Self-Supervised Pretraining of 3D Features on *any* Point-Cloud
(Supplemental Material)

We provide the model architecture details in § 1. Hyper-parameters used in training and fine-tuning for PointNet++ and UNet and additional results are shown in § 2. We also show hyper-parameters used in training and fine-tuning for PointnetMSG and Spconv-UNet and results for other categories of detection in KITTI in § 3.

1. Architecture Details

PointNet++ model used in Section 4, 5.1 and 5.2. As shown in Table 1, PointNet++ contains four set abstraction layers and two feature up-sampling layers, designed in [6]. Each SA layer is specified by $(n, r, [c_1, ..., c_k])$, where n represents number of output points, r represents the ball-region radius of the reception field, c_i represents the feature channel size of the i-th layer in the MLP. Each feature up-sampling (FP) layer upsamples the point features by interpolating the features on input points to output points, as designed in [7]. Each FP layer is specified by $[c_1, ..., c_k]$ where c_i is the output of the i-th layer in the MLP. In Section 4.1.1 of the main paper, we create higher capacity versions of the PointNet++ model by increasing the channel width. For PointNet++ $2\times$, $3\times$ and $4\times$, we multiply the feature size by $\{2, 3, 4\}$ of each layer in the MLP of the four set abstraction layers. When applying PointNet++ in VoteNet and H3DNet, we adjust the rest of the model accordingly based on different point feature size.

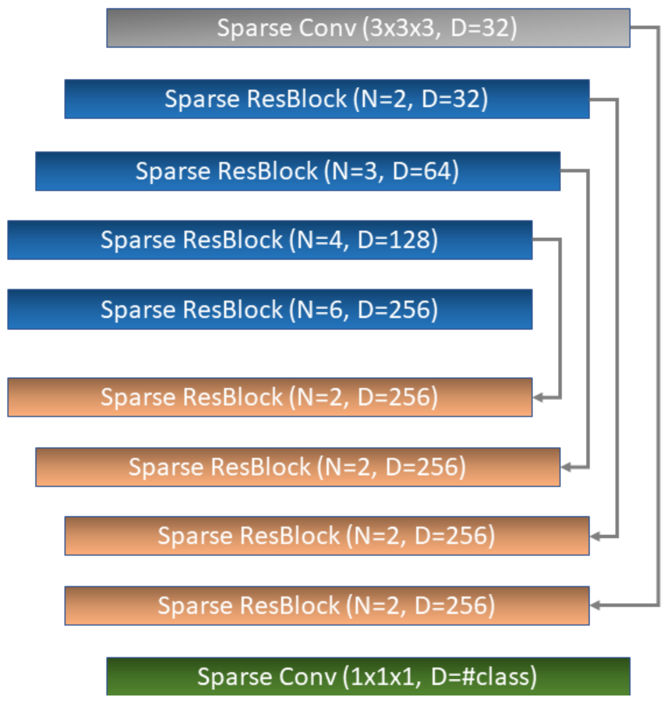
layer name	input layer	type	output size	layer params
sa1	point cloud (xyz)	SA	(2048,3+128)	(2048,0.2,[64, 64, 128])
sa2	sa1	SA	(1024,3+256)	(1024,0.4,[128, 128, 256])
sa3	sa2	SA	(512,3+256)	(512,0.8,[128, 128, 256])
sa4	sa3	SA	(256,3+256)	(256,1.2,[128, 128, 256])
fp1	sa3,sa4	FP	(512,3+512)	[512, 512]
fp2	sa2,sa3	FP	(1024,3+512)	[512, 512]

Table 1: PointNet++ Network Architecture used in Section 4, 5.1 and 5.2.

layer name	input layer	type	output size	layer params
sa1	point cloud (xyz)	SA	(4096,3+96)	(0.1, [16, 16, 32], 0.5, [32, 32, 64])
sa2	sa1	SA	(1024,3+256)	(0.5, [64, 64, 128], 1.0, [64, 96, 128])
sa3	sa2	SA	(256,3+512)	(0.1, [128, 196, 256], 0.5, [128, 196, 256])
sa4	sa3	SA	(64,3+1024)	(0.1, [256, 256, 512], 0.5, [256, 384, 512])
fp1	sa3,sa4	FP	(256,3+1024)	[512, 512]
fp2	sa2,sa3	FP	(1024,3+1024)	[512, 512]
fp3	sa1,sa2	FP	(4096,3+512)	[256, 256]
fp4	point cloud,sa1	FP	(16384,3+256)	[128, 128]

Table 2: PointnetMSG Network Architecture used in Section 5.3.

PointnetMSG model on LiDAR data used in Section 5.3. Table 2 shows the architecture details for PointnetMSG, which processes lidar point cloud for PointRCNN detection model. PointnetMSG contains four multi-scale set abstraction layers and four feature up-sampling layers. Multi-scale set abstraction samples points in different scales and process them with different MLPs. In here, we adopt the architecture design in [8], in which each set abstraction layer contains two point features produced with two ball-region radius and two MLPs. As shown in Table 2, each SA layer is specified by $(r^1, [c_1^1, ..., c_k^1], r^2, [c_1^2, ..., c_k^2])$, where r^1, r^2 indicates the ball-region radius for each scale and c_i^1, c_i^2 indicates the feature channel size of the i-th layer in the MLP of each scale. We directly apply the learnt PointnetMSG in



Sparse Conv (3x3x3, D=32)
Sparse ResBlock (N=2, D=32)
Sparse ResBlock (N=3, D=64)
Sparse ResBlock (N=4, D=128)
Sparse ResBlock (N=6, D=256)
Sparse ResBlock (N=2, D=256)
Sparse ResBlock (N=2, D=256)
Sparse ResBlock (N=2, D=256)
Sparse ResBlock (N=2, D=256)
Sparse Conv (1x1x1, D=#class)

Figure 1: UNet Network Architecture used in Section 4.2.

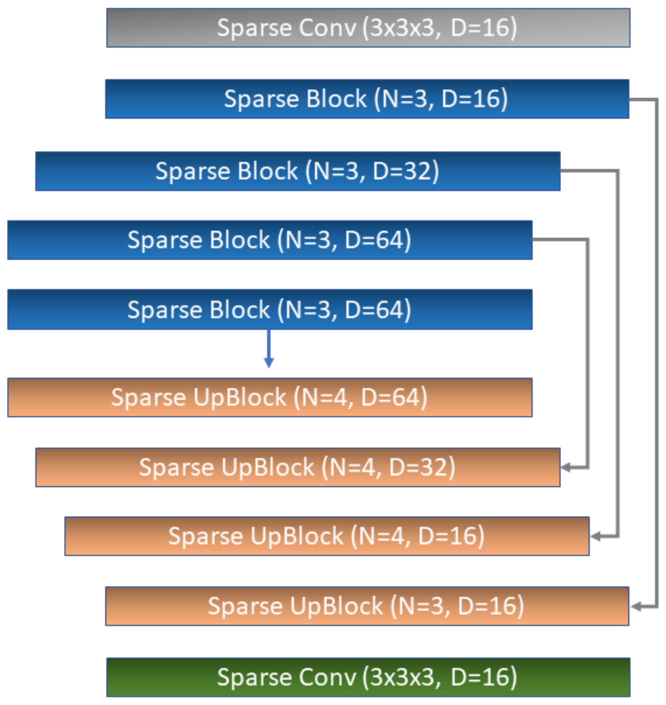

Sparse Conv (3x3x3, D=16)
Sparse Block (N=3, D=16)
Sparse Block (N=3, D=32)
Sparse Block (N=3, D=64)
Sparse Block (N=3, D=64)
Sparse UpBlock (N=4, D=64)
Sparse UpBlock (N=4, D=32)
Sparse UpBlock (N=4, D=16)
Sparse UpBlock (N=3, D=16)
Sparse Conv (3x3x3, D=16)

Figure 2: Spconv-UNet Network Architecture used in Section 5.3 of the main paper.

PointRCNN for detection evaluation in KITTI.

UNet model used in Section 4.2. Fig 1 shows the network architecture for UNet. It mainly contains four encoding resblock and four decoding resblock. We use sparse convolution and sparse resblock designed in [2]. For each sparse resblock, we first apply sparse convolution or deconvolution, depending on encoding or decoding, with kernel size 2 and stride 2. Then, we apply N number of sparse convolution layers with kernel size 3 and stride 1. D represents the output feature dimension. Since sparse convolution takes variable sized input and output, we do not specify the number of voxels in each layer here. We directly apply the learnt UNet backbone for different scene segmentation tasks.

Spconv-UNet model on LiDAR data used in Section 5.3. Fig 2 shows the network architecture for Spconv-UNet used for Part A^2 detection model. It mainly contains four sparse blocks for encoding and four sparse upblocks for decoding. We use sparse convolution and sparse resblock designed in [9]. For each sparse block/upblock, we show the number of convolution layers N and output feature dimension D. We directly adopted the architecture design in Part A^2. For KITTI detection evaluation, we directly load the learnt backbone for fine-tuning.

2. Training Details

2.1. Pretraining Details

As mentioned in the main paper, we use a standard SGD optimizer with momentum 0.9, cosine learning rate scheduler starting from 0.12 to 0.00012 and train the model for 1000 epochs with a batch size of 1024. We observed that pretraining for 400 epochs already gives good results, and 1000 epochs for pretraining only slightly improve the model.

2.2. Experimental Details for PointNet++

For all the VoteNet fine-tuning evaluations, we use the original configurations [6], where we apply Adam optimizer [4] and use a base learning rate 0.001 with a $10\times$ weight decrease at 80, 120 and 160 epochs. The model is trained for 180 epochs in total. Since the training set of S3DIS [1] only contains 200 training instances, we train for 360 epochs. We use a batch size of 8 for ScanNet, Matterport3D, and S3DIS, and a batch size of 16 for SUNRGBD. We use the same configuration for training from scratch and fine-tuning, and we only load the pretrained PointNet++ backbone during fine-tuning.

For the H3DNet fine-tuning, we only use one backbone network instead of the original four [12]. For initial learning rate and decays, we use the original configurations. We found that with $3\times$ PointNet++ backbone, we are able to re-produce the previous results reported in the paper with one backbone network. We use the same configuration for training from scratch and fine-tuning, and we only load the pretrained PointNet++ backbone during fine-tuning.

2.2.1 3D Shape Classification

In Section 5 of the main paper, we used the ModelNet dataset [11] for transfer learning. We trained linear classifiers on fixed features for this task and measured the classification accuracy.

Full finetuning. We now show results on the same task but using finetuning, *i.e.*, all the parameters of the backbone model are updated. We use the PointNet++ backbone to extract per-point feature and apply max-pooling to get the final global feature vector. We then apply one linear layer

Task	Scratch	DepthContrast
ModelNet Linear (Accuracy)	50.7	**85.4**
ModelNet Finetune (Accuracy)	88.2	**91.3**

Table 3: ModelNet [11]. We evaluate models by linear probing and full fine-tuning. We pretrain PointNet++ models using DepthContrast on the ScanNet-vid dataset.

Task	Without Aug	With Aug
ModelNet (Accuracy)	79.4	86.5

Table 4: Effect of Data Augmentation. We train with the Spherical CNN baseline [3] on the shape classification dataset [11], and we show test results with data augmentations during training versus without.

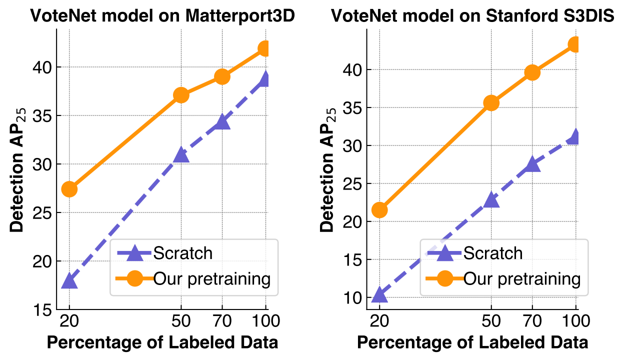


Figure 3: Label-efficiency for Matterport3D and S3DIS detection tasks. We pretrain a PointNet++ backbone model using our DepthContrast method on the ScanNet-vid dataset.

to get the final class labels. We use SGD+momentum optimizer with an initial learning rate 0.01. We use multistep LR scheduler with 10× weight decrease at 8, 16 and 24 epochs. We train for 28 epochs. We apply the data augmentation used in [6] during training. We use the same configuration for training from scratch and fine-tuning, and we only load the pretrained PointNet++ backbone during fine-tuning. For linear probing, we fix the pretrained weight and only fine-tune the last linear layer. In Table 3, we compare the fine-tuning and train from scratch results for both linear probing and full fine-tuning. The pretrained PointNet++ model provides consistent improvements.

Data Augmentation with Rotation-Equivariance Networks. We now show effects of data augmentations on rotation-equivariance neural networks. We use this Spherical CNN baseline [3], and we compare training from scratch without any data augmentation and with some data augmentation methods, such as random rotations. As shown in Table , despite of the nature of rotation-equivariance, applying data augmentations still provides considerate performance boost. Therefore, as long as there are output prediction or embedding variances with versus without data augmentations, those variances will serve as the main training signals for our self supervised pretraining method.

Task	Scratch	DepthContrast
ScanNet segmentation (mIOU)	70.3	**71.2**

Table 5: ScanNet scene segmentation. We evaluate UNet models after finetuning on the ScanNet scene segmentation task. We pretrain the DepthContrast UNet models on the ScanNet-vid dataset.

2.2.2 Label efficiency of PointNet++

We follow the same settings as Section 4.1.4 of the main paper and evaluate the label efficiency of the PointNet++ pretrained using DepthContrast. In Figure 1 (main paper) we showed that DepthContrast pretraining was label efficient on the ScanNet and SUNRGBD datasets. In Fig 3, we show the label efficiency plots for Matterport3D and S3DIS downstream detection tasks. Our results are consistently better than training from scratch across all the detection benchmarks used in the paper.

2.3. Experimental Details for UNet

For all the UNet fine-tuning evaluation, we use SGD+momentum optimizer with an initial learning rate 0.1. We use Polynomial LR scheduler with a power factor of 0.9. Weight decay is 0.0001. For voxel size, we use 0.05(5cm) for S3DIS and Synthia and 0.04(4cm) for ScanNet. We use the original data augmentation techniques in [2]. We use batch size 48 for S3DIS and 56 for ScanNet and Synthia. We train the model with 8 V100 GPUs with data parallelism for 20000 iterations for all three tasks. We use the same configuration for training from scratch and fine-tuning, and we only load the pretrained UNet backbone during fine-tuning.

2.3.1 ScanNet Segmentation

For ScanNet scene segmentation, due to memory issue, we increased the voxel size from the default 0.02(2cm) to 0.04(4cm), which leads to different scratch training results compared to [2]. Although we are pretraining and fine-tuning on the same dataset, our approach still provides improvements as shown in Table 5.

2.3.2 Label efficiency of UNet

We pretrain a UNet model using DepthContrast on the ScanNet-vid dataset. We finetune this model for scene segmentation task. In Fig 4, we show the data efficiency plot for S3DIS scene segmentation dataset. Our approach provides consistent performance boost for different percentages of training data used.

3. Experimental Details for Lidar Data

We now present the experimental details when using DepthContrast on LiDAR 3D data (Section 5.3 of the main paper).

3.1. Fine-tuning Details

We use the original configuration from PointRCNN [8] and Part A^2 [9] to get the scratch training results for different splits of training dataset. For PointRCNN, we use AdamW optimizer [5] with an initial learning rate 0.01, weight decay 0.01, and momentum 0.9. For Part A^2, we also use AdamW optimizer [5] with an initial learning rate 0.003, weight decay 0.01, and momentum 0.9. We use batch size 24 for PointRCNN and batch size 16 for Part A^2. We train both models for 80 epochs, and the learning rate will drop by 10× at 35 and 45 epochs. For both methods, we apply the same data augmentation and processing pipelines in [10].

We use the same configuration for training from scratch and fine-tuning for 5%, 10%, 20% and 50% of the labeled training data. For 100% training data, we observed overfitting issue for classes with fewer training instances, such as cyclist and pedestrian. Thus, we increased the initial learning rate by 2×. For PointRCNN, we also decreased the number of training epochs to 60 and set the first weight drop at 30 epochs. After modifying those parameters, we observed that the performance of scratch training didn't change.

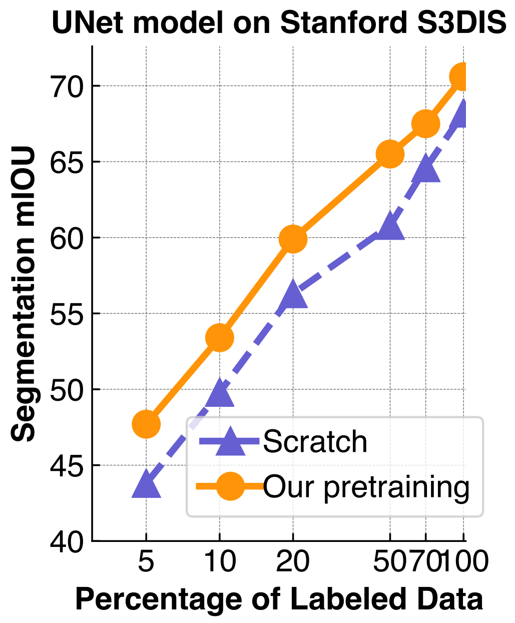


Figure 4: Label-efficiency for S3DIS scene segmentation task using the UNet model on voxel input. UNet model was pretrained on ScanNet-vid using our DepthContrast method.

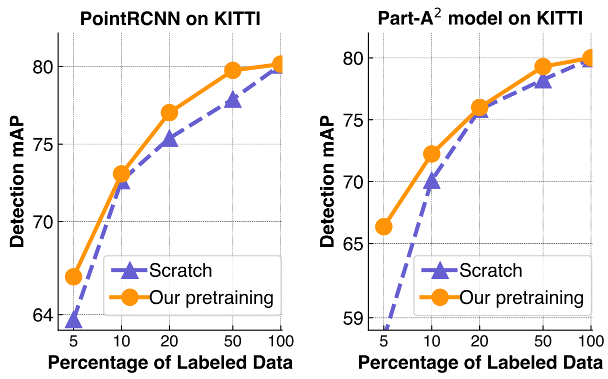


Figure 5: Label-efficiency for KITTI `car` detection at moderate difficulty level. We evaluate on the val split of the KITTI dataset.

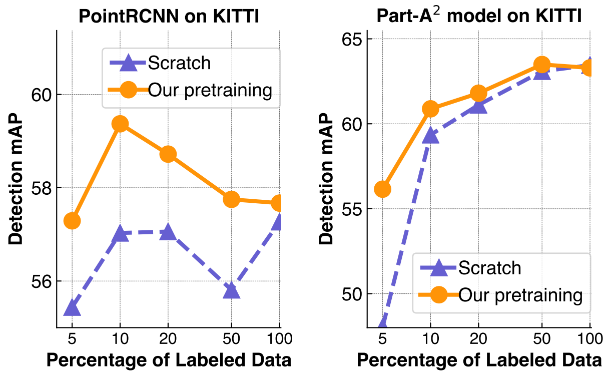


Figure 6: Label-efficiency evaluation for KITTI `pedestrian` detection at moderate difficulty level. We use the val split of the KITTI dataset.

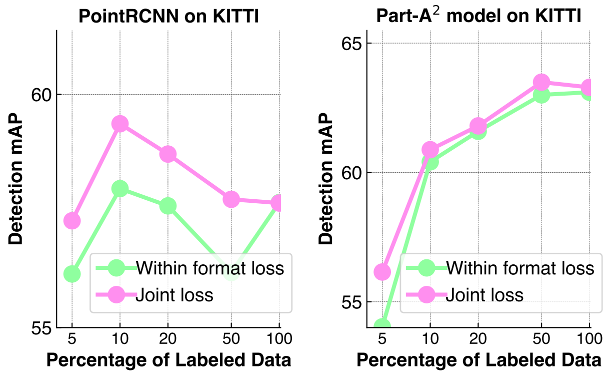


Figure 7: Comparison between within format and joint training loss. Label-efficiency evaluation for `pedestrian` detection at moderate difficulty level of the KITTI val split

3.2. Results on KITTI

In the main paper, due to space constraints, we only showed the results on the `cyclist` class in the KITTI dataset. We present results on the remainder classes.

We show the label efficiency evaluation results for `car` detection in KITTI in Fig 5. For simplicity, we only show the results at moderate difficulty level. The results at other difficulty levels maintain similar pattern. DepthContrast provides a performance boost with fewer training instances, especially with 5% of labeled data.

In Fig 6, we show the label efficiency evaluation results for `pedestrian` detection in KITTI. Our pretraining provides consistent gain over scratch for PointRCNN model. For Part A^2, our pretraining also provides significant performance boost with fewer training instances.

Advantage of joint training over within-format loss. Similar to Section 4.3 of the main paper, we analyze if training jointly with the within and across-format losses provides better transfer performance. Specifically, we compare (Equation 3 of the main paper) our full joint loss with the within-format only loss. We pretrain separate models with these losses and evaluate them on the KITTI dataset.

In Fig 7, we show the label efficiency evaluation results for `pedestrian` detection. We can see that with joint loss training, the model provides more performance gain with fewer training instances, which proves that the benefit of joint training holds across different pretraining data and architectures.

References

[1] Iro Armeni, Sasha Sax, Amir Roshan Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *CoRR*, abs/1702.01105, 2017.

[2] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[3] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning so (3) equivariant representations with spherical cnns. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–68, 2018.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[5] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[6] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. *arXiv preprint arXiv:1904.09664*, 2019.

[7] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[8] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019.

[9] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *arXiv preprint arXiv:1907.03670*, 2019.

[10] OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. `https://github.com/open-mmlab/OpenPCDet`, 2020.

[11] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1912–1920, 2015.

[12] Zaiwei Zhang, Bo Sun, Haitao Yang, and Qixing Huang. H3dnet: 3d object detection using hybrid geometric primitives. *arXiv preprint arXiv:2006.05682*, 2020.