Dense-Resolution Network for Point Cloud Classification and Segmentation Supplementary Material

1. Overview

In this supplementary material, we present more contents of our paper *Dense-Resolution Network for Point Cloud Classification and Segmentation*. To be specific, we provide the implementation of the *Adaptive Dilated Point Grouping (ADPG)* method and *loss function* for the experiments. Besides, we show the details of our Multi-resolution (MR) branch. By comparing the relevant model parameters with others on *ModelNet40* dataset, we discuss the complexity of our network.

2. Implementation

In the main paper, we introduce the pipeline for the *ADPG* method and the design of *loss function* for training. In this section, we provide more practical details in our experiments.

2.1. ADPG Learning Process

In practice, d_{max} is an empirical parameter which may vary between the data scales or networks. In our experiments, we set $d_{max} = 5$ for ShapeNet Part, ModelNet40 and ScanObjectNN datasets, since they share the similar scales of point clouds.

Assume that we already have the indices $\mathcal{I}_{N\times(k\cdot d_{max})}$ and metrics $E_{N\times(k\cdot d_{max})}$ for $k \cdot d_{max}$ candidates, the crucial step of *ADPG* is to learn a certain dilation factor for each point based on the known information. In Section 3.1 of the paper, we present the general description for this process:

$$\mathcal{D}_N = \mathcal{S}\Big(\mathcal{J}\big(\sigma\big(\mathcal{M}(E_{N\times(k\cdot d_{max})})\big)\big)\Big)$$

Specifically, we apply a two-layer $\mathcal{M}$: $Conv_{[1,1]}^{\{(k\cdot d_{max}/2),1\}}$ first, then activate corresponding *negative* values using a *logistic function*: $y = 1/(1+e^{-x})$. Since the values are in between 0 and 1, $\mathcal{J}$ can further enlarge the variance by projecting them to another interval. Here we expect the values to be in $[0.5, 5.5]$, thus a simple linear projection function $y = 5 \cdot x + 0.5$ serves as $\mathcal{J}$. Finally, we adopt *round function* as $\mathcal{S}$ to scale the continuous values in $[0.5, 5.5]$, by which an integer in $\{1, 2, 3, 4, 5\}$ can be assigned as the dilation factor for each point. To summarize, the dilation factors learning in our implementation follows:

$$\mathcal{D}_N = \left\lceil 5 \cdot \frac{1}{1 + e^{\left(Conv_{[1,1]}^{\{(k\cdot d_{max}/2),1\}}(E_{N\times(k\cdot d_{max})})\right)}} + 0.5 \right\rceil$$

2.2. Loss Function

As discussed in the Section 3.3, the total loss for training is the sum of cross-entropy loss $\mathcal{L}_{ce}$ and weighted error-minimizing module losses: $\sum w_i \cdot \mathcal{L}_{er_i}$. In practice, we apply 4 error-minimizing modules in the Full-resolution (FR) branch of our network, adopting the similar layers and feature dimensions as in [4]. In terms of our experiments on the ShapeNet Part, ModelNet40 and ScanObjectNN datasets, we empirically set a larger weight for the first error-minimizing module ($w_1 = 0.1$) since its output affects the both branches and constrains the network learning at the beginning. In contrast, the

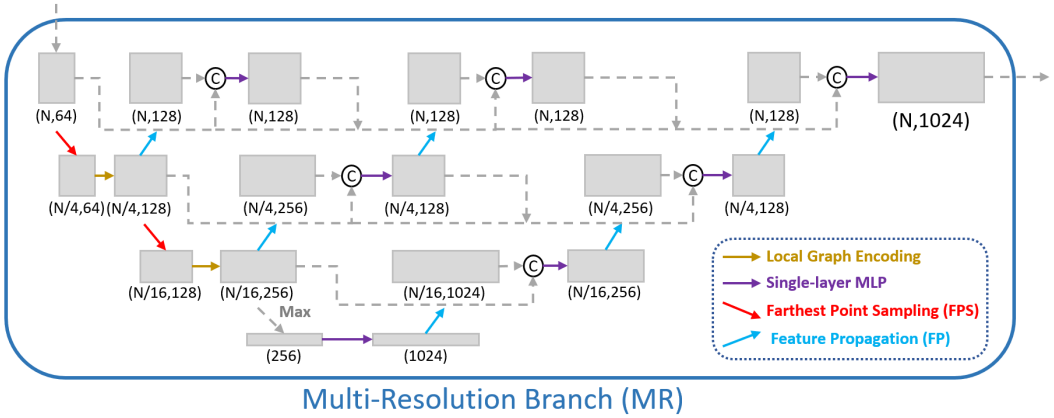

Figure 1. The input of the MR branch is the output of the first error-minimizing module in the FR branch, while the output of the MR branch merges with the output of the FR branch following the behavior as Equation 4 in the main paper.

method	model size (MB)	time (ms)	overall acc. (%)
PointNet [2]	40	16.6	89.2
PointNet++ [3]	12	163.2	90.7
PCNN [1]	94	117.0	92.3
DGCNN [4]	21	27.2	92.9
Ours	70	19.2*	**93.1**

Table 1. Complexity of classification network on *ModelNet40*. (*running on GeForce GTX 2080Ti)

weights for other modules' losses can be smaller ($w_2 = w_3 = w_4 = 0.01$). Although the additional losses are incorporated, the cross-entropy loss still contributes the most to the training. The overall loss $\mathcal{L}$ in our practice is formulated as:

$$\mathcal{L} = \mathcal{L}_{ce} + 0.1 \cdot \mathcal{L}_{er_1} + 0.01 \cdot \mathcal{L}_{er_2} + 0.01 \cdot \mathcal{L}_{er_3} + 0.01 \cdot \mathcal{L}_{er_4}.$$

3. Multi-resolution Branch

In general, the MR branch is implemented with light-weight operations such as single-layer MLPs, and only investigates 2 more resolutions of the point cloud using basic Local Graph Encoding as Equation 2 in the main paper. For upsampling and downsampling operations, they are implemented based on CUDA without learnable weights. Besides, we use the dense connections and concatenations to enhance the relations between the feature maps of different resolutions.

4. Model Complexity

In addition, we adopt the network complexity data provided in [4] for a fair comparison. As Table 1 shows, our model size is relatively large due to the parameters and operations needed. However, the inference time of our method running on a single GeForce GTX 2080Ti GPU is only *19.2 ms*, which indicates the ability of our model in forward procedure thanks to the algorithm optimization and relevant CUDA implementation.

References

[1] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018.

[2] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[3] 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.

[4] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):146, 2019.