Appendices

In this supplementary material, we describe the implementation details, the pseudo code for CPP pooling, and the ablation experiments for the auxiliary loss weight with CPP pooling. The additional visualizations of the conventional pooling and CPP pooling are provided. Also, further illustrations of CPP pooling and auxiliary heads in the backbones are shown to supplement the descriptions in the paper. More qualitative results for CPP pooling and SeeThroughNet are included. Finally, the results of reproducibility experiments are demonstrated, and the dataset conversion details are described.

A Implementation details

We utilized MMSegmentation [8] to implement the Class Probability Preserving (CPP) pooling experiments. For the SeeThroughNet implementation, we utilized the codebase in [16].

A.1 CPP pooling experiments

For the CPP pooling experiments, we experimented seven popular semantic segmentation models on Cityscapes [9], Pascal VOC [11], Pascal Context [12], and NYU-Depth-v2 [13] datasets. We initialized the backbones with the pre-trained weights on ImageNet [10]. We used Stochastic Gradient Descent (SGD) optimizer and Cross Entropy (CE) loss for the baseline models, and Kullback-Leibler (KL) divergence for the CPP pooling models. To train our models, we used NVIDIA TESLA A100 8 GPUs with the distributed data parallel (DDP) training and the synchronized batch normalization (SyncBN). For data augmentation, we employed photometric distortion, random horizontal flip, and random scaling (0.5 to 2.0). The hyper-parameters for each dataset are as follows.

Cityscapes: We used polynomial learning rate policy with factor 0.9, 0.01 as initial learning rate, $5e^{-4}$ for weight decay, 80K iterations with batch size of 2 per GPU, and crop size as 1024x512.

Pascal VOC: We used polynomial learning rate policy with factor 0.9, 0.01 as initial learning rate, $5e^{-4}$ for weight decay, 40K iterations with batch size of 2 per GPU, and crop size as 512x512.

Pascal Context: We used polynomial learning rate policy with factor 0.9, 0.004 as initial learning rate, $1e^{-4}$ for weight decay, 40K iterations with batch size of 2 per GPU, and crop size as 480x480.

NYU-V2: We used polynomial learning rate policy with factor 0.9, 0.004 as initial learning rate, $1e^{-4}$ for weight decay, 40K iterations with batch size of 4 per GPU, and crop size as 640x480.

A.2 SeeThroughNet

Training setting. We used Stochastic Gradient Descent (SGD) optimizer, batch size of 1 per GPU, and $5e^{-4}$ for weight decay. We used polynomial learning rate policy with factor 2.0, 0.01 as initial learning rate, and 175 epochs. As in [16], we also followed the class uniform sampling in the data loader. To train the models, we used NVIDIA TESLA A100 8 GPUs, with the distributed data parallel (DDP) training and the synchronized batch normalization (SyncBN). For the network training and inference, we followed the two scale training and the multi-scale inference, respectively, as in [16].

To achieve our state-of-the-arts results, we used the Cityscapes fine set (2,975 images), the auto-labelled coarse set (19,998 images), the validation set (500 images), and the Mapillary [14] train/validation set (20,000 images) converted to the Cityscapes labeling rule. We adopted the uniform sampling following [16] to sample each class equally from the Cityscapes coarse and the Mapillary set. For each batch, we used the dataset proportions as 40% from the fine + val set, 40% from the Mapillary set, and 20% from the coarse set. For example, given the batch size of 2,975 images, we sampled 1,190 (40% of 2,975) images from the Mapillary set (20,000 images).

Data augmentation. We used the crop size as 2048x1024 for Cityscapes. We employed color augmentation, gaussian blur, random horizontal flip, and random scaling (0.5 to 2.0).

B Pseudo code for CPP pooling

Algorithm 1 Class Probability Preserving (CPP) Pooling

Input: Segmentation ground truth map $\mathbf{Y}_{(i,j)}$, down-sampling scale factor **1/s**
Output: Output map of CPP pooling $\mathbf{Y}^{'}_{(l,m)}$
for each pixel (l, m) in $\mathbf{Y}^{'}_{(l,m)}$ **do**
    for all class $k \in K$ **do**
        for (i, j) in $s \times l \leq i < s \times (l + 1)$ and $s \times m \leq j < s \times (m + 1)$ **do**
            Count the # of pixels matching the class k in $\mathbf{Y}_{(i,j)}$ to compute probability of $\mathbf{Y}^{'}_{k_{(l,m)}}$
        end for
        Normalize $\mathbf{Y}^{'}_{k_{(l,m)}}$ by s^2
    end for
end for
return $\mathbf{Y}^{'}_{(l,m)}$

C Ablation experiment - Auxiliary loss weight for CPP pooling

We conducted ablation experiments for the auxiliary loss weight with CPP pooling. Ten models with the auxiliary loss weights between 0 and 1 (step size 0.1) were trained on the Cityscapes fine set and evaluated on the validation set. PSPNet [17] with CPP pooling was used for the experiments. As in Table 1, we chose 0.6 for the CPP experiments in the paper.

Table 1: Ablation study for the CPP Pooling loss weight. PSPNet [17] with CPP Pooling was used. Trained and evaluated on the Cityscapes fine and validation set, respectively.

Weight	0.1	0.2	0.3	0.4	0.5	**0.6**	0.7	0.8	0.9	1.0
mIoU(%)	79.52	80.24	79.65	80.11	80.44	**81.07**	80.09	80.28	80.27	80.37

D Additional Figures

D.1 Additional Illustration of CPP pooling effect

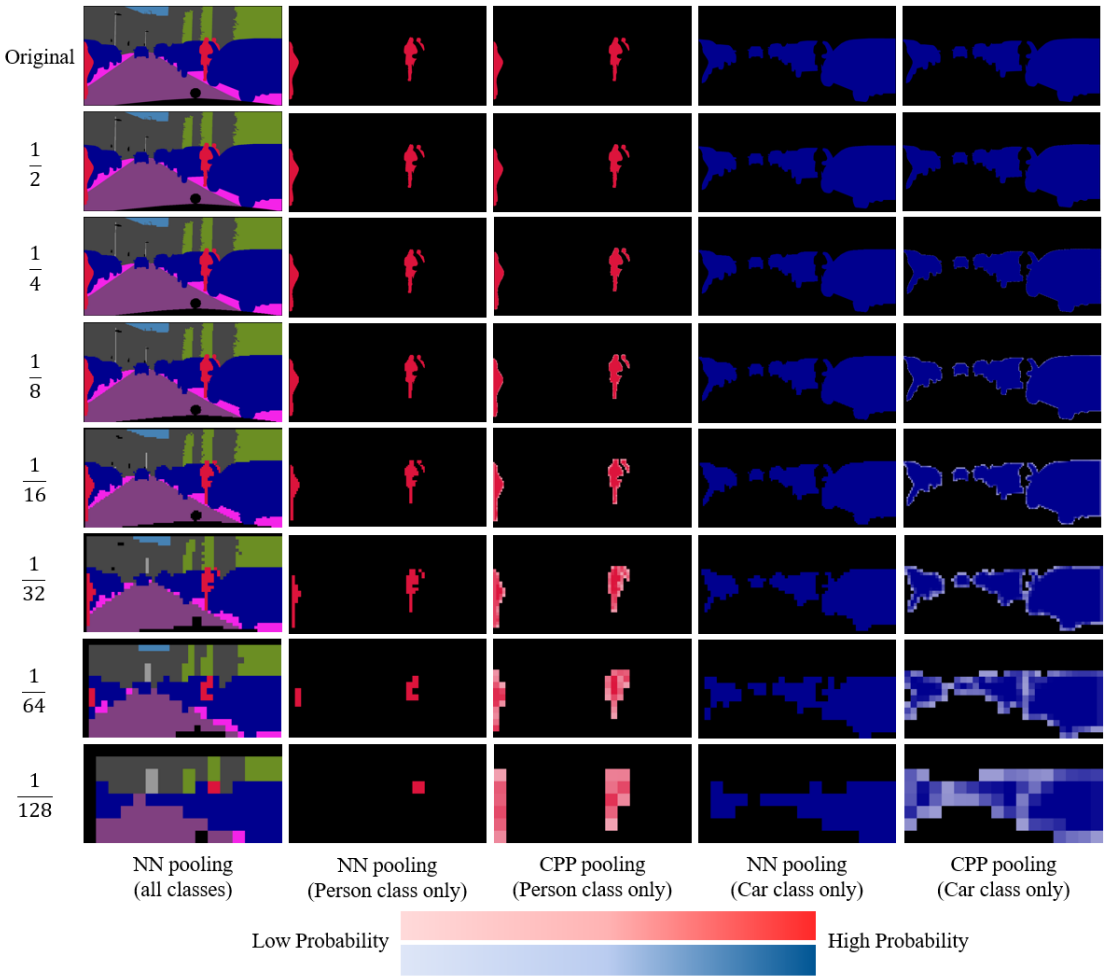
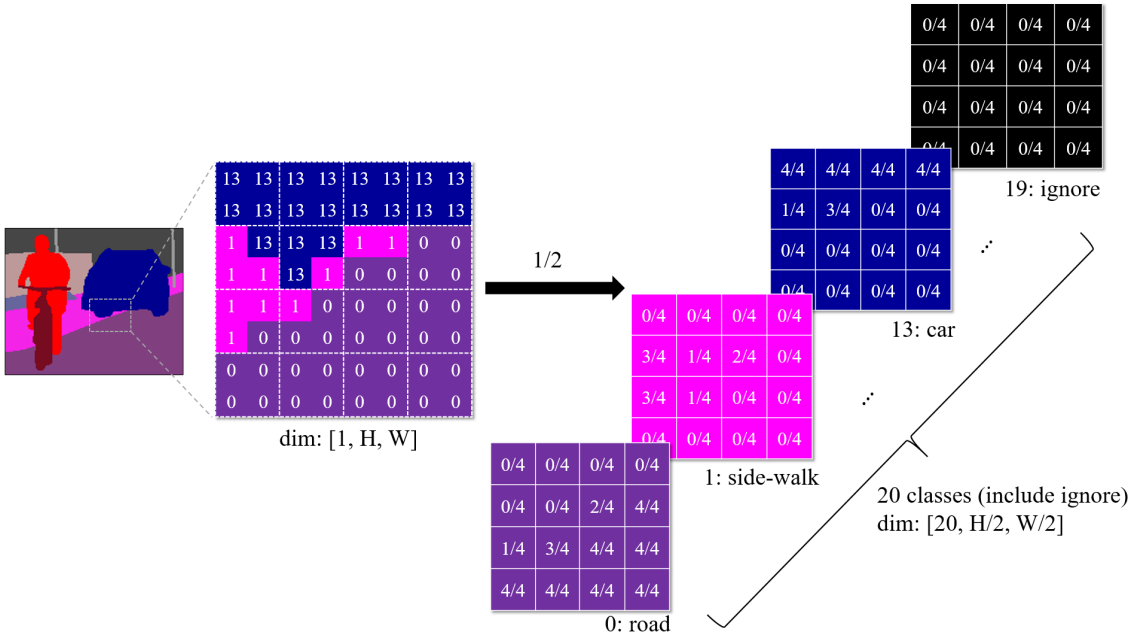
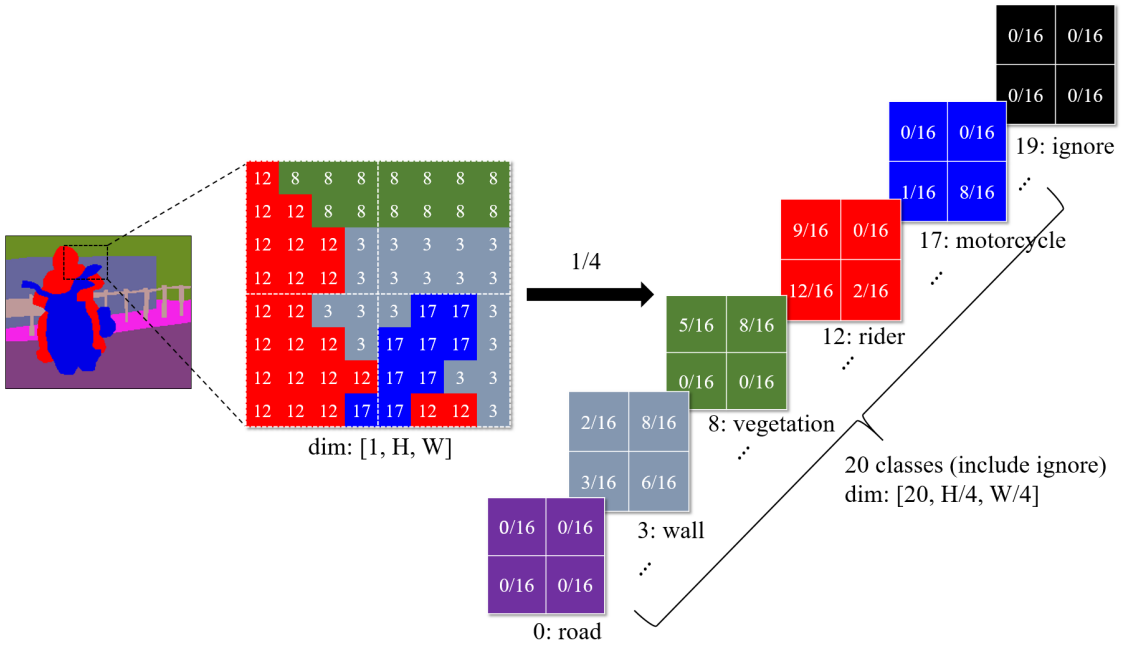


Figure 1: Visualization of the effects of the conventional pooling (i.e. nearest neighbor [NN] pooling) and CPP pooling. NN pooling has been widely used for down-sampling ground truth for segmentation task since the ground truth ids are integer. A semantic ground truth from the Cityscapes dataset is used. As the scale factor of the pooling increases, more information is lost. Especially, in NN pooling, far object and boundary information are lost a lot, while they still remain as a probability in CPP pooling.

D.2 Additional Illustration of Class Probability Preserving (CPP) poolig



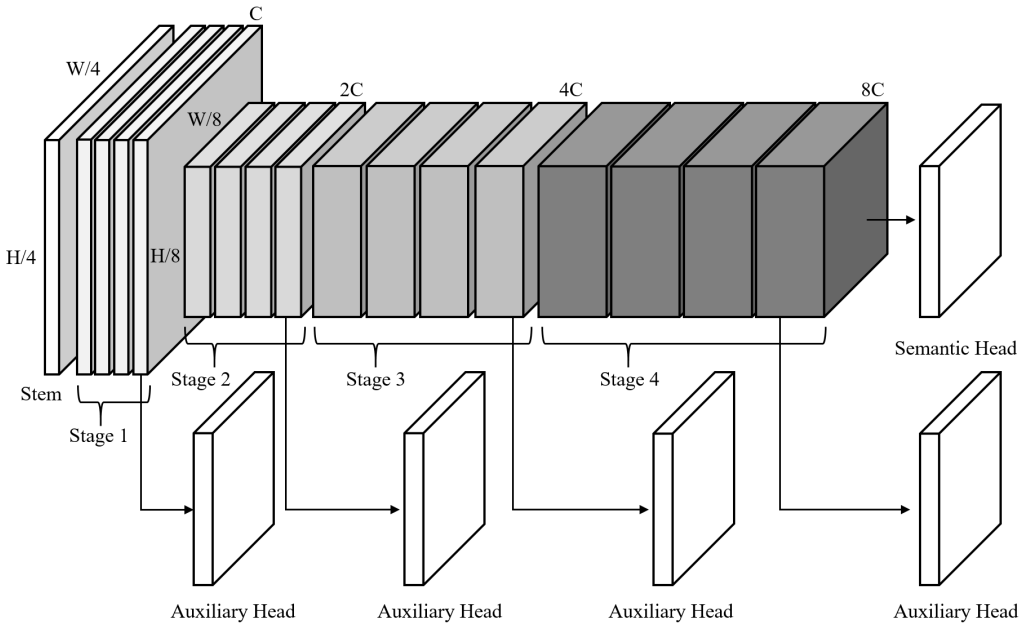
(a) 2×2 Class Probability Preserving Pooling



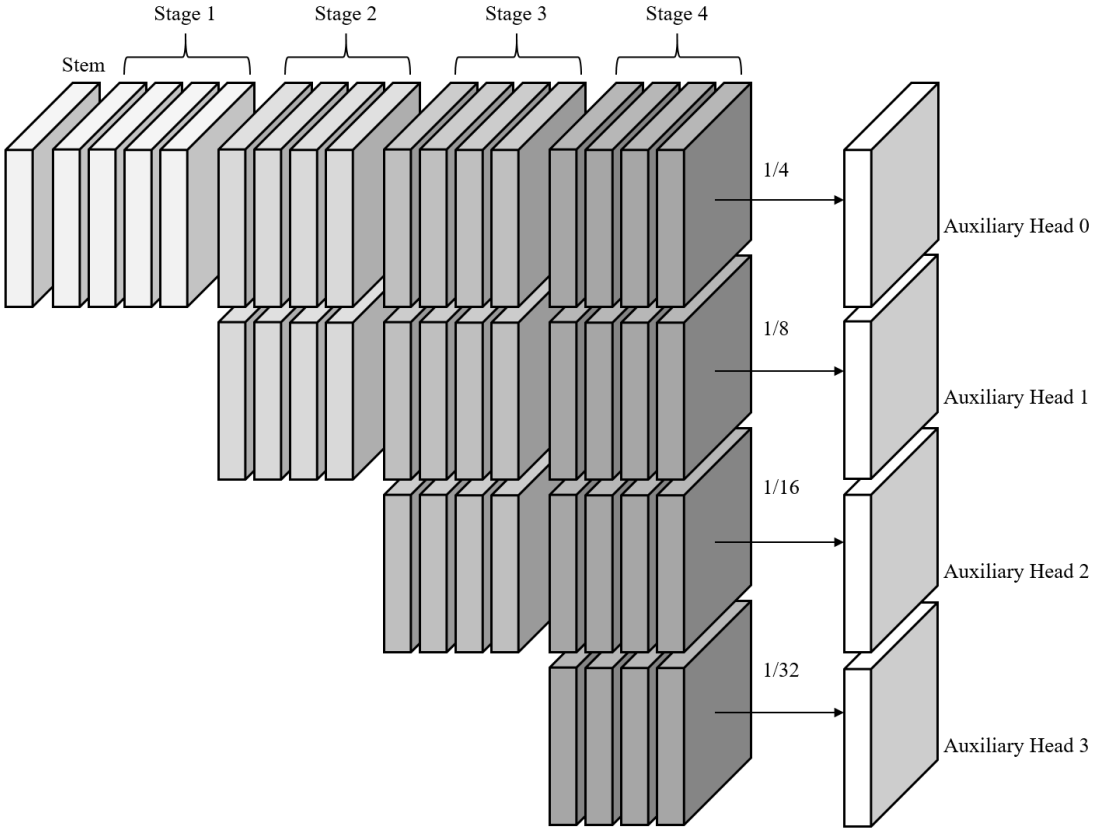
(b) 4×4 Class Probability Preserving Pooling

Figure 2: Illustration of Class Probability Preserving (CPP) pooling. (a) shows 1/2 pooling where the 2x2 grids information is down sampled to one grid. (b) shows 1/4 pooling where the 4x4 grids information is down sampled to one grid. The number of channels of the output feature map equals to the total number of classes (e.g. 20 classes for cityscapes datasets including ignore class), where each channel represents the class probability of the corresponding grid.

D.3 Illustration of Auxiliary Heads



(a) Four auxiliary heads in ResNet101



(b) Four auxiliary heads in HRNetV2-48

Figure 3: Illustration of auxiliary heads for ResNet101 and HRNetV2-48 backbones. The figures supplement the description in Section 4.1 in the paper.

D.4 Qualitative Results - CPP Pooling

D.4.1 PASCAL VOC

Figure 4: Qualitative results of DeepLabv3+ with and without CPP pooling on PASCAL VOC 2012. The models with CPP pooling show better results for the objects with occlusions.

D.4.2 Cityscapes

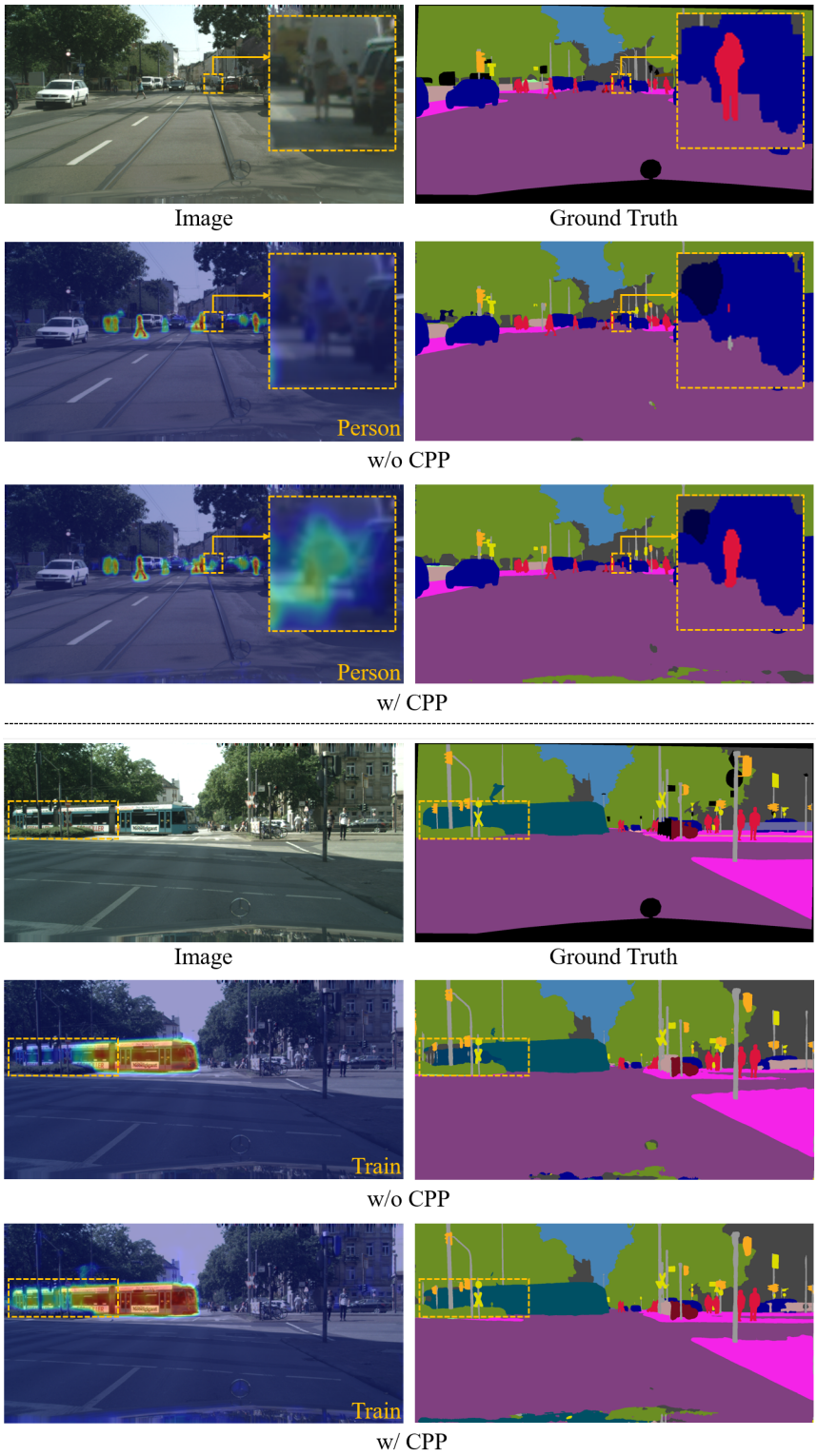


Figure 5: Qualitative results of DeepLabv3+ with and without CPP pooling on the Cityscapes val set. The Grad-CAM [15] visualizations and the inference results are shown. The models with CPP pooling show better results for the distant objects and occlusions. The enlarged views are provided for the upper examples.

D.5 Qualitative Results - SeeThroughNet on Cityscapes

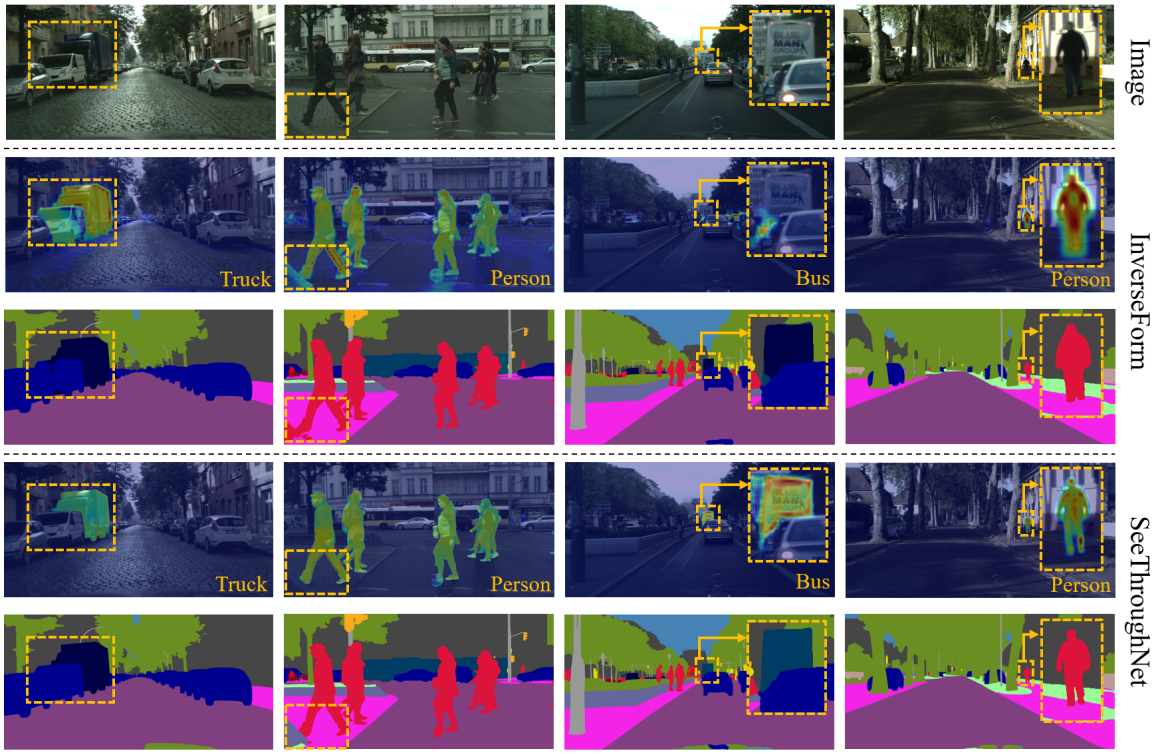


Figure 6: SeeThroughNet qualitative results on the Cityscapes *test* set. The inference results and Grad-CAM [15] visualizations of the SeeThroughNet and another state-of-the-art model InverseForm [6] are compared. The SeeThroughNet shows finer and more accurate results for the occluded and distant objects. The Grad-CAM classes are marked in the right-bottom corner of the Grad-CAM images. The enlarged views are provided for columns 3 and 4.

E Reproducibility Experiments

For the experiments, we followed [1–5] to control reproducibility.

E.1 CPP pooling

To see the reproducibility of CPP pooling, we ran the experiments ten times for each method on the PASCAL VOC 2012 [11] validation set. Note that the results are not completely reproducible even with the fixed random seed in Table 2, as stated in [3].

Table 2: The results of the ten-times experiments for CPP pooling on the PASCAL VOC 2012 validation set.

Method			Min	Max.	Average	Var.	Std.
DeepLabV3+ [7]	w/o CPP pooling	w/ Same Seed (9802)	77.38	77.80	77.58	0.0365	0.1911
		w/ Random Seed	77.19	78.40	77.76	0.2091	0.4573
	w/ CPP pooling	w/ Same Seed (9802)	79.13	79.76	79.37	0.0699	0.2644
		w/ Random Seed	79.21	80.80	79.65	0.5613	0.7492
PSPNet [17]	w/o CPP pooling	w/ Same Seed (9802)	76.26	77.02	76.75	0.0856	0.2926
		w/ Random Seed	76.13	77.53	77.03	0.2822	0.5312
	w/ CPP pooling	w/ Same Seed (9802)	79.23	79.84	79.49	0.0501	0.2238
		w/ Random Seed	78.29	79.96	79.08	0.3612	0.6010

E.2 SeeThroughNet

To check the reproducibility of SeeThroughNet, we trained the model ten times with the Cityscapes train and coarse set, and evaluated on the validation set.

Table 3: The results of the ten-times experiments for SeeThroughNet on Cityscapes

	Min.	Max.	Average	Var.	Std.
w/ Random Seed	86.98	87.23	87.08	0.007	0.088

F Mapillary to Cityscapes conversion details.

We converted the Mapillary [14] training and validation set to the Cityscapes [9] labels as the following mapping rule.

Table 4: Mapping rule - Mapillary to Cityscapes

Id	Cityscapes	Mapillary
0	Road	Road, Bike Lane, Crosswalk plain, Rail Track, Service Lane, Lane Marking Crosswalk, Lane Marking General
1	Sidewalk	Sidewalk, Curb, Curb cut
2	Building	Building
3	Wall	Wall
4	Fence	Fence, Guard Rail, Barrier
5	Pole	Pole, Utility Pole, Traffic Sign Frame
6	Traffic light	Traffic Light
7	Traffic sign	Traffic Sign Front
8	Vegetation	Vegetation
9	Terrain	Terrain
10	Sky	Sky
11	Person	Person
12	Rider	Bicyclist, Motorcyclist, Other Rider
13	Car	Car Other Vehicle
14	Truck	Truck
15	Bus	Bus
16	Train	N/A
17	Motorcycle	Motorcycle
18	Bicycle	Bicycle

References

[1] CublasAPI Reproducibility, howpublished = https://docs.nvidia.com/cuda/cublas/index.html#cublasapi_reproducibility.

[2] NumPy Random Generator, howpublished = https://numpy.org/doc/stable/reference/random/generator.html#numpy.random.generator.

[3] Pytorch Document - Reproducibility, howpublished = https://pytorch.org/docs/stable/notes/randomness.html.

[4] Randomness in multi-process data loading, howpublished = https://pytorch.org/docs/stable/data.html#randomness-in-multi-process-data-loading.

[5] Torch Use Deterministic Algorithms, howpublished = https://pytorch.org/docs/stable/generated/torch.use_deterministic_algorithms.html#torch.use_deterministic_algorithms.

[6] Borse, S., Y. Wang, Y. Zhang, and F. Porikli (2021). Inverseform: A loss function for structured boundary-aware segmentation. In *IEEE CVPR*, pp. 5901–5911.

[7] Chen, L.-C., Y. Zhu, G. Papandreou, F. Schroff, and H. Adam (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 801–818.

[8] Contributors, M. (2020). MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. https://github.com/openc-mmlab/mmsegmentation.

[9] Cordts, M., M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele (2016). The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[10] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee.

[11] Everingham, M., L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[12] Mottaghi, R., X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille (2014). The role of context for object detection and semantic segmentation in the wild. In *IEEE CVPR*.

[13] Nathan Silberman, Derek Hoiem, P. K. and R. Fergus (2012). Indoor segmentation and support inference from rgbd images. In *ECCV*.

[14] Neuhold, G., T. Ollmann, S. Rota Bulo, and P. Kontschieder (2017). The mapillary vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4990–4999.

[15] Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE ICCV*, pp. 618–626.

[16] Tao, A., K. Sapra, and B. Catanzaro (2020). Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*.

[17] Zhao, H., J. Shi, X. Qi, X. Wang, and J. Jia (2017). Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2881–2890.