains the head attention loss. As we described in section 3.3 of the main paper, we use the l_1 loss as a similarity function.

D. Visualization of the student feature

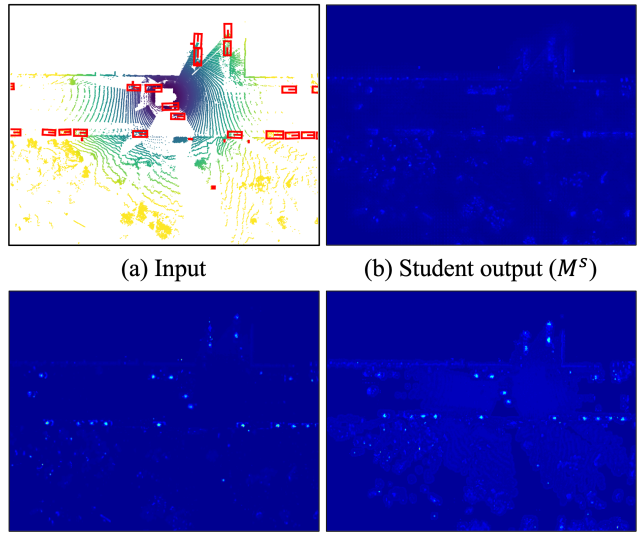
We visualize the output features of the student, the encoder, and the decoder, which take the same input as in Fig. 4 of the main paper. As shown in Fig. 1, the visualization results show that both objects and backgrounds are well-activated.

E. Inference time

Table 3. Lantency and FPS.

Model	latency	FPS
Teacher	46.0	21.7
Ours	23.4	42.7

Table 3 shows the inference time of the teacher and our student (Ours, $1/4$). The inference time is averaged 100 frames with a NVIDIA Titan V. Our student network achieves a computation speed of 42.7 FPS.



(a) Input (b) Student output (M^s)

(c) Encoder output (d) Decoder output

Figure 1. **Output feature visualization of the student backbone.**

Table 4. Performances on the voxel-based encoder.

Method	Teacher	Student	Baseline	SparseKD	Ours
mAPH/L2	65.50	63.26	64.03	64.05	**64.26**

F. Performance of voxel-based encoders

We made additional experiments in Table 4 that shows the results of the voxel-based encoder. Our method shows 64.26% mAPH/L2 and outperforms SparseKD, which is the latest KD method for 3D object detectors.

G. Limitation

The limitation of the interchange transfer lies in the fact that both the teacher and the student networks must maintain the same spatial resolution, as the interchange transfer is based on feature-based knowledge distillation. We also note that using the autoencoder often requires additional effort for identifying the proper network structure or its hyper-parameters for the different 3D object detection, but we believe that the deviations of the optimal hyper-parameters are not high.

H. Potential negative societal impacts

Our KD method aims to make an efficient 3D object detection network, which is crucial for the autonomous driving system that requires real-time response. One potential negative societal impact of our method is that the quantitative performance of the student network follows similarly to that of the teacher network; also, it has not been confirmed whether there are any parts that can be fatal to the safety of the autonomous driving system in the wild.

References

[1] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. *Advances in neural information processing systems*, 30, 2017. 1

[2] Quanquan Li, Shengying Jin, and Junjie Yan. Mimicking very efficient network for object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 6356–6364, 2017. 1

[3] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 1

[4] Tao Wang, Li Yuan, Xiaopeng Zhang, and Jiashi Feng. Distilling object detectors with fine-grained feature imitation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4933–4942, 2019. 1

[5] Yue Wang and Justin M Solomon. Object dgcnn: 3d object detection using dynamic graphs. *Advances in Neural Information Processing Systems*, 34, 2021. 1

[6] Jihan Yang, Shaoshuai Shi, Runyu Ding, Zhe Wang, and Xiaojuan Qi. Towards efficient 3d object detection with knowledge distillation. *arXiv preprint arXiv:2205.15156*, 2022. 1

[7] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11784–11793, 2021. 1

[8] Linfeng Zhang, Yukang Shi, Zuoqiang Shi, Kaisheng Ma, and Chenglong Bao. Task-oriented feature distillation. *Advances in Neural Information Processing Systems*, 33:14759–14771, 2020. 1

[9] Wu Zheng, Weiliang Tang, Li Jiang, and Chi-Wing Fu. Se-ssd: Self-ensembling single-stage object detector from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14494–14503, 2021. 1