Supplementary Material
DensePoint: Learning Densely Contextual Representation for Efficient Point Cloud Processing

Yongcheng Liu Bin Fan Gaofeng Meng Jiwen Lu Shiming Xiang Chunhong Pan

A. Outline

This supplementary material provides: (1) further investigations of the proposed DensePoint (Sec B); (2) more shape retrieval examples of DensePoint and some analysis (Sec C); (3) network configuration details (Sec D). (4) training details (Sec E)

B. Further Investigations

In this section, we provide further investigations of DensePoint on four aspects. Specifically, the discussion of neighborhood method is presented in Sec B.1. The effect of dropout on $\mathbf{f}_{\mathcal{N}(x)}$ in Eq. (4) is analyzed in Sec B.2. The impact of network depth on classification performance is investigated in Sec B.3. The memory and runtime are summarized in Sec B.4. All the investigations are conducted on ModelNet40 dataset.

B.1. Neighborhood Method

In the main paper, the local convolutional neighborhood $\mathcal{N}(x)$ in Eq. (1) is set to be a spherical neighborhood, from which a fixed number of neighbors are randomly sampled for batch processing. We compare this strategy (Random-In-Sphere) with another typical one, *i.e.*, k-nearest neighbor (k-NN). For a fair comparison, the models with these two strategies are configured with the same settings. Table I summarizes the results.

As can be seen, the model with Random-In-Sphere performs better. We speculate that the model with k-NN will suffer from the distribution inhomogeneity of points. In this case, the contextual learning in DensePoint will be less effective, as the receptive fields will be confined to a local region with large density, which leads to ignoring those sparse points that are essential for recognizing the implicit shape. By contrast, Random-In-Sphere can have a better coverage of points even in the case of inhomogeneous distribution.

B.2. Dropout on $\mathbf{f}_{\mathcal{N}(x)}$ in Eq. (4)

The dropout technique can force the whole network to behave as an ensemble of a lot of subsets and reduce the risk of model overfitting. To analyze its effect on DensePoint, we apply it with different ratios on $\mathbf{f}_{\mathcal{N}(x)}$ in Eq. (4). The results are summarized in Table II. As can be seen, the best result of 93.2% can be achieved with a dropout ratio of 20%.

Table I. The results (%) of two neighborhood strategies. The number of neighbors is equally set in each layer of the two models.

neighborhood method	acc.
k-NN	91.3
Random-In-Sphere	**93.2**

Table II. The results (%) of dropout with different ratios applied on $\mathbf{f}_{\mathcal{N}(x)}$ in Eq. (4).

ratio (%)	0	10	20	30	40	50
acc.	92.9	92.8	**93.2**	93.0	92.8	92.5

Table III. The results (%) of different network depths (fully connected layers are not included).

#layers	#params	#FLOPs/sample	acc.
6	0.53M	148M	92.1
9	0.56M	510M	92.9
11	0.67M	651M	**93.2**
15	0.78M	779M	93.0
19	0.88M	1222M	92.7
23	1.03M	1416M	92.6

B.3. Network Depth

We further explore the impact of the network depth (fully connected layers are not included) on classification performance. The results are summarized in Table III. Surprisingly, a 6-layer network equipped with DensePoint can achieve an accuracy of 92.1% with only 0.53M params and 148M FLOPs/sample. This even outperforms PointNet++ [33] (accuracy 90.7%, params 1.48M [26], FLOPs/sample 1684M [26]) by 15% in error rate, whilst being one order of magnitude faster in terms of FLOPs/sample. We also observe that it is unnecessary to develop a very deep network (*e.g.*, 23 layers) with DensePoint, as it increases complexity without bringing any gain. Eventually, the best result of 93.2% can be reached with acceptable complexity by an

Table IV. Time and memory of classification network, where k is network narrowness, L is network depth. The statistics of all the models are summarized with batch size 16 on NVIDIA TITAN Xp, and time is the mean time of 1000 tests. The compared models are tested using their available official codes.

method	#points	Time (ms)		Memory (GB)	
		training	test	training	test
PointNet [31]	1024	55	22	1.318	0.469
PointNet++ [33]	1024	195	47	8.311	2.305
DGCNN [52]	1024	300	68	4.323	1.235
PointCNN [26]	1024	55	38	2.501	1.493
Ours (k=24, L=11)	1024	21	10	3.745	1.228
Ours (k=24, L=6)	1024	10	5	1.468	0.886

11-layer network.

B.4. Memory and runtime

The memory and runtime of the proposed DensePoint are summarized in Table IV. As can be seen, the model (L=11) is competitive while another model (L=6) is the best one in terms of efficiency. Actually, the memory and training time issues in dense connection mode are greatly alleviated due to the shallow design of DensePoint and our highly-efficient implementation. Moreover, although extremely deep network could be unnecessary for 3D currently, in case of very deep DensePoint in the future, the technique of Shared Memory Allocations can be applied to achieve linear memory complexity.

C. Shape Retrieval

In this section, we show more shape retrieval examples in Fig. 1. As can be seen, compared with PointNet [31], our DensePoint obtains superior shape identification results. Specifically, PointNet is confused between the query "bottle" and the sample "vase" due to their similar shapes. Nevertheless, DensePoint with densely contextual semantics acquired can identify them accurately. We notice that DensePoint could also be confused for some very alike shapes, *e.g.*, the query "bench" and the sample "tv_stand". This could be improved by learning to weight multi-level contextual information instead of identically aggregating all levels of information. We leave it as future work.

D. Network Configuration Details

In this section, we present the configuration details of three networks on shape classification, shape part segmentation and normal estimation, respectively. For clearness, we describe the layer and corresponding setting format as follows:

PPool: [downsampling rate, neighborhood radius, #number of neighbors, SLP$^{\phi}$(#input channels, #output channels)]. The global pooling is achieved by directly applying PConv to convolve all points.

ePConv: [neighborhood radius, #number of neighbors, SLP$^{\widetilde{\phi}}$(#input channels, #output channels, #group number), SLP$^{\psi}$(#input channels, #output channels), dropout ratio].

FP (feature propagation layer): MLP(#channels, $\cdots$). Feature propagation layer [33] is used for transforming the features that are concatenated from current interpolated layer and long-range connected layer. We employ a multi-layer perceptron (MLP) to implement this transformation.

FC (fully connected layer): [(#input channels, #output channels), dropout ratio]. Note that the dropout technique is applied for all FC layers except for the last FC layer (used for prediction).

In addition, except for the last prediction layer, all layers (including the inside perceptrons) are followed with batch normalization and ReLU activator. The output shape is in the format of (#feature dimension, #number of points).

D.1. Shape Classification Network

The configuration details of shape classification network are presented in Table VI. The network has 14 layers in total, which comprises 3 PPools (the last one is global pooling layer) and 2 DensePoints (the 1$^{\text{st}}$ one has 3 layers while the 2$^{\text{nd}}$ one has 5 layers), followed by 3 FC layers.

D.2. Shape Part Segmentation Network

Table V summarizes the configuration details of shape part segmentation network. As it shows, the network has 23 layers in total, which comprises 4 PPools, 3 DensePoints (4 layers, 6 layers and 3 layers in 2$^{\text{nd}}$ stage, 3$^{\text{rd}}$ stage and 4$^{\text{th}}$ stage respectively) and 4 FP layers, followed by 2 FC layers. As in [31, 33], we concatenate the one-hot encoding (16-d) of the object label to the last feature layer.

D.3. Normal Estimation Network

The normal estimation network is presented in Table VII. It is almost the same as the segmentation network, except for three aspects: (1) the input becomes 1024-d and the one-hot encoding becomes 40-d for ModelNet40 dataset; (2) the settings of some layers are slightly changed to be consistent with the 1024-d input; (3) the final output becomes 3-d for normal prediction. As done in the segmentation network, we also concatenate the one-hot encoding (40-d) of the object label to the last feature layer.

E. Training Details

Our DensePoint is implemented using Pytorch. The Adam optimization algorithm is employed for training, with a mini-batch size of 32. The momentum for batch normalization starts with 0.9 and decays with a rate of 0.5 every 20 epochs. The learning rate begins with 0.001 and decays with a rate of 0.7 every 20 epochs. The weight is initialized using the techniques introduced by He *et al*. [1].

Table V. The configuration details of shape part segmentation network. "long-range" indicates the long-range connections (see Fig. 3(b) in the main paper). K is the number of classes.

stage	layer type	setting detail	output shape	long-range
-	Input	-	(3, 2048)	FP_4
1	PPool	[1/2, 0.1, 32, (3, 64)]	(64, 1024)	FP_3
2	PPool	[1/4, 0.2, 64, (64, 128)]	(128, 256)	
	ePConv	[0.3, 32, (128, 96, 2), (96, 24), 20%]	(24, 256)	
	ePConv	[0.3, 32, (152, 96, 2), (96, 24), 20%]	(24, 256)	
	ePConv	[0.3, 32, (176, 96, 2), (96, 24), 20%]	(24, 256)	
	ePConv	[0.3, 32, (200, 96, 2), (96, 24), 20%]	(24, 256)	
		The output of DensePoint in 2nd stage	(224, 256)	FP_2
3	PPool	[1/4, 0.3, 32, (224, 192)]	(192, 64)	
	ePConv	[0.5, 16, (192, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (216, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (240, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (264, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (288, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (312, 96, 2), (96, 24), 20%]	(24, 64)	
		The output of DensePoint in 3rd stage	(336, 64)	FP_1
4	PPool	[1/4, 0.8, 32, (336, 360)]	(360, 16)	
	ePConv	[0.8, 8, (360, 96, 2), (96, 24), 20%]	(24, 16)	
	ePConv	[0.8, 8, (384, 96, 2), (96, 24), 20%]	(24, 16)	
	ePConv	[0.8, 8, (408, 96, 2), (96, 24), 20%]	(24, 16)	
		The output of DensePoint in 4th stage	(432, 16)	
	FP_1	(768, 512, 512)	(512, 64)	
	FP_2	(736, 384, 384)	(384, 256)	
	FP_3	(448, 256, 256)	(256, 1024)	
	FP_4	(259, 128, 128)	(128, 2048)	
	FC	[(128+16[1], 128), 50%]	(128, 2048)	
	FC	[(128, K), -] $\rightarrow$ softmax	(K, 2048)	

[1] This is the one-hot encoding of the object label on ShapeNet part dataset.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *ICCV*, pages 1026–1034, 2015. 2

Table VI. The configuration details of shape classification network. K is the number of classes.

stage	layer type	setting detail	output shape
-	Input	-	(3, 1024)
1	PPool	[1/2, 0.25, 64, (3, 96)]	(96, 512)
	ePConv	[0.2, 32, (96, 96, 2), (96, 24), 20%]	(24, 512)
	ePConv	[0.2, 32, (120, 96, 2), (96, 24), 20%]	(24, 512)
	ePConv	[0.2, 32, (144, 96, 2), (96, 24), 20%]	(24, 512)
		The output of DensePoint in 1st stage	(168, 512)
2	PPool	[1/4, 0.3, 64, (168, 144)]	(144, 128)
	ePConv	[0.4, 16, (144, 96, 2), (96, 24), 20%]	(24, 128)
	ePConv	[0.4, 16, (168, 96, 2), (96, 24), 20%]	(24, 128)
	ePConv	[0.4, 16, (192, 96, 2), (96, 24), 20%]	(24, 128)
	ePConv	[0.4, 16, (216, 96, 2), (96, 24), 20%]	(24, 128)
	ePConv	[0.4, 16, (240, 96, 2), (96, 24), 20%]	(24, 128)
		The output of DensePoint in 2nd stage	(264, 128)
3	PPool	[-, -, 128, (264, 512)]	(512,)
	FC	[(512, 512), 50%]	(512,)
	FC	[(512, 256), 50%]	(256,)
	FC	[(256, K), -] $\rightarrow$ softmax	(K,)

Table VII. The configuration details of normal estimation network. "long-range" indicates the long-range connections (see Fig. 3(b) in the main paper).

stage	layer type	setting detail	output shape	long-range
-	Input	-	(3, 1024)	FP_4
1	PPool	[1, 0.2, 32, (3, 64)]	(64, 1024)	FP_3
2	PPool	[1/4, 0.2, 32, (64, 128)]	(128, 256)	
	ePConv	[0.3, 32, (128, 96, 2), (96, 24), 20%]	(24, 256)	
	ePConv	[0.3, 32, (152, 96, 2), (96, 24), 20%]	(24, 256)	
	ePConv	[0.3, 32, (176, 96, 2), (96, 24), 20%]	(24, 256)	
	ePConv	[0.3, 32, (200, 96, 2), (96, 24), 20%]	(24, 256)	
		The output of DensePoint in 2nd stage	(224, 256)	FP_2
3	PPool	[1/4, 0.3, 32, (224, 192)]	(192, 64)	
	ePConv	[0.5, 16, (192, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (216, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (240, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (264, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (288, 96, 2), (96, 24), 20%]	(24, 64)	
	ePConv	[0.5, 16, (312, 96, 2), (96, 24), 20%]	(24, 64)	
		The output of DensePoint in 3rd stage	(336, 64)	FP_1
4	PPool	[1/4, 0.8, 32, (336, 360)]	(360, 16)	
	ePConv	[0.8, 8, (360, 96, 2), (96, 24), 20%]	(24, 16)	
	ePConv	[0.8, 8, (384, 96, 2), (96, 24), 20%]	(24, 16)	
	ePConv	[0.8, 8, (408, 96, 2), (96, 24), 20%]	(24, 16)	
		The output of DensePoint in 4th stage	(432, 16)	
	FP_1	(768, 512, 512)	(512, 64)	
	FP_2	(736, 384, 384)	(384, 256)	
	FP_3	(448, 256, 256)	(256, 1024)	
	FP_4	(259, 128, 128)	(128, 1024)	
	FC	[(128+40[2], 128), 50%]	(128, 1024)	
	FC	[(128, 3), -]	(3, 1024)	

[2] This is the one-hot encoding of the object label on ModelNet40 dataset.

Figure 1. Retrieval examples on ModelNet40 dataset. Top-10 matches are shown for each query, with the 1st line for PointNet [31] and the 2nd line for our DensePoint. The mistakes are highlighted in red.