Point-BERT : Pre-Training 3D Point Cloud Transformers with Masked Point Modeling

Supplementary Material

A. More Analysis

More Visual Results: We provide more visual results on the real-world ScanObjectNN dataset [7] to show the generality of our Point-BERT model. Since our model is trained on the synthetic dataset ShapeNet [1], the masked point prediction results on ScanObjectNN can reflect the performance of our model on challenging scenarios with both unseen objects and domain gap. The results are illustrated in Figure 1. Our Point-BERT model can correctly predict the point tokens of the masked regions and reconstruct the input point cloud through the pre-trained dVAE decoder. The generic knowledge of local point cloud structure can be learned by our model, for it also works well on the cross-domain data that it have not been seen during the MPM training. It further confirms our claim that the proposed Point-BERT can encode the local geometric patterns into discrete tokens, and a point cloud (even never seen before) can be represented as a sequence of such tokens.

Complexity Analysis: We provide the detailed complexity analysis of our method in the following table. As shown in the table, even with more parameters, our method with pure Transformer is still competitive in 'FLOPs' and much faster on GPU. Besides, more parameters facilitate Transformer to learn more general knowledge for a superior result.

Methods	Params	GFLOPs	Throughput	ModelNet40 Acc.
PointNet++	1.5M	0.9	108.7 pcs	91.9
PCT	2.9M	2.3	520.3 pcs	93.2
KPConv	14.3M	–	140.0 pcs	92.9
DGCNN	1.8M	2.7	315.1 pcs	92.9
Point-BERT	22.1M	2.3	1237.9 pcs	93.2

B. Implementation Details

B.1. Discrete VAE

Architecture: Our dVAE consists of a tokenizer and a decoder. Specifically, the tokenizer contains a 4-layer DGCNN [9], and the decoder involves a 4-layer DGCNN followed by a FoldingNet [10]. The detailed network architecture of our dVAE is illustrated in Table 1, where C_{in} and C_{out} are the dimension of input and output features, C_{middle} is the dimension of the hidden layers. N_{out} is the number of point patches in each layer, and K is the number of neighbors in kNN operation. Additionally, `FoldingLayer` concatenates a 2D grids to the inputs and finally generates 3D point clouds.

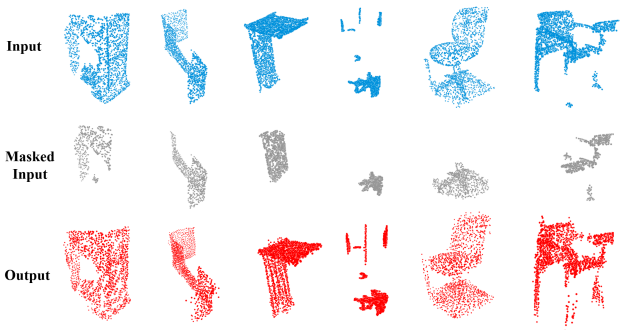

Figure 1. Visual results of masked point clouds reconstruction on the ScanObjectNN dataset, using our Point-BERT model trained on ShapeNet. Taking a point cloud as original input (top), we mask a portion of the original point cloud (middle) and apply Point-BERT model to predict the masked point tokens. We can obtain the reconstructed point cloud (bottom) by feeding the complete point tokens to the decoder of dVAE.

Optimization: During the training phase, we consider reconstruction loss and distribution loss simultaneously. For reconstruction, we follow PoinTr [11] to supervise both coarse-grained prediction and fine-grained prediction with the ground-truth point cloud. The ℓ_1-form Chamfer Distance is adopted, which is calculated as:

$$d_{CD}^{\ell_1}(\mathcal{P},\mathcal{G}) = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\min_{g\in\mathcal{G}}\|p-g\| + \frac{1}{|\mathcal{G}|}\sum_{g\in\mathcal{G}}\min_{p\in\mathcal{P}}\|g-p\|, \quad (1)$$

where $\mathcal{P}$ represents the prediction point set and $\mathcal{G}$ represents the ground-truth point set. Except for the reconstruction loss, we follow [5] to optimize the KL-divergence $\mathcal{L}_{KL}$ between the predicted tokens' distribution and a uniform prior. The final objective function is

$$\mathcal{L}_{\text{dVAE}} = d_{CD}^{\ell_1}(\mathcal{P}_{fine},\mathcal{G}) + d_{CD}^{\ell_1}(\mathcal{P}_{coarse},\mathcal{G}) + \alpha\mathcal{L}_{KL}. \quad (2)$$

Table 1. Detailed architecture of our models. C_{in}/C_{out} represents the dimension of input/output features, and N_{out} is the number of points in the query point cloud. K is the number of neighbors in kNN operation. C_{middle} is the dimension of the hidden layers for MLPs.

Module	Block	C_{in}	C_{out}	K	N_{out}	C_{middle}
dVAE Tokenizer	Linear	256	128			
	DGCNN	128	256	4	64	
	DGCNN	256	512	4	64	
	DGCNN	512	512	4	64	
	DGCNN	512	1024	4	64	
	Linear	2304	8192			
dVAE Decoder	Linear	256	128			
	DGCNN	128	256	4	64	
	DGCNN	256	512	4	64	
	DGCNN	512	512	4	64	
	DGCNN	512	1024	4	64	
	Linear	2304	256			
	MLP	256	48			1024
	FoldingLayer	256	3			1024
Classification Head	MLP	768	N_{cls}			256
Segmentation Head	MLP	387	384			384×4
	DGCNN	384	512	4	128	-
	DGCNN	512	384	4	128	
	DGCNN	384	512	4	256	
	DGCNN	512	384	4	256	
	DGCNN	384	512	4	512	
	DGCNN	512	384	4	512	
	DGCNN	384	512	4	2048	
	DGCNN	512	384	4	2048	

Table 2. Experiment setting for training the dVAE.

config	value
optimizer	AdamW [4]
learning rate	5e-4
weight decay	5e-4
learning rate schedule	cosine [3]
warmingup epochs	10
augmentation	RandSampling
batch size	64
number of points	1024
number of patches	64
patch size	32
training epochs	300
dataset	ShapeNet [1]

Experiment Setting: We report the default setting for dVAE training in Table 2.

Hyper-parameters of dVAE: We set the size of the learnable vocabulary to 8192, and each 'word' in it is a 256-dim vector. The most important and sensitive hyper-parameters of dVAE are α for $\mathcal{L}_{KL}$ and the temperature τ for Gumbel-softmax. We set α to 0 in the first 18 epochs (about 10,000 steps) and gradually increase to 0.1 in the following 180 epochs (about 100,000 steps) using a cosine schedule. As for τ, we follow [5] to decay it from 1 to 0.0625 using a cosine schedule in the first 180 epochs (about 100,000 steps).

config	value
optimizer	AdamW
learning rate	5e-4
weight decay	5e-2
learning rate schedule	cosine
warmingup epochs	3
augmentation	ScaleAndTranslate
batch size	128
number of points	1024
number of patches	64
patch size	32
mask ratio	[0.25, 0.45]
mask type	rand mask
training epochs	300
dataset	ShapeNet

Table 3. Experiment setting for Point-BERT pre-training

B.2. Point-BERT:

Architecture: We follow the standard Transformer [2] architecture in our experiments. It contains a stack of Transformer blocks [8], and each block consists of a multi-head self-attention layer and a FeedForward Network (FFN). In these two layers, LayerNorm (LN) is adopted.

Multi-head Attention: Multi-head attention mechanism enables the network to jointly consider information from different representation subspaces [8]. Specifically, given the input values V, keys K and queries Q, the multi-head attention is computed by:

$$\text{MultiHead}(Q, K, V) = W^o \text{Concat}(\text{head}_1, ..., \text{head}_h), \quad (3)$$

where W^o is the weights of the last linear layer. The feature of each head can be obtained by:

$$\text{head}_i = \text{softmax}(\frac{QW_i^Q(KW_i^K)^T}{\sqrt{d_k}})VW_i^V, \quad (4)$$

where W_i^Q, W_i^K and W_i^V are the linear layers that project the inputs to different subspaces and d_k is the dimension of the input features.

Feed-forward network (FFN): Following [8], two linear layers with ReLU activations and dropout are adopted as the feed-forward network.

Point-BERT pre-training: We report the default setting for our experiments in Point-BERT pretraining in Table 3. The pre-training are conducted on ShapeNet.

End-to-end finetuning: We finetune our Point-BERT model follow the common practice of supervised models

config	value
optimizer	AdamW
learning rate	5e-4
weight decay	5e-2
learning rate schedule	cosine
warmingup epochs	10
augmentation	ScaleAndTranslate
batch size	32(C),16(S)
number of points	1024(C),2048 (S)
number of patches	64(C),128(S)
patch size	32
training epochs	300

Table 4. Experiment setting for end-to-end finetuning. S represents segmentation task, C represents classification task.

strictly. The default setting for end-to-end finetuning is in Table 4.

Hyper-parameters of Transformers: We set the number of blocks in the Transformer to 12. The number of heads in each multi-head self-attention layer is set to 6. The feature dimension of the transformer layer is set to 384. We follow [6] to adopt the stochastic depth strategy with a drop rate of 0.1.

Classification Head: An two-layer MLP with dropout is applied as our classification head. In classification tasks, we first take the output feature of [CLS] token out, and max-pool the rest of nodes' features. These two features are then combined together and sent into the classification head. The detailed architecture of classification head is shown in Table 1, where N_{cls} is the number of class for a certain dataset.

Segmentation Head: There are no downsampling layers in the standard Transformers, making it challenging to perform dense prediction based on a single-resolution feature map. We adopt an upsampling-propagation strategy to solve this problem, consisting of two steps: 1) Geometry-based feature upsampling and 2) Hierarchical feature propagation.

We extract features from different layers of the Transformer, where features from shallow layers tend to capture low-level information, while features from deeper layers involve more high-level information. To upsample the feature maps to different resolutions, we first apply FPS to the origin point cloud and obtain point clouds at various resolutions. Then we upsample the feature maps from different layers to different resolutions accordingly. As shown in the left part of Figure 2, 'A' is a point from the dense point cloud, and 'a','b','c' are its nearest points in the sparser point cloud, with distance of d_a, d_b and d_c respectively. We obtain the point feature of 'A' based on weighted addition of those features, which can be written as:

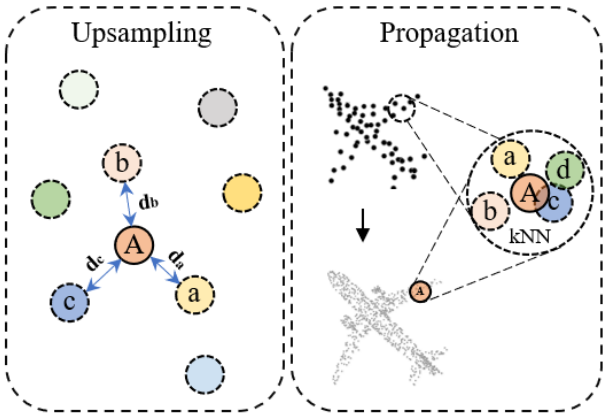

Figure 2. Two main operations of our segmentation head: 1) Upsampling: upsample the feature map for the sparse point cloud to the dense point cloud. 2) Propagation: propagate the feature hierarchically from deep layers to shallow layers for dense prediction.

$$\mathcal{F}_A = \text{MLP}(\text{Concat}(\frac{\sum_{i\in[a,b,c]} \frac{1}{d_i}\mathcal{F}_i}{\sum_{i\in[a,b,c]} \frac{1}{d_i}}, p_A)), \quad (5)$$

where p_A represents the coordinates of point 'A'.

After obtaining the feature maps at different resolutions, we perform feature propagation from coarse-grained feature maps to fine-grained feature maps. As shown in the right part of Figure 2, for a point 'A' in the dense point cloud, we find its k nearest points in the sparser point cloud. Then a lightweight DGCNN [9] is used to update the feature of 'A'. We hierarchically update the feature with the resolution increases and finally obtain the dense feature map, which can be used for segmentation tasks. The detailed architecture for the segmentation head is shown in Table 1.

References

[1] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 1, 2

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2

[3] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 2

[4] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. 2018. 2

[5] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021. 1, 2

[6] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 3

[7] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *ICCV*, 2019. 1

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 2

[9] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *TOG*, 2019. 1, 3

[10] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, 2018. 1

[11] Xumin Yu, Yongming Rao, Ziyi Wang, Zuyan Liu, Jiwen Lu, and Jie Zhou. Pointr: Diverse point cloud completion with geometry-aware transformers. In *ICCV*, 2021. 1