Unsupervised 3D Point Cloud Representation Learning by Triangle Constrained Contrasting for Autonomous Driving

Bo Pang* Hongchi Xia* Cewu Lu†
Shanghai Jiao Tong University
{pangbo, xiahongchi, lucewu}@sjtu.edu.cn

A. Algorithm

Here, we provide the pseudo-code of the proposed TriCC in Algo. 1. mm and sm denote the matrix multiplication and softmax functions respectively. We do not mention the super-pixel process (see Appendix. E) in the algorithm for concise and clarity.

B. Details of Downstream Tasks' Training

B.1. Fine-tuning Details of Semantic Segmentation

B.1.1 nuScenes

We use pre-trained MinkUNet [8] and VoxelNet [20] as backbones to perform semantic segmentation on nuScenes [4]. The pre-training details are presented in Sec. 4. A linear classification head is added to the end of the backbone to build the segmentation model. For downstream fine-tuning, we use SGD as our optimizer with a batch size of 16, momentum of 0.9, dampening of 0.1, and a cosine learning rate scheduler. The backbone and head are trained with different learning rates for better transfer performance. The learning rate for backbone is selected from [0.005, 0.01, 0.02, 0.04] and the learning rate for classification head is 100 times greater than the former. The weight decay is chosen from [0.001, 0.0005, 10^{-4}, 10^{-5}]. We use the combination of the cross-entropy and lovász [3] as the loss function. Augmentations composed of rotation and flipping axis are also performed on point clouds like pre-training. The batchnorm momentum is set to 0.02. The voxel size and point cloud range are the same as the pre-training settings.

B.1.2 SemanticKITTI

We also use the pre-trained MinkUNet and VoxelNet as backbones to perform semantic segmentation on SemanticKITTI [2]. The pre-training details are the same as nuScenes and the fine-tuning details are almost identical to nuScenes except we use a different setting of the classification head's learning rate. Here, the learning rate for classification head is 40 times greater than the backbone.

B.2. Fine-tuning Details of 3D Object Detection

B.2.1 KITTI

We use pre-trained MinkUNet and VoxelNet as backbones to perform 3D object detection on KITTI [9]. We adopt the well-known OpenPCDet [1] codebase and follow its default model settings. For PointRCNN [15], we replace its 3d backbone to MinkUNet and set other parameters by default. The learning rate is selected from [0.0025, 0.005, 0.01] and weight decay is chosen from [0.01, 0.03]. For VoxelNet, we utilize it as the backbone of PV-RCNN [14] and SECOND [17] detection algorithms and follow the default setting in OpenPCDet. In PV-RCNN, the learning rate is selected from [0.0025, 0.005, 0.01] and weight decay is chosen from [0.002, 0.01, 0.05]. In SECOND, the learning rate is selected from [0.0015, 0.003, 0.006] and weight decay is chosen from [0.002, 0.01, 0.05].

B.2.2 nuScenes

On nuScenes, we use VoxelNet as the backbone to perform 3D object detection. We follow OpenPCDet's model settings except learning rate, weight decay, and training epoch. We adopt CenterPoint [19] with voxel size of 0.1 meters and SECOND as our detection algorithms. In CenterPoint, the learning rate is selected from [0.0015, 0.005, 0.003] and weight decay is chosen from [0.002, 0.01, 0.05]. In SECOND, the learning rate is selected from [0.0015, 0.003, 0.006] and weight decay is chosen from [0.002, 0.01, 0.05]. All the models are fine-tuned for 30 epochs.

C. Augmentation Details

Augmentation is important for discriminative unsupervised methods to get diverse pairs and learn effective representations. We adopt two groups of augmentation for point clouds and images respectively following SLidR [13].

*Equal contribution. † Cewu Lu is the corresponding author.

[1] https://github.com/open-mmlab/OpenPCDet

Algorithm 1 Pseudocode of TriCC in a PyTorch-like style.

```
# fp, fc: Backbone networks for point clouds and
    camera images
# gc, hc, gp, hp: Projection heads for consistent
    contraint and contrast
# aug_p, aug_c: Augmentations for point clouds and
    images
# t: Temperature

for x_p1, x_p2, x_c, cr in loader: # get a minibatch

  # random augmentation
  x_p1 = aug_p(x_p1) # for point cloud in time t
  x_p2 = aug_p(x_p2) # for point cloud in time t+1
  x_c = aug_c(x_c) # for image in time t

  # forward to get features for constraint
  Pg1, Pg2 = gp(fp(x_p1)), gp(fp(x_p2))
  Cg = gc(fc(x_c).detach())
  # forward to get features for contrast
  Ph1, Ph2 = hp(fp(x_p1)), hp(fp(x_p2))
  Ch = hc(fc(x_c).detach())

  # norm
  Pg1, Pg2, Cg = norm(Pg1), norm(Pg2), norm(Cg)
  Ph1, Ph2, Ch = norm(Ph1), norm(Ph2), norm(Ch)

  # get constraint losses
  ls = constraint((Pg1, Pg2, Cg), cr)
  l_shortcut = constraint((Pg1, Pg2))

  # get contrast loss
  lc = contrast(Ph1, Ph2, Ch, Pg1, Pg2, Cg, cr))

  loss = ls + l_shortcut +lc

  # SGD update
  loss.backward()
  update(fp, fc, gc, hc, gp, hp)


def constraint(feat_g, calib_relation=None):
  # get transition matrices
  m = []
  n_f = len(feat_g)
  for i in range(n_f):
    m.append(sm(mm(feat_g[i], feat_g[(i+1)%n_f])/t))
  if calib_relation != None:
    m[-1] = calib_relation

  # get self-cycle
  s = m[0]
  for i in range(1, len(m)):
    s = mm(s, m[i])

  loss = crossEntropy(log(s), I)
  return loss


def contrast(Ph1, Ph2, Ch, Pg1, Pg2, Cg, calib_rel=
    None):
  # get matching relations
  m_cp2 = mm(Cg, Pg2.T)
  m_pp = mm(Pg2, Pg1.T)
  m_p1c = calib_rel if calib_rel else mm(Pg1, Cg.T)

  # contrast
  l1 = crossEntropy(mm(Ch, Ph2.T)/t, argmax(m_cp2))
  l2 = crossEntropy(mm(Ph2, Ph1.T)/t, argmax(m_pp))
  l3 = crossEntropy(mm(Ph1, Ch.T)/t, argmax(m_p1c))

  return (l1 + l2 + l3)/3
```

For point clouds, we adopt three augmentation methods: random rotation, random flip, and random cuboid drop. The rotation is applied on the z-axis with a random angle. The flip is applied on the x and y-axis with 50% probability respectively. For cuboid dropping, the center of the dropped cuboid is randomly selected and the scale on every axis is larger than 10% of the corresponding point cloud scale to cover at least 1024 pairs.

For images, we adopt two augmentation methods: random horizontal flip and random crop-resizing. The random probability for the former is 50%. All the crops are resized to 416×224. The aspect ratio of crops is between 14/9 and 17/9, and the scale of crops is larger than 30% of the original image area.

D. More Results

Here, we provide detailed comparisons of segmentation and detection performance on every single category with our baselines and provide more ablation studies.

D.0.1 Detailed Results of nuScenes 1% Segmentation Fine-tuning

We report the detailed fine-tuning results on nuScenes semantic segmentation with 1% annotations in Tab. 1. We can see that the proposed TriCC achieves best performances on most categories.

D.0.2 Detailed Results on KITTI 100% 3D Detection

We report the detailed fine-tuning results on KITTI object detection with 100% annotations in Tab. 2. We can see that the proposed TriCC achieves great performances in all three categories, comparable with all the top values.

D.0.3 Ablation Study on Cycle Shortcut

We add one cycle shortcut between $\mathbf{P}_t$ and $\mathbf{P}_{t+1}$. We do not add more cycle shortcuts between ($\mathbf{C}_t$, $\mathbf{P}_{t+1}$) and ($\mathbf{C}_t$, $\mathbf{P}_t$) because they cannot lead to better performance in experiments as shown in Tab. 3. One shortcut is enough for boosting the efficiency.

E. Details to Adopt Super-Pixel

For triplet contrast, we follow the SLidR [13] to contrast in the unit of super-pixels instead of single pixels. This is an effective trick to cluster similar pixels and reduce false negatives in contrast. The core idea is to merge pixels (points) into super-pixels (points) and contrast them instead of the original ones. For TriCC, instead of adopting the matching relationships to find pixel (point)-level contrast pairs, we adopt matching relationships to find matching super-pixels (points) and contrast them. The detailed pipeline is:

- Get super-pixels of input images with SLIC [1] algorithm. For each image, we get at most K super-pixels and each super-pixel $\mathcal{S}_i$ is a set of original pixels in the image. In SLIC algorithm, an image can be split into less than K super-pixels. Here, for concise, we assume

Table 1. Detailed results of nuScenes 1% semantic segmentation fine-tuning.

Method	barrier	bicycle	bus	car	const. veh.	motorcycle	pedestrian	traffic cone	trailer	truck	driv. surf.	other flat	sidewalk	terrain	manmade	vegetation	**mIoU**
Random	0.0	0.0	8.1	65.0	0.1	6.6	21.0	9.0	9.3	25.8	89.5	14.8	41.7	48.7	72.4	73.3	30.3
PointContrast	0.0	1.0	5.6	67.4	0.0	3.3	31.6	5.6	12.1	30.8	91.7	21.9	48.4	50.8	75.0	74.6	32.5
DepthContrast	0.0	0.6	6.5	64.7	0.2	5.1	29.0	9.5	12.1	29.9	90.3	17.8	44.4	49.5	73.5	74.0	31.7
PPKT	0.0	2.2	**20.7**	**75.4**	**1.2**	13.2	45.6	8.5	**17.5**	**38.4**	92.5	19.2	52.3	56.8	80.1	80.9	37.8
SLidR	0.0	3.1	15.2	72.0	0.9	18.8	43.2	12.5	14.7	33.3	92.8	29.4	54.0	61.0	80.2	81.9	38.3
TriCC(ours)	0.0	**2.6**	**20.7**	73.6	0.3	**18.9**	**49.2**	**22.0**	16.9	33.4	**94.5**	**43.1**	**57.2**	**62.1**	**82.3**	**82.6**	**41.2**

Table 2. Comparisons with SOTA unsupervised 3D representation learning methods on KITTI 3D object detection fine-tuning with 100% annotations. We report AP@R11 and mAP@R11 for SECOND models. AP@R40 and mAP@R40 are reported for PV-RCNN models. For category-wise performances, we report the moderate-level results.

Pretrain	Detection Model	Vehicle	Pedestrian	Cyclist	Fine-tuning		
					Easy	Moderate	Hard
AP@R11 & mAP@R11 w/o road planes							
Train from scratch	SECOND	77.5	48.7	63.3	73.3	63.2	60.3
SwAV [6]	SECOND	77.6	49.5	65.0	73.2 (-0.1)	64.0 (+0.8)	60.9 (+0.6)
DeepCluster [5]	SECOND	77.5	49.5	63.2	73.2 (-0.1)	63.4 (+0.2)	60.1 (-0.2)
BYOL [10]	SECOND	76.9	43.3	61.0	71.1 (-2.2)	60.4 (-2.8)	57.0 (-3.3)
Point Contrast [16]	SECOND	77.5	45.3	65.4	72.7 (-0.6)	62.7 (-0.5)	59.2 (-1.1)
GCC-3D [12]	SECOND	78.0	47.9	64.5	73.9 (+0.6)	63.5 (+0.3)	59.8 (-0.5)
STRL [11]	SECOND	77.6	48.5	65.5	74.0 (+0.7)	63.9 (+0.7)	60.9 (+0.6)
SLidR [13]	SECOND	**78.2**	49.9	**65.8**	73.6 (+0.3)	64.6 (+1.4)	61.5 (+1.2)
CO^3 [7]	SECOND	78.0	49.6	65.6	74.4 (+1.1)	64.4 (+1.2)	60.9 (+0.6)
TriCC (ours)	SECOND	77.9	**53.8**	65.5	**75.0** (+1.7)	**65.7**(+2.5)	**62.2** (+1.9)
AP@R40 & mAP@R40 with road planes							
Train from scratch	PV-RCNN	84.5	57.1	70.1	81.3	70.6	66.1
Point Contrast [16]	PV-RCNN	84.2	57.7	72.7	82.8 (+1.5)	71.6 (+1.0)	67.5 (+1.4)
GCC-3D [12]	PV-RCNN	-	-	-	-	71.3 (+0.7)	-
STRL [11]	PV-RCNN	84.7	57.8	71.9	-	71.5 (+0.9)	-
SLidR [13]	PV-RCNN	84.3	58.3	71.4	82.9 (+1.6)	71.9 (+1.3)	68.0 (+1.9)
ProposalContrast [18]	PV-RCNN	84.7	**60.4**	73.7	**84.5** (+3.2)	72.9 (+2.3)	69.0 (+2.9)
TriCC (ours)	PV-RCNN	**84.9**	60.1	**74.8**	84.1 (+2.8)	**73.3** (+2.7)	**69.4** (+3.3)

all the images can get K super-pixels. In practice, we just simply remove the empty ones. K is set to 150 in all of our experiments.

- With the super-pixel splitting sets, we can split and merge the image feature-map $\mathbf{C}_t \in \mathbb{R}^{n_{\mathbf{C}_t},c}$ containing features of each pixel into features of each super-pixel:

$$\hat{\mathbf{C}}_t^q = \frac{1}{|\mathcal{S}_q|} \sum_{\mathbf{C}_t^i \in \mathcal{S}_q} \mathbf{C}_t^i \tag{1}$$

where $\hat{\mathbf{C}}_t^q$ is the feature of the qth super-pixel which is the mean value of all the pixels belonging to it.

- Since we get the matching relationships $\hat{\mathbf{m}}_{\mathbf{C}_t,\mathbf{P}_t}$ and $\mathbf{m}_{\mathbf{P}_t,\mathbf{P}_{t+1}}$ between image pixels and LiDAR points through the calibrated relationship and our consistent constraint, we can also get the super-points with:

$$\begin{aligned} \hat{\mathbf{P}}_t^q &= \frac{1}{n_q} \sum_{\mathbf{C}_t^i \in \mathcal{S}_q} \sum_{\hat{\mathbf{m}}_{\mathbf{C}_t,\mathbf{P}_t}^{i,j}=1} \mathbf{P}_t^j \\ \hat{\mathbf{P}}_{t+1}^q &= \frac{1}{n_q} \sum_{\mathbf{C}_t^i \in \mathcal{S}_q} \sum_{\hat{\mathbf{m}}_{\mathbf{C}_t,\mathbf{P}_t}^{i,j}=1} \mathbf{P}_t^{\sigma(\mathbf{m}_{\mathbf{P}_t,\mathbf{P}_{t+1}}^j)} \end{aligned} \tag{2}$$

Table 3. Ablations on more cycle shortcuts. We pre-train the backbones for 20 epochs and compare the 1% nuScenes segmentation fine-tuning performance (mIoU).

shortcut between	mIoU
train from scratch	30.3
no shortcut	39.7
$(\mathbf{P}_{t+1}, \mathbf{P}_t)$	40.8
$(\mathbf{P}_{t+1}, \mathbf{P}_t)$ & $(\mathbf{C}_t, \mathbf{P}_{t+1})$	40.8
$(\mathbf{P}_{t+1}, \mathbf{P}_t)$ & $(\mathbf{C}_t, \mathbf{P}_t)$	40.5

where $\hat{\mathbf{m}}_{\mathbf{C}_t,\mathbf{P}_t}$ is the transposition of $\hat{\mathbf{m}}_{\mathbf{P}_t,\mathbf{C}_t}$, and σ is the argmax function. n_q is the number of points belonging to the qth super-point. Since $\hat{\mathbf{m}}_{\mathbf{C}_t,\mathbf{P}_t}$ may be a one-to-many mapping relationship (one pixel is mapped to many points), we adopt the inner summary function and, thus, $n_q \geq |\mathcal{S}_q|$.

- With $\hat{\mathbf{C}}_t \in \mathbb{R}^{K,c}$, $\hat{\mathbf{P}}_t \in \mathbb{R}^{K,c}$, and $\hat{\mathbf{P}}_{t+1} \in \mathbb{R}^{K,c}$, we just replace $\mathbf{C}_t \in \mathbb{R}^{n_{\mathbf{C}_t},c}$, $\mathbf{P}_t \in \mathbb{R}^{n_{\mathbf{P}_t},c}$, and $\mathbf{P}_{t+1} \in \mathbb{R}^{n_{\mathbf{P}_{t+1}},c}$ in the L_c with them to build the super-pixel (point) version of triple contrast:

$$L_c^{\hat{\mathbf{C}}_t,\hat{\mathbf{P}}_{t+1}} = \frac{1}{K}\sum_q -\log \frac{\exp(\mathrm{sim}(\hat{\mathbf{C}}_t^q, \hat{\mathbf{P}}_{t+1}^q)/\tau)}{\sum_j \exp(\mathrm{sim}(\hat{\mathbf{C}}_t^q, \hat{\mathbf{P}}_{t+1}^j)/\tau)} \tag{3}$$

F. Mentionable Misc Notes

Readers may notice that the transition matrix between $\mathbf{P}_t$ and $\mathbf{P}_{t+1}$ can be pretty large, making the calculation of the final loss costly. In our implementation, we randomly sample x (a common value is 4096) points from $\mathbf{P}_t$. Then all the transition matrices are in the appropriate size. This compromise may affect the performance since it reduces the transition path but it is a common trick adopted in many methods.

One limitation of our TriCC for auto-driving scenes is that we need 360° images provided for aligning with the 360° LiDAR. This is commonly available for most auto-driving scenarios since most auto-driving car contains multiple cameras around the car.

G. Code Implementation

We utilize the official nuScenes-devkit [2] to conduct the pre-training and segmentation. For 3D object detection, OpenPCDet is adopted and we follow its settings of the detection models.

[2] https://github.com/nutonomy/nuscenes-devkit

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012. 2

[2] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *ICCV*, pages 9297–9307, 2019. 1

[3] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *CVPR*, pages 4413–4421, 2018. 1

[4] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, pages 11621–11631, 2020. 1

[5] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, pages 132–149, 2018. 3

[6] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *NeurIPS*, 33:9912–9924, 2020. 3

[7] Runjian Chen, Yao Mu, Runsen Xu, Wenqi Shao, Chenhan Jiang, Hang Xu, Zhenguo Li, and Ping Luo. Coˆ3: Cooperative unsupervised 3d representation learning for autonomous driving. *arXiv preprint arXiv:2206.04028*, 2022. 3

[8] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, pages 3075–3084, 2019. 1

[9] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 1

[10] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *NeurIPS*, 33:21271–21284, 2020. 3

[11] Siyuan Huang, Yichen Xie, Song-Chun Zhu, and Yixin Zhu. Spatio-temporal self-supervised representation learning for 3d point clouds. In *ICCV*, pages 6535–6545, 2021. 3

[12] Hanxue Liang, Chenhan Jiang, Dapeng Feng, Xin Chen, Hang Xu, Xiaodan Liang, Wei Zhang, Zhenguo Li, and Luc Van Gool. Exploring geometry-aware contrast and clustering harmonization for self-supervised 3d object detection. In *ICCV*, pages 3293–3302, 2021. 3

[13] Corentin Sautier, Gilles Puy, Spyros Gidaris, Alexandre Boulch, Andrei Bursuc, and Renaud Marlet. Image-to-lidar self-supervised distillation for autonomous driving data. In *CVPR*, pages 9891–9901, 2022. 1, 2, 3

[14] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-

voxel feature set abstraction for 3d object detection. In *CVPR*, pages 10529–10538, 2020. 1

[15] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR*, pages 770–779, 2019. 1

[16] Saining Xie, Jiatao Gu, Demi Guo, Charles R Qi, Leonidas Guibas, and Or Litany. Pointcontrast: Unsupervised pretraining for 3d point cloud understanding. In *ECCV*, pages 574–591. Springer, 2020. 3

[17] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 1

[18] Junbo Yin, Dingfu Zhou, Liangjun Zhang, Jin Fang, Cheng-Zhong Xu, Jianbing Shen, and Wenguan Wang. Proposalcontrast: Unsupervised pre-training for lidar-based 3d object detection. *ECCV*, 2022. 3

[19] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *CVPR*, pages 11784–11793, 2021. 1

[20] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, pages 4490–4499, 2018. 1