Supplementary for Dual Adaptive Transformations for Weakly Supervised Point Cloud Segmentation

Zhonghua Wu[1,2], Yicheng Wu[3], Guosheng Lin[⋆1,2], Jianfei Cai[2,3], and Chen Qian[4]

[1] S-lab, Nanyang Technological University
[2] School of Computer Science and Engineering, Nanyang Technological University
[3] Department of Data Science and AI, Monash University
[4] SenseTime Research
zhonghua001@e.ntu.edu.sg

A More Ablation Studies

Different hyper-parameters α and β. We conduct experiments with different α and β under the OTOC setting on the S3DIS dataset. From Table A, we can see that setting both α and β as 2 yields the best performance for the weakly supervised point cloud segmentation task.

Table A. Ablation studies on different α and β under the OTOC setting on the S3DIS dataset.

α	β	mIoU(%)
2	1	55.6
2	2	**56.5**
2	5	56.2

α	β	mIoU(%)
1	2	55.1
2	2	**56.5**
5	2	56.1

Different Superpoint in RAD. We conduct experiments to analyze the sensitivity of superpoints, which are used in RAD. Specifically, we generate the superpoints with different hyperparameters of the number of neighbors for the geometric features *geo* and adjacency graph *adj*. As shown in Table B, we can see that our RAD module is not very sensitive to different superpoints.

B Pseudocodes

Algorithm A and B respectively provide the pseudocodes of our designed LAP and RAD modules.

⋆ Corresponding author: G. Lin (e-mail: gslin@ntu.edu.sg)

Table B. Ablation studies on different *geof* and *adj* under the OTOC setting on the S3DIS dataset.

geof	*adj*	mIoU(%)
30	10	54.4
45	5	54.7
45	10	**56.5**

C More Qualitative Results

Figure A and B respectively show five more segmentation results obtained by our proposed model on the S3DIS and ScanNet-v2 dataset. Our DAT model is able to generate accurate segmentation masks for most of the points only with the weak annotation training.

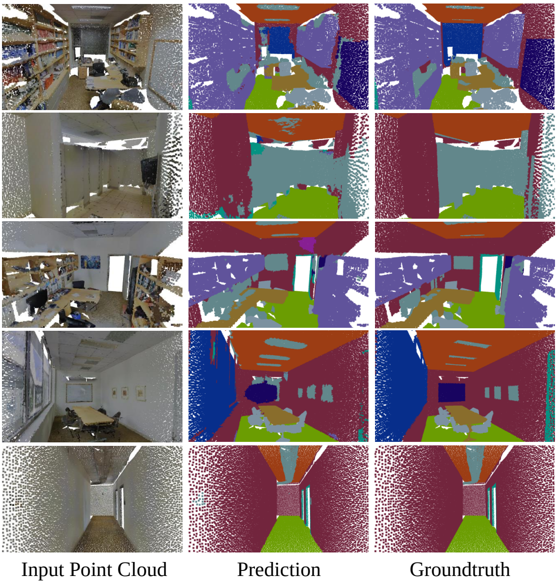

Fig. A. Visualization results obtained by our DAT model on the S3DIS dataset under the "OTOC" setting.

Algorithm A Pseudocode of Local Adaptive Perturbation (LAP) module in a PyTorch-like style.

```
# CPG: class-aware perturbation generator
# c: point coordinates
# f: point features
# xi_c: theta for coordinates
# xi_f: theta for features
# eps_c: epsilon for coordinates
# eps_f: epsilon for features
# ip: iteration times of computing adv noise (default: 1)

import torch.nn.functional as F

def LAP(c, f, model):

    pred = model(c, f)

    pseudo_label = pred.max(dim=1)[1] # generate pseudo label
    CPG.update_CV(c, f, pseudo_label) # update the covariance matrices

    c_init, f_init = CPG.generator(c, f, pseudo_label) # generate initial unit vectors for
         coordinates and features

    # normalize the initial unit vectors
    c_init = _l2_normalize(c_init)
    f_init = _l2_normalize(f_init)

    for _ in range(ip):
        c_init.require_grad()
        f_init.require_grad()

        pred_init = model(c + xi_c * c_init, f + xi_f * f_init)
        adv_distance = F.kl_div(F.log_softmax(pred_init, dim=1), pred)

        # generate adversarial perturbations
        adv_distance.backward()

        c_adv = _l2_normalize(c_init.grad)
        f_adv = _l2_normalize(f_init.grad)

        model.zero_grad()

    pred_hat = model(c + eps_c * c_adv, f + eps_f * f_adv)

    Loss_pc = F.kl_div(F.log_softmax(pred_hat, dim=1), pred) # point-level consistency loss
```

Algorithm B Pseudocode of Regional Adaptive Deformation (RAD) module in a PyTorch-like style.

```
# c: point coordinates
# f: point features
# xi: theta
# eps: epsilon
# ip: iteration times of computing adv noise (default: 1)

import torch.nn.functional as F

def RAD(c, f, model):

    pred = model(c, f)

    S = SP_G(c, f) # offline superpoint extraction

    A_init = A_generator(c, S) # generate initial affine transformation matrices
    A_init = normalize(A_init) # normalize the matrices

    for _ in range(ip):
        A_init.require_grad()

        c_init = affine(c, S, A_init) # generate initial perturbed point cloud

        pred_init = model(c_init, f)

        adv_distance = F.kl_div(F.log_softmax(pred_init, dim=1), pred)

        # generate region-level adversarial examples
        adv_distance.backward()
        A_adv = normalize(A_init.grad)
        model.zero_grad()

    c_adv = affine(c, S, eps * A_adv)
    pred_hat = model(c_adv, f)

    Loss_rc = F.kl_div(F.log_softmax(pred_hat, dim=1), pred) # region-level consistency loss
```

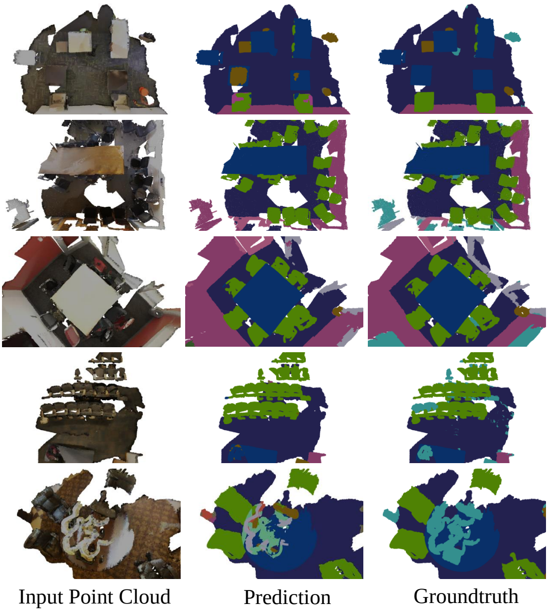

Fig. B. Visualization results obtained by our DAT model on the ScanNet-v2 dataset under the “20 points” setting.