Cross-Modality Knowledge Distillation Network for Monocular 3D Object Detection - Supplementary Materials

Yu Hong[1], Hang Dai[2]*, and Yong Ding[1]*

[1] Zhejiang University, Zhejiang, China
[2] MBZUAI, Abu Dhabi, UAE
yuhong_1999@zju.edu.cn hang.dai@mbzuai.ac.ae dingy@vlsi.zju.edu.cn
* Corresponding authors.

1 Details of BEV Feature Learning

In this part, we describe the detailed process of generating the Bird's-Eye-View (BEV) feature maps in the two branches of our cross-modality knowledge distillation network. We take the detailed settings on KITTI [3] with ResNet-101 [4] backbone as an example for description.

1.1 LiDAR-based BEV Feature Learning

Voxelization. We use the LiDAR-based 3D detector SECOND [10] to generate the LiDAR BEV feature map. Given the point clouds of range (L, W, H), where L, W and H are the length, width and height of the 3D space corresponding to the x, y and z axes respectively, we first filter out the points that can be projected onto the image plane, i.e., field of view (FOV) points. Then, we subdivide the 3D space into equal 3D voxels of size (dx, dy, dz) to obtain the original voxels of size $(L/dx, W/dy, H/dz)$. In this paper, the range for the point clouds in the 3D space is set to $[2.0, 46.8] \times [-30.08, 30.08] \times [-3.0, 1.0]$ meters. The voxel size is set to $(0.04, 0.04, 0.1)$ meters, and the size of the original LiDAR voxel volume is $(1120, 1504, 40)$.

Voxel Feature Extraction. The features of the original LiDAR voxels are the mean values of the point coordinates in each voxel, which are denoted as $\tilde{F}_L^v \in \mathbb{R}^{1120\times1504\times40\times4}$. After voxelization, $\tilde{F}_L^v$ is fed to a sparse 3D convolution backbone, gradually converting the original voxel features into higher dimensional space with $1\times$, $2\times$, $4\times$, $8\times$ down-sample rates using a series of sparse 3D convolution blocks. The output features of the voxel backbone are denoted as $F_L^v \in \mathbb{R}^{140\times188\times2\times64}$.

Map to BEV. The voxel features $F_L^v \in \mathbb{R}^{140\times188\times2\times64}$ are collapsed to a 2D BEV feature map by stacking the features in height dimension to obtain the LiDAR BEV features $F_L^{bev} \in \mathbb{R}^{140\times188\times2*64}$.

We illustrate the BEV feature map generation process of the LiDAR-based method in the top row of Fig. 1.

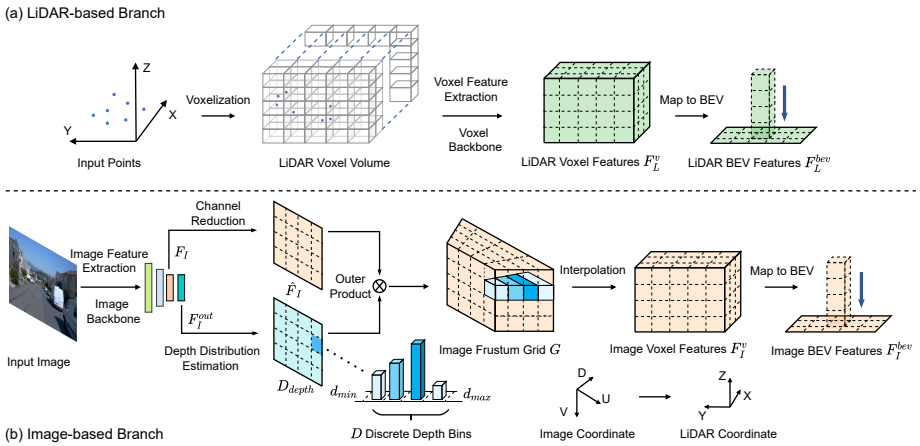

Fig. 1. BEV feature map generation. (a) The LiDAR-based branch. (b) The image-based branch.

1.2 Image-based BEV Feature Learning

Image Feature Extraction. We follow CaDDN [8] to generate the image BEV feature map. Given the monocular image $I \in \mathbb{R}^{W \times H \times 3}$, where W and H are around 1242 and 375 in KITTI [3], we first use the image backbone ResNet-101 [4] on it to extract the image features. And we use the intermediate features with down-sample rate 4 from $layer1$ as the image features $F_I \in \mathbb{R}^{W_I \times H_I \times C}$, where $W_I = 311$, $H_I = 94$ and $C = 256$.

Voxelization via Depth Distribution Estimation. The image features $F_I \in \mathbb{R}^{W_I \times H_I \times C}$ go through a channel reduction network to obtain $\hat{F}_I \in \mathbb{R}^{W_I \times H_I \times C'}$, where $C' = 64$ is the number of reduced feature channels. For each position in $\hat{F}_I$, we predict its depth in a classification manner. Specifically, the continuous depth range $[d_{min}, d_{max}]$ is subdivided into D discrete bins using linear-increasing discretization (LID) as:

$$d_i = d_{min} + \frac{d_{max} - d_{min}}{D(D+1)} \cdot i(i+1), i \in [0, D] \tag{1}$$

where d_i is the discrete depth value of the i-th depth bin, $d_{min} = 2.0\,m$, $d_{max} = 46.8\,m$ and $D = 80$. We use the depth distribution estimation head DeepLabV3 [1] after the output features of the image backbone $F_I^{out} \in \mathbb{R}^{W_I^{out} \times H_I^{out} \times C^{out}}$ where $W_I^{out} = 156$, $H_I^{out} = 47$ and $C^{out} = 2048$ to predict pixel-wise depth distribution $D_{depth} \in \mathbb{R}^{W_I \times H_I \times D}$ for each location in $\hat{F}_I$. We then calculate the outer product of $\hat{F}_I \in \mathbb{R}^{W_I \times H_I \times C'}$ and $D_{depth} \in \mathbb{R}^{W_I \times H_I \times D}$ to construct a frustum grid $G \in \mathbb{R}^{W_I \times H_I \times D \times C'}$, where the feature of each location in $\hat{F}_I$ is placed at each predicted depth bin and weighted by the predicted probabilities. The frustum grid G is not in the same geometry shape as the voxel volume in the LiDAR-based branch and it is represented in image coordinate, so we need to

transform it into a cuboid voxel volume in the LiDAR coordinate. Specifically, we first construct a voxel volume for the image-based branch in LiDAR coordinates, where the range of the 3D space is the same as the LiDAR branch, and the voxel size is set to $(0.16, 0.16, 0.16)$ meters. For each position in the image voxel volume, we project its center (x, y, z) into the image coordinate to get (u, v, d), and apply a trilinear interpolation operation to obtain the features, and we thus get the voxel features $F_I^v \in \mathbb{R}^{280\times376\times25\times64}$ in the image branch.

Map to BEV. The voxel features $F_I^v \in \mathbb{R}^{280\times376\times25\times64}$ are collapsed to a 2D BEV feature map by stacking the features in height dimension to obtain $\tilde{F}_I^{bev} \in \mathbb{R}^{280\times376\times25*64}$, which then goes through a channel reduction network to get the image BEV features $F_I^{bev} \in \mathbb{R}^{280\times376\times64}$.

We illustrate the BEV feature map generation process for the image-based method in the bottom row of Fig. 1.

2 Details of Domain Adaptation Module

The image BEV features F_I^{bev} are different from LiDAR BEV features F_L^{bev} in spatial-wise and channel-wise feature distribution due to the fact that they come from different input modalities with different backbones. We employ a domain adaptation (DA) module to align the feature distribution of F_I^{bev} to that of F_L^{bev} and enhance F_I^{bev} at the meantime. Since the feature dimension of $F_I^{bev} \in \mathbb{R}^{280\times376\times64}$ may not match with that of $F_L^{bev} \in \mathbb{R}^{140\times188\times128}$, we first apply a convolution layer on F_I^{bev} to align the BEV feature dimension:

$$F_I^{bev'} = Conv(F_I^{bev}) \tag{2}$$

where $F_I^{bev'} \in \mathbb{R}^{140\times188\times128}$ has the same shape as F_L^{bev}. Then, we stack five Self-Calibrated Blocks [6] after F_I^{bev} to apply spatial-wise and channel-wise transformations:

$$\hat{F}_I^{bev} = DA(F_I^{bev'}) \tag{3}$$

where $\hat{F}_I^{bev} \in \mathbb{R}^{140\times188\times128}$ are the enhanced BEV features after the DA module.

Fig. 2 illustrates the detailed network architecture of the Self-Calibrated Block. The input $X \in \mathbb{R}^{H\times W\times C}$ is first split into two portions $X_1 \in \mathbb{R}^{H\times W\times \frac{C}{2}}$ and $X_2 \in \mathbb{R}^{H\times W\times \frac{C}{2}}$ through two 1×1 convolution layers:

$$X_1 = Conv_{1\times1}(X) \tag{4}$$
$$X_2 = Conv_{1\times1}(X) \tag{5}$$

A branch of X_1 goes through a series of layers for the self-attention $A \in \mathbb{R}^{H\times W\times \frac{C}{2}}$:

$$X_4 = Up(Conv3\times3(Down(X_1))) \tag{6}$$
$$A = Sigmoid(X_1 + X_4) \tag{7}$$

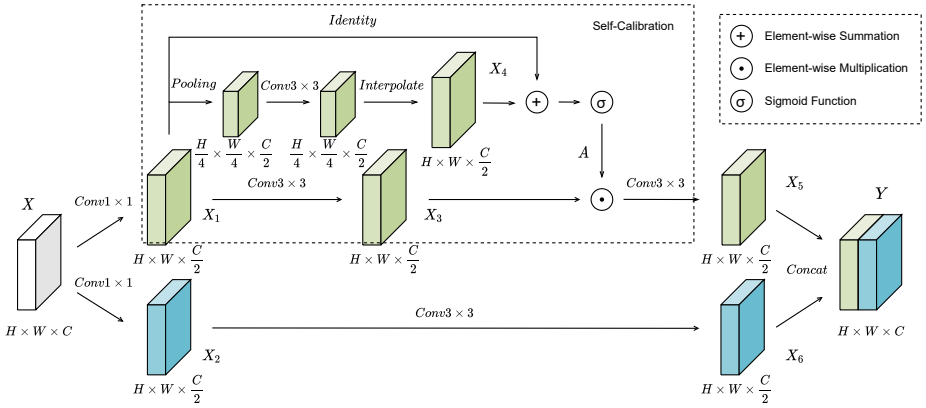

Fig. 2. Self-Calibrated Block

where $Down$ denotes a $4\times$ average pooling layer and Up denotes an interpolation layer. X_1 itself goes through a 3×3 convolution layer to get $X_3 \in \mathbb{R}^{H \times W \times \frac{C}{2}}$:

$$X_3 = Conv_{3\times3}(X_1) \tag{8}$$

which is weighted by the self-attention $A \in [0,1]^{H \times W \times \frac{C}{2}}$ and then goes through another 3×3 convolution layer to get $X_5 \in \mathbb{R}^{H \times W \times \frac{C}{2}}$:

$$X_5 = Conv_{3\times3}(X_3 \cdot A) \tag{9}$$

X_2 goes through a 3×3 convolution layer to get $X_6 \in \mathbb{R}^{H \times W \times \frac{C}{2}}$:

$$X_6 = Conv_{3\times3}(X_2) \tag{10}$$

And finally we concatenate $X_5 \in \mathbb{R}^{H \times W \times \frac{C}{2}}$ and $X_6 \in \mathbb{R}^{H \times W \times \frac{C}{2}}$ for the output $Y \in \mathbb{R}^{H \times W \times C}$:

$$Y = Concat(X_5, X_6) \tag{11}$$

Note that the convolution layers are followed by BatchNorm and ReLU operations.

3 Generalization Study with Different Backbones

In this part, we conduct experiments on the generalization ability of CMKD using different student models. For the network structure after the BEV feature map, we simply use the most basic one in 3D detection, so we mainly change the backbone of the model for comparison. Specifically, we choose backbones with different weights and different structures and compare the performance of CMKD, including running speed, running memory and $3D\,AP$. On the one hand, we want to show the performance of CMKD using backbones with different

Table 1. Generalization study of CMKD using backbones with different weights and different structures.

Backbone	Speed (fps)	Memory (G)	$3D\ AP$		
			Easy	Moderate	Hard
ResNet-101 [4]	7.5	4.3	30.2	**21.5**	**19.4**
ResNet-50 [4]	10.1	4.1	30.4	21.3	19.0
EfficientNet-b5 [9]	20.8	2.2	**30.8**	20.4	18.5
EfficientNet-b3 [9]	21.6	2.1	30.7	20.5	17.9
ConvNeXt-B [7]	23.6	2.8	30.6	20.7	18.5
ConvNeXt-S [7]	26.7	2.4	29.7	20.2	17.8
MobileNet [5]	30.0	2.6	29.8	20.5	17.8

structures, on the other hand, we want to show a trade-off comparison of speed and accuracy. The running speed and memory are tested on a single NVIDIA 3090 GPU with the batch size of 1, the $3D\ AP$ is tested on KITTI *val*.

The results are shown in Tab. 1. We use backbones with different weights and different structures for different versions of CMKD. Among them, there are both heavy and deep models (running speed < 10 fps), and light and shallow models (running speed > 20 fps, which can meet the requirement for real-time applications). As can be seen from the table, for the Easy class, the performance gap between different models is not large, and some light-weight models perform even better than the heavy-weight ones. For the Moderate and Hard classes, the heavy-weight models perform better than the light-weight ones, but the performance of the light-weight models is still not bad.

The above experiments, on the one hand, prove that our framework has good generalization performance and can cooperate with various backbones with different structures and weights to meet the needs of different application scenarios. On the other hand, it also highlights our main point of this work, that is, what we emphasize is the idea of our cross-modality knowledge distillation (CMKD) framework, not a specific model to be used in the framework.

4 Potential Limitation of CMKD: Soft Label Quality Matters

To make our work more comprehensive and complete, we proactively explore the limitations of CMKD and provide our solution. We notice that CMKD may have the following limitation, i.e., soft label quality matters.

Looking at the results in Tab. 2, we find that the performance of CMKD* (with $\sim 42k$ training samples) on Car and Cyclist is a lot better than CMKD (with $\sim 7.5k$ training samples), while the performance on Pedestrian is just the opposite, i.e., more training samples lead to worse results. This is due to the large gap between the soft label qualities of Car, Cyclist and Pedestrian. The typical performances of LiDAR-based detectors for Car, Cyclist and Pedestrian on KITTI leaderboard in Moderate level are around 80, 70 and 40, and the

Table 2. Results for Car, Cyclist and Pedestrian on KITTI *test* set. CMKD is trained with the official training set KITTI *trainval* ($\sim 7.5k$) and CMKD* is trained with the unlabeled KITTI Raw ($\sim 42k$). With unlabeled data from KITTI Raw, the performance for Car and Cyclist improves significantly, but the performance for Pedestrian instead decreases.

Class	Methods	$3D\,AP$			$BEV\,AP$		
		Easy	Moderate	Hard	Easy	Moderate	Hard
Car	CMKD	25.09	16.99	15.30	33.69	23.10	20.67
	CMKD*	**28.55**	**18.69**	**16.77**	**38.98**	**25.82**	**22.80**
Cyclist	CMKD	9.60	5.24	4.50	12.53	7.24	6.21
	CMKD*	**12.52**	**6.67**	**6.34**	**14.66**	**8.15**	**7.23**
Pedestrian	CMKD	**17.79**	**11.69**	**10.09**	**20.42**	**13.47**	**11.64**
	CMKD*	13.94	8.79	7.42	16.03	10.28	8.85

quality of predictions for Pedestrian is not at the same level as Car and Cyclist at all. That is, the soft labels provided by the teacher model for Pedestrian themselves are of very low quality, which can not serve as good guidance for the student model. The training of our framework on unlabeled data is under the assumption that the soft labels provided by the teacher model are of sufficient quality, which is the case for Car and Cyclist but not Pedestrian.

To verify the above discussion, we conduct additional experiments. Specifically, we choose two categories, Car and Pedestrian, and use hard label and soft label from KITTI *train* to supervise them respectively and report the performance on KITTI *val* for comparative experiments. For soft labels, we choose the early-stopped teacher model epochs whose performance on KITTI *val* are close to the typical one on the KITTI *test* set ($3D\,AP \approx 80\%$ for Car and $3D\,AP \approx 40\%$ for Pedestrian), in order to simulate the soft label quality provided by the teacher model on unlabeled data.

Table 3. Comparison between hard labels and soft labels used in $\mathcal{L}_{res}$. Among them, the soft labels for Car are of sufficient quality, and the soft labels for Pedestrian are of insufficient quality. We choose the early-stopped teacher model epochs to simulate the soft label quality that can be provided on unlabeled data.

Settings	$3D\,AP$		
	Easy	Moderate	Hard
Car, Sufficient Quality Soft Labels			
Hard Labels	23.20	15.78	13.77
Soft Labels	**23.85**	**16.22**	**14.33**
Pedestrian, Insufficient Quality Soft Labels			
Hard Labels	**12.24**	**8.65**	**6.82**
Soft Labels	4.57	3.20	2.47

As can be seen from Tab. 3, for Car, sufficient quality soft labels can provide useful information, and the results using soft labels are better than using hard labels. But for Pedestrian, insufficient quality soft labels can not provide effective guidance, so the results are far worse than using hard labels. When we train CMKD on unlabeled data, the teacher model can extract beneficial information for Car from the massive unlabeled data and transfer it to the student model, thereby boosting the performance of the student model. But for Pedestrian, the soft labels provided by the teacher model themselves are of insufficient quality, which can not serve as good guidance for the student model, and on the contrary reduce the performance of the student model. It is for this reason that the results in Tab. 2 appear. Based on the above experiments and discussions, when the quality of the soft label is bad, the solution we provide is to change soft labels to hard labels in the loss term $\mathcal{L}_{res}$ without changing the overall framework.

5 Impact of Different Amounts of Unlabeled Data

In this section, we conduct experiments to explore the impact of different amounts of unlabeled data on the performance. Here, the baseline is CMKD trained on KITTI *train* with $\sim 3.7k$ samples, and we gradually add unlabeled samples from *Eigen clean* split to the training set. We calculate the mean $3D\ AP$ and $BEV\ AP$ for Car on KITTI *val*.

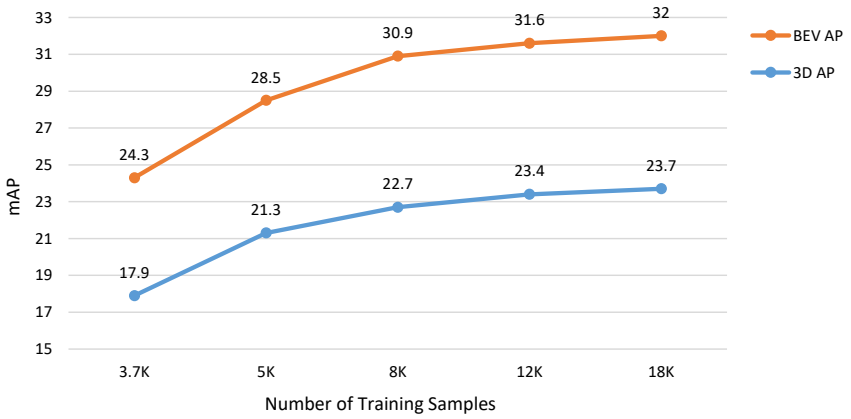

Fig. 3. Impact of different amounts of unlabeled data to the performance. We use CMKD trained on KITTI *train* with $\sim 3.7k$ samples as the baseline and gradually add unlabeled samples from *Eigen clean* split to the training set. We calculate the mean $3D\ AP$ and $BEV\ AP$ for Car on KITTI *val*.

As can be seen from Fig. 3, the performance of CMKD improves as the number of unlabeled samples increases. Specifically, when the training samples are limited (e.g., there are only $\sim 3.7k$ samples on KITTI *train*, which are very few

to well train a deep network like CMKD), a small number of unlabeled samples ($\sim 1.3k$) can bring significant performance gains. When the number of unlabeled samples becomes larger ($+4.3k$, $+8.3k$, $+14.3k$ respectively), the magnitude of the performance improvement tends to moderate. And this is consistent with the trend of performance gains from pre-training with additional unlabeled data in other tasks, e.g., image classification task on ImageNet [2].

Note that, here, the amount of additional unlabeled data and the information it can provide is not linear. As mentioned before, KITTI 3D is a sub-set of KITTI Raw and KITTI Raw is in continuous sequence form, so there are a large number of similar, repeated samples which can only provide limited new information. Moreover, KITTI Raw is a massive unlabeled dataset which also contains many low-quality samples, e.g., with only repetitive and noisy background information, and these low-quality samples may in turn degrade the performance of the model. However, one of our starting points of this work is that **end-to-end training can be performed directly on massive unlabeled data** to greatly reduce the cost of annotation and other pre-processing steps. Therefore, we do not filter these unlabeled samples, but directly use all of them for training, which is exactly the motivation of the proposed semi-supervised training method.

References

1. Chen, L., Papandreou, G., Schroff, F., et al.: Rethinking atrous convolution for semantic image segmentation. CoRR **abs/1706.05587** (2017)
2. Deng, J., Dong, W., Socher, R., et al.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR (2009)
3. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: CVPR (2012)
4. He, K., Zhang, X., Ren, S., et al.: Deep residual learning for image recognition. In: CVPR (2016)
5. Howard, A.G., Sandler, M., Chu, G., et al.: Searching for mobilenetv3. ICCV (2019)
6. Liu, J., Hou, Q., Cheng, M., et al.: Improving convolutional networks with self-calibrated convolutions. In: CVPR (2020)
7. Liu, Z., Mao, H., Wu, C.Y., et al.: A convnet for the 2020s. arXiv preprint arXiv:2201.03545 (2022)
8. Reading, C., Harakeh, A., Chae, J., et al.: Categorical depth distribution network for monocular 3d object detection. CVPR (2021)
9. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International conference on machine learning. pp. 6105–6114. PMLR (2019)
10. Yan, Y., Mao, Y., Li, B.: SECOND: sparsely embedded convolutional detection. Sensors (2018)