APP-Net: Auxiliary-point-based Push and Pull Operations for Efficient Point Cloud Classification

Tao Lu, Chunxu Liu, Youxin Chen, Gangshan Wu *Member, IEEE*, Limin Wang *Member, IEEE*

***Abstract*—Aggregating neighbor features is essential for point cloud classification. In the existing work, each point in the cloud may inevitably be selected as the neighbors of multiple aggregation centers, as all centers will gather neighbor features from the whole point cloud independently. Thus each point has to participate in the calculation repeatedly and generates redundant duplicates in the memory, leading to intensive computation costs and memory consumption. Meanwhile, to pursue higher accuracy, previous methods often rely on a complex local aggregator to extract fine geometric representation, which further slows down the classification pipeline. To address these issues, we propose a new local aggregator of linear complexity for point cloud classification, coined as APP. Specifically, we introduce an auxiliary container as an anchor to exchange features between the source point and the aggregating center. Each source point pushes its feature to only one auxiliary container, and each center point pulls features from only one auxiliary container. This avoids the re-computation issue of each source point. To facilitate the learning of the local structure of cloud point, we use an online normal estimation module to provide the explainable geometric information to enhance our APP modeling capability. Our built network is more efficient than all the previous baselines with a clear margin while still consuming a lower memory. Experiments on both synthetic and real datasets demonstrate that APP-Net reaches comparable accuracies to other networks. It can process more than 10,000 samples per second with less than 10GB of memory on a single GPU. We will release the code in https://github.com/MCG-NJU/APP-Net.**

***Index Terms*—3D Shape Classification, Local Aggregator, Efficient.**

I. Introduction

With the growing demand for 3D applications, how to classify 3D objects with neural networks has become an important topic in recent years. Extensive work has been devoted to obtaining a higher accuracy for this task. Based on the data type and the employed networks, existing methods can be grouped into two categories. The first one is the multi-view-based 2D solution [1]–[3] which projects the 3D object into 2D image planes from multiple views and then applies the well-designed 2D convolutional neural network [4]–[6] to learn cross-view consistent representations. These methods focus on the selection of informative views and cooperation across different views. The second solution directly learns from the 3D data with point-based networks [7], [8]. They focus on how to integrate the spatial relation into feature aggregation process. Several kinds of delicate local aggregators, like the point-wise MLP style and the position adaptive weighting style, are proposed to extract the fine geometric structure.

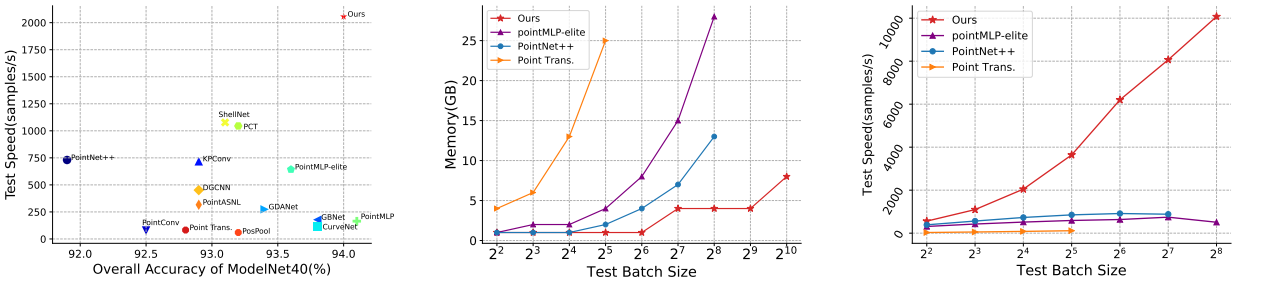
Thanks to the previous efforts, more and more works hit an accuracy of over 93% in the most popular classification dataset of ModelNet40 [9], in the last three years. In fact, the benchmark performance has shown saturation for a long time. The detailed analysis in CloserLook3D [10] points out that, under fair comparison, the performance gap among different local aggregators can be bridged by unified network architecture and fair training process. This conclusion reminds us to think about whether it is necessary to simply pursue a higher accuracy when designing point cloud network. Instead, we argue that, in practice, running speed and memory consumption are also important factors that should be taken into account.

The essential factors responsible for the total overhead are the amount of computation and the degree of parallelism. Specifically, in the multi-view based methods, the computation and memory consumption are both linearly growing with the number of the views due to each view is processed independently. Such solutions may take several times intensive overhead to obtain slight improvement by introducing more views. For the point-based methods, due to the lack of neighbor index in the irregular point cloud, each point takes extra efforts to query and gather neighbors during the learning process. As analyzed in PVCNN [11], the pure point-based methods are slowed down heavily by the random memory access since it corrupts the parallelism heavily by the bank conflicts. So they propose to gather neighbor features efficiently with a voxel branch by benefiting from the memory locality.

In this paper, we propose a computation and memory-efficient point-based solution to 3D classification based on the following three observations: (1) If a point is queried as multiple points' neighbors, it has to participate in computations repeatedly and occupies serveral times footprints in memory, which leads to redundant computations and memory consumption. (2) Previous architectures, except for the PointNet [7], are all designed to accomplish feature aggregation and receptive field expansion simultaneously through the overlapped neighboring area for different center points. The points in the overlapped area are inevitably queried more than one time. (3) Due to the natural sparsity in the point cloud, the extra effort on the neighbor query is unavoidable. Even for the voxel-aided neighbor gathering, the dense voxelization manner still wastes extra resources on a large amount of blank voxels. The sparse manner also suffers from memory bank conflicts in the scatter and gather process. According to Figure I, although the kNN algorithm costs too much time, its 1-NN variant shows great efficiency surpassing all the other

T. Lu, C. Liu, G. Wu, L. Wang are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, China.

Y. Chen is with the Samsung Electronics (China) R&D Centre, Nanjing, 210012, China.



(a) Accuracy-Speed (b) (Batch Size)-Memory (c) (Batch Size)-Speed

Fig. 1. Quantitative Comparisons. (a) shows the accuracy-speed tradeoff on ModelNet40. We achieve the highest efficiency while maintaining comparable accuracy. (b) and (c) is the GPU memory consumption and inference speed under different inference batch sizes, respectively. In both of them, we outperform other methods with a large margin. All the experiments in (b) and (c) use 1024 points, (a) uses the default number of points in corresponding papers with a batch size=16.

methods. Based on the first two observations, we conclude that one key towards an efficient point-based classification network is *to decouple the process of feature aggregation and receptive field expansion*. In addition, the third observation suggests that for the classification task (often uses 1024∼4096 points), the 1-NN algorithm is efficient enough to query neighbors.

Based on the above analysis, we propose a new network whose overall computation is reduced to linear complexity, and it only costs linear memory occupations. Specifically, we aggregate features and expand the receptive field in separate steps. During aggregation, to avoid repeated operations to each point, we introduce an auxiliary container as a proxy to exchange features among points. The container consists of a series of pre-specified anchors in 3D space. Then each point is processed by two operations: first, push its feature to only one nearest auxiliary anchor; second, pull the feature from only one nearest auxiliary anchor. According to the corresponding anchor, the point cloud is split into several non-overlapped blocks. Points closest to the same auxiliary anchor in Euclidean space will fall into the same block and accomplish the features exchange automatically. Each anchor only costs a tiny maintenance overhead. To avoid the artifact introduced by the auxiliary anchor, we propose a novel push and pull pair through which the influence from the anchor is reducible. To enable receptive field expansion, we introduce a second auxiliary container to produce a different partition of the whole point cloud. Combining the two-stage block partitions, we obtain an equivalent receptive field expansion. Finally, to facilitate learning the local structure in the early layer, we use an online normal estimation module to provide explainable geometric information to enhance our APP block's modeling capability.

The auxiliary-anchor-based push and pull operations pair, the so-called APP operator, achieve a huge reduction in the memory consumption and computation complexity while maintaining comparable accuracy to the previous methods. The comparisons among different styles of network structure, including PointNet [7], PointNet++ [8], point cloud transformer [12], and the proposed APP-based network, are depicted in Fig 3. It is easy to see that the overhead of the APP is linear to the number of input points. We conducted a detailed

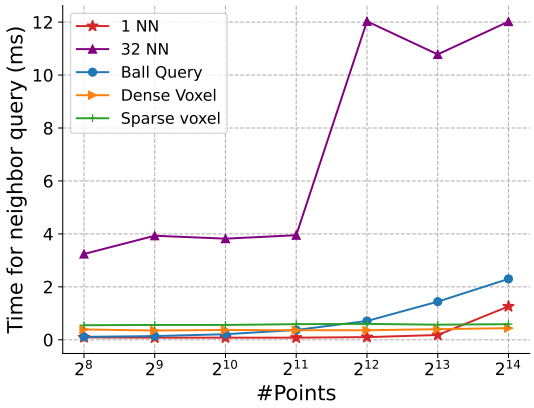
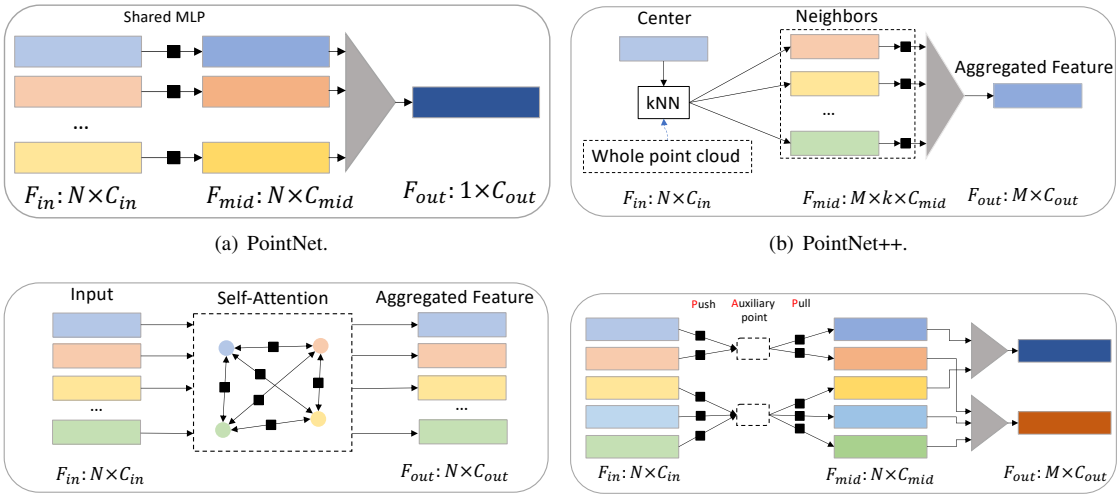


Fig. 2. Duration for Neighbor Query. Thanks to the parallelism, the 1-NN algorithm is efficient enough to process small scale point cloud.

quantitative analysis of the running speed and memory consumption. As depicted in Fig 1, we achieve an inference speed of more than 2000 samples/second with batch size=16. Among those networks outperforming 92%, we are clearly faster than the second efficient network, ShellNet [13]. Moreover, our network consumes a remarkably low GPU memory. According to Fig 1, APP-Net only costs memories less than 10GB with a batch size=1024. Correspondingly, our machine's maximum supported batch size for Point Transformer [14] is 64, which costs more than 30GB of memory. And the lightweight version of PointMLP [15] consumes more than 25GB with a maximum batch size of 256. Furthermore, according to Fig 1, we can even achieve a speed more than **10,000 samples/s** with a batch size of 256, which is $\mathbf{5}\times$ faster than the peak of other baselines. More details and analysis are presented in the following sections. In summary, the main contributions of this paper are:

1) We propose to decouple the feature aggregation and receptive field expansion process to facilitate redundancy reduction.
2) We propose an auxiliary-anchor-based operator to exchange features among neighbor points with linear computation complexity and linear memory consumption.



(a) PointNet. (b) PointNet++.

(c) Point Cloud Transformer. (d) APP-Net.

Fig. 3. Comparisons among Point Cloud Networks, including the PointNet, PointNet++, Point Cloud Transformer, and our proposed APP-Net. The **black boxes** in each method refer to the learnable parts.

3) We propose to use the online normal estimation to improve the classification task.
4) Experiments show that the proposed network achieves remarkable efficiency and low memory consumption while keeping competitive accuracy.

II. Related Work

In this section, we review some remarkable works in 3D object classification. According to the data type, we divide those methods into multi-view-based and point-cloud-based methods.

A. Multi-view Based Methods

Multi-view-based methods consider learning point features with the mature CNN by projecting the 3D object into a series of 2D images from multiple views. Some works [1], [2], [16]–[20] devoted to investigating how to fuse the features with pooling policy. Yang et al. [21] propose to combine with the guidance of the inter-relationship among views. MVTN [22] learns to search a series of more informative viewpoints. Wei et al. [3] treats the different views as nodes in a graph, which facilitates the use of GCN to exploit the relation among views. Carlos et al. [23] proposes to use rotation equivariant convolution on multi-views.Liu et al. [24] introduces a more complex setting of fine-grained classification and proposes to explicitly detect the regions of interest.

In general, the projection process is view-dependent. It requires processing many views to alleviate the geometrical information loss caused by the projection, making it intensive to analyze each 3D object. And it's challenging to fuse the view-wise features from different views to obtain a consistent and discriminative representation.

B. Point Based Methods

This is a widely used category in 3D classification. Motivated by the 2D CNN [25], [26], many works are designed to aggregate local point features. The local area is determined by the distance in Euclidean space or the topology. DGCNN [27] constructs a graph to learn with the connections between nodes. PointNet++ [8] provides a standard paradigm for fully point-based method. It points out how to locate the neighbor area and aggregate local features point-wisely. Subsequent works [13], [28] improve the designs of local area structure, downsampling methods, and local aggregator. Different from PointNet++ [8], some works design convolution-like operators for point cloud. SpiderNet [29] uses Taylor expansion to approximate the spatial distribution of filters. KPConv [30] specifies a series of kernel points to implement convolution. PointConv [31] directly learns the values of the filter through coordinates. And PAConv [32] proposes to assemble the pre-specified weight banks according to the spatial relation. The recent hot topic, transformer, has started to show its power in the point cloud. PCT [12] is the first totally transformer-based network which conducts a global self-attention in each layer. It cuts down the process to query neighbors because each point serves as all the other points' neighbors. Point Transformer [14] enhances the local aggregator with a Transformer-like operator. Point2SpatialCapsule [33] proposes to not only model the geometric information but also model the spatial relation with the Capsule Network [34]. L3DOC [35] introduces lifelong learning to extending the 3D classification task into open environments. DSDAN [36] investigates the problem of cross-domain 3D classification. These methods have achieved remarkable accuracy. However, efficiency and low memory consumption are not their main targets. Some of the few attempts for efficiency explore by eliminating the existing architectures. ShellNet [13] proposes to shrink some heavy operations the PointNet++ to construct a lightweight architecture, PointMLP-Elite leverages the bottleneck architecture to reduce the feature transforming burden. Although they have shown effectiveness in accelerating, they do not solve the problem of redundant resource calls. So they leave us with a huge room for eliminating the overhead. We achieve

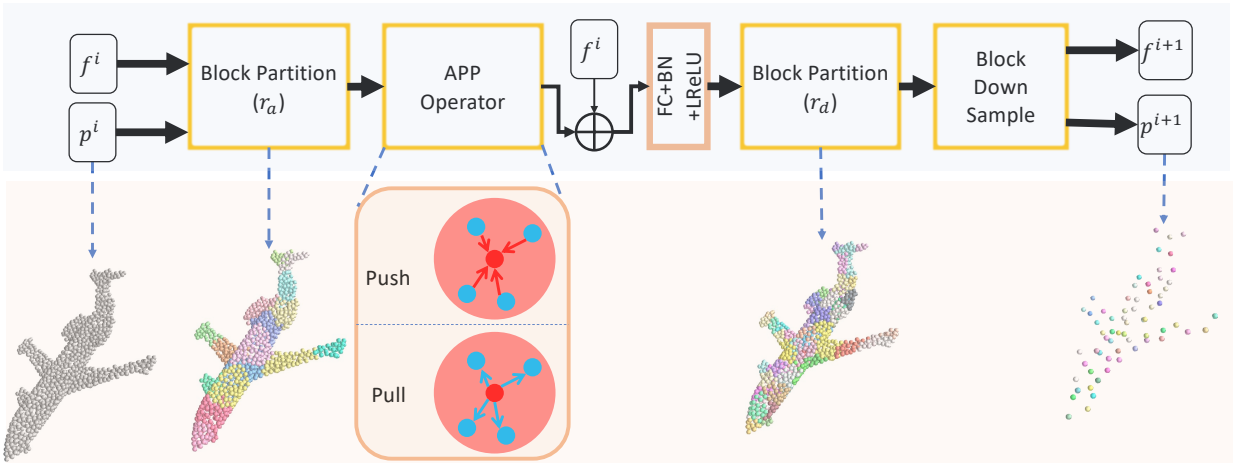


Fig. 4. The APP Block Structure. The inputs are coordinates with features, and the output is downsampled coordinates and updated features.

the most efficient network for point cloud classification with the proposed APP operator.

III. Method

A. Background and Insights

1) Preliminaries: The source point cloud with N points is denoted as $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_N\}$. The target is to aggregate current source features into a set of center points $\mathbf{Q} = \{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_M\}$. The general process of a local point feature aggregation is as follows:

$$\mathbf{g}_i = R(\{G(r(\mathbf{q}_i, \mathbf{p}_j, [\mathbf{f}_i, \mathbf{f}_j]), \mathbf{f}_j) | \mathbf{p}_j \in N(\mathbf{q}_i)\}) \quad (1)$$

where $\mathbf{q}_i$ is the center point, $\mathbf{g}_i$ is the output feature for the center point. $N(\mathbf{q}_i)$ queries neighbors for $\mathbf{q}_i$. $\mathbf{f}_* \in \mathbb{R}^{C_{in}}$ denotes the input feature of $\mathbf{p}_*$ or $\mathbf{q}_*$. $r(\mathbf{q}_i, \mathbf{p}_j, [\mathbf{f}_i, \mathbf{f}_j])$ measures the relation between the neighbor point and the center point. Most methods mainly consider the position relation, and others combine the feature relation. G and R refer to the features transformation and reduction, respectively. The reduction function is MAX Pooling in most cases.

2) Analysis and Insights: For the common point-based architecture [8], each center point selects neighbors from the source points independently. Each source point would be selected as multiple centers' neighbors to accomplish the receptive field expansion. Thus, each source point has to repeatedly participate in the aggregation process and occupies several duplicates of memories to facilitate parallelization. The advantage of this manner is obvious: it enables the neighbor point to adaptively contribute to different center points according to the current spatial and feature relation, and it combines the feature extraction and receptive field expansion in one step. However, it inevitably induces extra computations and memory consumption for redundant operations. In the first layer of PointNet++ [8], $N = 1024$, $M = 512$, each center point queries 32 neighbors, which indicates that each source point would be replicated and re-computed for $\frac{512\times32}{1024}$ =16 times.

To avoid using redundant resources, we notice that all the previous methods conduct the aggregation procedure from the view of the center point $\mathbf{Q}$, and each center point is agnostic of how many times other center points have accessed the neighbor. To limit the access times of each source point, we turn to guide the aggregation from the view of the source point. In this paper, we proposed a so-called APP (auxiliary-based push and pull) operator which introduces the auxiliary point set $\mathbf{A}$ as a bridge between the source points and the center points. Each source point $\mathbf{p}$ only pushes its information to one auxiliary point, and each center point $\mathbf{q}$ only pulls information from one auxiliary point, i.e., during the whole feature aggregation process, each source point only participates in emitting feature for one time. And the center point only participates in gathering features for one time. Such paired operations linearize the complexity of computation. Although reducing some overhead, auxiliary points may also introduce incorrect dependencies on the artifacts. To eliminate the influence of auxiliary points, we implement the operation pair in a novel manner, with which the influence of auxiliary points is reducible. The core idea is to find a group of functions $\{\alpha, \beta, \gamma, \epsilon\}$ which satisfy the following relation:

$$\gamma(x \to y) = \epsilon[\alpha(x \to a), \beta(a \to y)] \quad (2)$$

where $\alpha(x \to a)$ and $\beta(a \to y)$ denote the process of pushing features from x to the auxiliary point a and pulling features from a to destination point y respectively. ϵ combines $\alpha(*)$ and $\beta(*)$ in a reducible way such that the resulted $\gamma(x \to y)$ only depends on the x and y, not affected by the auxiliary point. In the following sections, we will introduce some instantiations of the proposed $\{\alpha, \beta, \gamma, \epsilon\}$.

B. Auxiliary-based Push and Pull Operations

1) Auxiliary Point Generation: As introduced above, the auxiliary points serve as bridges to pass information among local points. For simplicity, we directly downsample the original point cloud to obtain a subset $\mathbf{A} = \{\mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{\frac{N}{r_a}}\}$, where the r_a is a downsample ratio. Following RandLA-Net [37],

we adopt the Random Sample instead of the widely-used FPS (Farthest Point Sample) [8] for further acceleration. Although the uniformity is slightly disturbed, the experimental results show little sensitivity to the sampling strategy. As mentioned in Section III-A that each source point only emits its information to one auxiliary point, we conduct a 1-NN query for every source point from the auxiliary point set $\mathbf{A}$. For a point $\mathbf{p}_i$, we denote its auxiliary point as $A(\mathbf{p}_i)$. Points close to each other in Euclidean space naturally choose the same auxiliary point. As a result, the whole point cloud is partitioned into several non-overlapped blocks $\{B(\mathbf{a}_0), B(\mathbf{a}_1), ..., B(\mathbf{a}_{\frac{N}{r_a}})\}$, where $B(\mathbf{a}_i)$ denotes the source point set whose 1-NN auxiliary point is $\mathbf{a}_i$.

2) The Design of Operation Groups: According to the previous analysis, a reducible operation group conforming to Equation 2 eliminates the auxiliary point's influence to prevent artifacts. Given two points x and y belonging to the same block, we first instantiate the $\epsilon(*)$ with two optional basic operators: element-wise multiplication and element-wise addition. The Equation 2 is rewritten as follows,

$$\gamma(x \to y) = \alpha(x \to a) \cdot \beta(a \to y), \tag{3}$$

or

$$\gamma(x \to y) = \alpha(x \to a) + \beta(a \to y). \tag{4}$$

For simplicity, we will use α to denote $\alpha(x \to a)$ and β to denote $\beta(a \to y)$ in the following. Then we construct satisfying α and β for the two operators. For the addition operator, it's obviously that if α and β are linear mappings, i.e.

$$\begin{aligned}\gamma(x-y) &= W \times (x-y), \\ &= \alpha + \beta, where \begin{cases}\alpha = W \times (x-a), \\ \beta = W \times (a-y),\end{cases}\end{aligned} \tag{5}$$

where W is a weight matrix shared by α and β, the influence from the auxiliary point is easily eliminated, and the resulted γ is also a linear mapping function. Following a similar idea, we construct the α and β with exponential mapping for the multiplication operator. Specifically,

$$\begin{aligned}\gamma(x-y) &= e^{W \times (x-y)}, \\ &= \alpha \cdot \beta, where \begin{cases}\alpha = e^{W \times (x-a)}, \\ \beta = e^{W \times (a-y)}.\end{cases}\end{aligned} \tag{6}$$

The above designs are based on a single operator (multiplication or addition). When we jointly employ multiple basic operators for η, more sophisticated operation groups can be derived. According to the "sum-difference-product" formula for trigonometric functions, we obtain sin and cos based operator groups as follows,

$$\begin{aligned}&\gamma(x-y) = cos(W \times (x-y)) \\ &= \alpha[0]\beta[0] - \alpha[1]\beta[1], where \begin{cases}\alpha[0] = cos(W \times (x-a)) \\ \alpha[1] = sin(W \times (x-a)) \\ \beta[0] = cos(W \times (a-y)) \\ \beta[1] = sin(W \times (a-y))\end{cases}\end{aligned} \tag{7}$$

$$\begin{aligned}&\gamma(x-y) = sin(W \times (x-y)), \\ &= \alpha[0]\beta[1] + \alpha[1]\beta[0], where \begin{cases}\alpha[0] = sin(W \times (x-a)), \\ \alpha[1] = cos(W \times (x-a)), \\ \beta[0] = sin(W \times (a-y)), \\ \beta[1] = cos(W \times (a-y)).\end{cases}\end{aligned} \tag{8}$$

We believe that there are infinite operation groups satisfying the reducible philosophy. And we have no intention of exhausting those potential superior combinations. We adopt the above operation groups to form the basis of this work. They decompose the inter-communication process within the local area. For a local patch with n points, to pass information between every point-pair, previous methods like [38] induce $\mathcal{O}(n^2)$ overhead. While with the proposed operation groups, all points share the same $\alpha(x-a)$ to obtain the information of x. The number of all possible $\alpha(*)$ is n. Thus we obtain a $\mathcal{O}(n)$ complexity overhead. Furthermore, due to some mathematical properties (parity, reciprocal relation, opposite relation, and so on) of the basic operators, some parts of the operations can be reused by other parts, which also contributes a lot to lowering the consumption of computation and memory resources. We summarize the reusable parts as follows,

- **Exponential**-based: $\beta = \frac{1}{\alpha}$;
- **Cosine**-based: $\beta[0] = \alpha[0]$, $\beta[1] = -\alpha[1]$;
- **Sine**-based: $\beta[0] = -\alpha[0]$, $\beta[1] = \alpha[1]$.

C. Push and Pull based Feature Aggregation

One key advantage of the point cloud is the preservation of geometry. Previous local aggregators mine the 3D structure information through modeling the spatial relation among points. According to how to mine the spatial relation, they are classified into different categories, like the point-wise MLP-based or adaptive-weight-based methods. For the point-wise MLP type, the Equation 1 is usually instantiated as follows,

$$\mathbf{g}_i = R(\{MLP(\mathbf{q}_i - \mathbf{p}_j, \mathbf{f}_j) | \mathbf{p}_j \in N(\mathbf{q}_i)\}). \tag{9}$$

The adaptive-weight-based methods generate position adaptive kernel to weight neighbors. The Equation 1 is instantiated as follows,

$$\mathbf{g}_i = R(\{\mathcal{W}(\mathbf{q}_i - \mathbf{p}_j) \cdot \mathbf{f}_j | \mathbf{p}_j \in N(\mathbf{q}_i)\}) \tag{10}$$

where $\mathcal{W}(*)$ generates the kernel weight according to the spatial relation.

In this part, we show examples of how to build the two types local aggregators based on the above operation groups. Only parts of the proposed operation groups will be exhibited, and the others can be implemented by following similar steps or referring to our released code. The process consists of Position Encoding, Push-step, Pull-step, Channel Mixing, and Block-Based Down Sample.

-Position Encoding At the beginning of every setting, we implement a simple and fast position encoding to lift the original coordinate from a 3-channel vector to an embedding

of C_{in} channels. The global sharing encoding function $\phi(*)$ is implemented by [FC Layer + BatchNorm + Leaky ReLU]. We use the position encoding, instead of the original coordinate, to cope with the features in all the following steps.

-Push & Pull In the Push-step, the information of the source point is delivered to the auxiliary point. And in the Pull-step, each point gathers features from the corresponding auxiliary point. We introduce how to build a point-wise MLP-based and adaptive-weight-based local aggregator as examples.

1) **Point-wise MLP** According to Equation 9, a point-wise MLP style aggregator requires concatenating features and spatial relation. Thus the push step is defined by

$$\mathbf{g}_{\mathbf{p}_i \to A(\mathbf{p}_i)} = W \times [\mathbf{f}_i, \phi(\mathbf{p}_i) - \phi(A(\mathbf{p}_i))], \tag{11}$$

where the $W \in \mathbb{R}^{2C_{in} \times C_{in}}$ is a weight matrix corresponding to the one in Equation 5. This step pushes the source point's feature to the auxiliary point according to the spatial relation. Mind that all the source points belonging to the same auxiliary block will push their feature to the same auxiliary point. Thus the final feature in the auxiliary point is computed by

$$\mathbf{g}_{A(\mathbf{p}_i)} = \frac{1}{|B(A(\mathbf{p}_i))|} \sum_{\substack{\mathbf{p}_j \in \\ B(A(\mathbf{p}_i))}} W \times [\mathbf{f}_j, \phi(\mathbf{p}_j) - \phi(A(\mathbf{p}_i))]. \tag{12}$$

In the pull step, following Equation 5, we apply an inverse operation for each source point to pull features from the corresponding auxiliary point. The inverse operation is formulated as follows:

$$\begin{aligned} \mathbf{g}_i &= \mathbf{g}_{A(\mathbf{p}_i) \to \mathbf{p}_i}, \\ &= \mathbf{g}_{A(\mathbf{p}_i)} + W \times [-\mathbf{f}_i, \phi(A(\mathbf{p}_i)) - \phi(\mathbf{p}_i)] \\ &= \frac{1}{|B(A(\mathbf{p}_i))|} \sum_{\substack{\mathbf{p}_j \in \\ B(A(\mathbf{p}_i))}} W \times [\mathbf{f}_j - \mathbf{f}_i, \phi(\mathbf{p}_j) - \phi(\mathbf{p}_i)]. \end{aligned} \tag{13}$$

According to the output, the resulted $\mathbf{g}_i$ is only computed by the point feature and spatial relation. The auxiliary point $A(\mathbf{p}_i)$ only serves to provide a neighbor query. Different from the common practice, we adopt AVG Pooling as the reduction function. One more difference is, for leveraging the reusable computation, Equation 13 finally concatenates the feature difference and spatial relation. To realise the concatenation of original feature and spatial relation, one can modify the β operation by replacing the feature with zeros vector, i.e. $\beta = W \times [\mathbf{0}, \phi(A(\mathbf{p}_i)) - \phi(\mathbf{p}_i)]$. And the β operation cannot reuse the results of α. We will discuss its influence in the experiment section.

2) **Adaptive Weight** Following similar steps, we can easily construct an adaptive weight aggregator. Here we use the **Exponential**-based operation groups. The push step is defined by

$$\mathbf{g}_{\mathbf{p}_i \to A(\mathbf{p}_i)} = \mathbf{f}_i \cdot e^{W \times [\phi(\mathbf{p}_i) - \phi(A(\mathbf{p}_i))]}, \tag{14}$$

where $W \in \mathbb{R}^{C_{in} \times C_{in}}$ generates weight kernel according to the spatial relation. The $e^{(*)}$ provides channel-wise weights to the input feature. The features in $A(p_i)$ is

Algorithm 1 Exponential-based adaptive weight aggregator

```
# points: [N, 3], F: [N, C]
# r_a, r_d

# Block Partition
aux_points = rand_choice(points, N/r_a)
                                # [N/r_a, 3]
idx_PtoA = one_nn(points, aux_points)
                                # [N, 1]
# push and pull
pos_enc = Linear_BN_LReLU(points) # [N, C]
kernel = exp(Linear(pos_enc)) # [N, C]
F_weighted = F * kernel # [N,C]
F_PtoA = scatter_mean(F_weighted, idx_PtoA)
                                # [N/r_a, C]
F_AtoP = gather(F_PtoA, idx_PtoA) # [N, C]
aggregated_F = F_AtoP / kernel # [N,C]
#Channel Mixing
new_F = Linear_BN_LReLU([F, new_F])
                              # [N,C_out]
# Block Based Down Sample
centroids = rand_choice(points, N/r_d)
                                # [N/r_d, 3]
idx_PtoC = one_nn(points, centroids)
                                # [N, 1]
out_F = scatter_max(new_F, idx_P2C)
                        # [N/r_d,C_out]

return out_F, centroids
```

$$\mathbf{g}_{A(\mathbf{p}_i)} = \frac{1}{|B(A(\mathbf{p}_i))|} \sum_{\substack{\mathbf{p}_j \in \\ B(A(\mathbf{p}_i))}} \mathbf{f}_j \cdot e^{W \times [\phi(\mathbf{p}_j) - \phi(A(\mathbf{p}_i))]} \tag{15}$$

Then the pull step is formulated as

$$\begin{aligned} \mathbf{g}_i &= \mathbf{g}_{A(\mathbf{p}_i) \to \mathbf{p}_i} \\ &= \mathbf{g}_{A(\mathbf{p}_i)} \cdot e^{W \times [\phi(A(\mathbf{p}_i)) - \phi(\mathbf{p}_i)]} \\ &= \frac{1}{|B(A(\mathbf{p}_i))|} \sum_{\substack{\mathbf{p}_j \in \\ B(A(\mathbf{p}_i))}} \mathbf{f}_j \cdot e^{W \times [\phi(\mathbf{p}_j) - \phi(\mathbf{p}_i)]}. \end{aligned} \tag{16}$$

Essentially speaking, the final output is an instantiation of Formula 10, whose reduction function is AVG Pooling. It weights each channel according to the spatial relation with neighbors.

-Channel Mixing The push and pull steps are efficient operations for mixing the features among points in the local area. However, there exist two obstacles towards a better representation: first, the use of AVG Pooling tends to obscure some high-frequency patterns in each local area. Then, some push and pull operations, like the **Exponential**-based groups, only conduct channel-wise weighting without inter-channel interaction, which damages the model capacity. Thus we employ a skip connection from the input features to make up for the high-frequency information. And we introduce a Fully Connection layer to enhance channel mixing. The feature is updated by

$$\mathbf{g}_i = \delta([\mathbf{g}_i, \mathbf{f}_i]). \tag{17}$$

where $\delta(*)$ is a non-linear function constituted by {FC+BatchNorm+LeakyReLU}. The resulted $\mathbf{g}_i$ is of C_{out} channels.

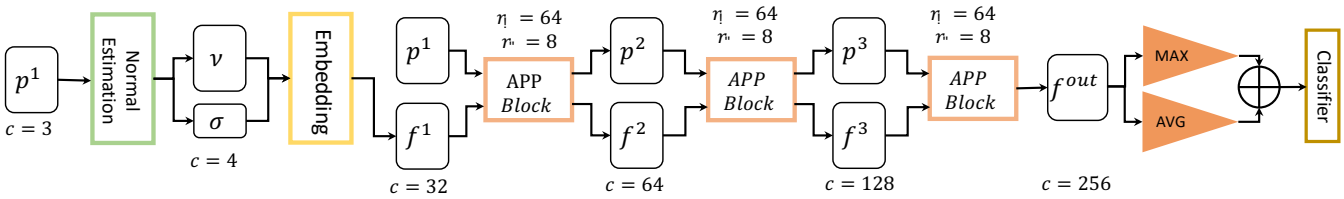


Fig. 5. The whole structure of the proposed APP-Net. The channel of normal estimation is 3 or 4, which corresponds to whether to use the curvature.

-Block Based Downsampling After processing by Push-step and Pull-step, the output point cloud still has the same number of points as the input point cloud, which is similar to the PointNet layer [7]. We design a block-based down sample strategy to reduce the intermediate overhead further. Like the operation in Section III-B1, we downsample the point cloud with a rate of $\mathbf{r}_d$ and re-split the whole cloud into serval non-overlap blocks $\{\mathbf{D}_0, \mathbf{D}_1, ..., \mathbf{D}_{\frac{N}{\mathbf{r}_d}}\}$ based on 1-NN algorithm. Then, for all the points belonging to the same block $\mathbf{D}_i$, their features are aggregated by a MAX pooling function as follows:

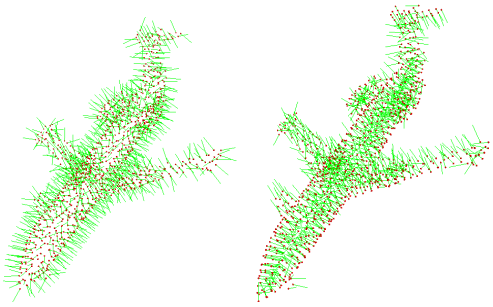
$$\mathbf{g}_{\mathbf{d}_i} = MAX\{\mathbf{f}_j | \mathbf{p}_j \in \mathbf{D}_i\}. \tag{18}$$

Then the aggregated features are registered to the corresponding block centroids. A python-style pseudo-code for an Exponential-based adaptive weight aggregator is shown in Algorithm 1.

-The Full Structure of APP-Net This part introduces how to build an end-to-end network with the APP operator. As shown in Figure 5, the input is the information of the point, including the position xyz, the normal and local curvature. An embedding layer consisting of [Linear($C_{in} \rightarrow 32$), BatchNorm, LeakyReLU] lifts the input normal (and curvature) to an embedding with a dimension of 32. Three layers of the APP block are cascaded to aggregate neighbor features for every point. The APP operators and the block down sample are configured with the corresponding auxiliary rate $r_a = [64, 64, 64]$ and downsample rate $r_d = [8, 8, 8]$, respectively. A pooling layer (concatenates the results of AVG Pooling and MAX Pooling) outputs the global feature after the last APP layer. According to the global feature, a classifier consisting of MLPs predicts the probability for each category.

D. Local Geometry from Online Normal and Curvature Estimation

The normal and curvature reflect the local surface property. Previous methods often use the coordinates as input features and design a delicate and heavy local extractor to model the local geometric structure. In this work, we turn to directly feed the network with the explicit geometric descriptor, i.e., normal and curvature, to simplify the process of modeling geometric structure. Then the network can be more concentrated on learning the semantic relation. In point cloud, normal estimation is approximated by the problem of estimating the normal of a plane tangent to the surface [39]. In this paper, we adopt the simplest PCA-based method. A brief revision is present here to make the text more self-contained. For the centroid point $\mathbf{p}$, computing its local covariance matrix by



(a) Ground Truth Normal. (b) Estimated Normal.

Fig. 6. Different types of normals.

$$C = \frac{1}{|N(\mathbf{p})|} \sum_{\mathbf{p}_i \in N(\mathbf{p})} (\mathbf{p}_i - \mathbf{p}) \times (\mathbf{p}_i - \mathbf{p})^T, \tag{19}$$

$$C \cdot \vec{v_j} = \lambda_j \cdot \vec{v_j}, j \in \{0, 1, 2\} \tag{20}$$

$\vec{v}_*$ and λ_* represent the eigenvectors and eigenvalues of the covariance matrix, respectively. The eigenvector $\vec{v}$ corresponding to the minimum eigenvalue is the estimated normal. Supposed that λ_0 is the minimum eigenvalue, then the curvature σ of the local surface is determined by

$$\sigma = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{21}$$

A direction consistency check will flip those normal who do not orient towards a pre-specified viewpoint $(0, 0, 0)$ to alleviate the ambiguity in the normal direction. The comparison among different normals is depicted in Fig 6.

E. Discussion and Analysis on the APP-Net

1) Receptive Field Analysis: The auxiliary point-based feature aggregation process includes two non-overlapped partitions, one for point mixing in the push step and pull step and the other for downsampling points. It's a common practice that large and expandable receptive fields are critical to learning good representation. As illustrated in Fig 7, the receptive field expands rapidly by combining the two non-overlapped partitions. A difference from the previous methods is that the expanded receptive field is irregular and random. Although introducing some uncertainties to the local descriptor, the random receptive field does not damage the global descriptor. And the global descriptor is more crucial to the classification task.

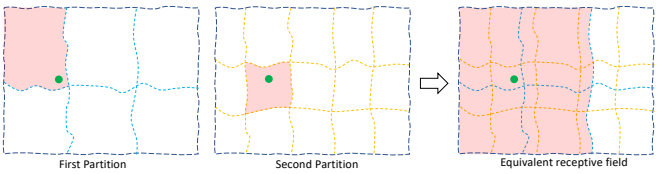


Fig. 7. Two-stage Block Partition. The first stage is in the push and pull step. The second stage is used for the block down sample. The whole point cloud are randomly partitioned into several non-overlapped blocks with different partition ratios. Thus the effect of the two-stage block partition is equivalent to the overlapped blocks.

2) Relation to Prior Work: **Relation to PointNet** Both PointNet [7] and the APP-Net's complexity are $\mathcal{O}(n)$ in each layer. However, the PointNet layer lacks point mixing, while the APP-Net introduces point mixing with the push and pull step, enabling information passing among points. Moreover, PointNet lacks a downsampling operation. Thus its high layers suffer intensive computations. In the APP-Net, the block-based down sample makes the high layers lightweight.

Relation to PointNet++ and its follow-up PointNet++ and its follow-ups gather features from a sphere area around the center point, while the APP-Net gathers features from a random and irregular area. Another crucial difference is that PointNet++ imitates the convolution operation to accomplish receptive field expansion in one step, while the APP-Net does it in two stages. And the two-step style facilitates the linearization of feature extraction. One more difference is that we use the normal and curvature as the input to the network, while most of the previous work use the coordinates as the input.

Relation to Transformer-like work Both the local Transformer block and the APP block model the pair-wise relations in the local area. The self-attention in Transformer constructs a $N \times N$ relation matrix. However, in the APP block, we decompose the $N \times N$ relation matrix into a $N \times 1$ matrix and a $1 \times N$ matrix, corresponding to the push step and pull step, respectively. This implies that the APP block is a linear version of the Transformer block and models a low-rank relation matrix.

Relation to Conv-Deconv architecture The APP block is similar to the Conv-Deconv operation pair, where the push step convolves the input from N points into M points, and the pull step deconvolves the M points back into N points. However, they are completely different designs. The convolution and deconvolution are independent steps and do not share the learnable parameters. And the intermediate M points are nonnegligible. However, in the APP block, the push and pull steps are highly coupled. They reuse the same learnable parameters, and the influence from the intermediate M points is reducible.

3) Complexity Analysis: We compare the overhead of the proposed auxiliary-based method with the point-wise-MLP layer. Given the input $\{P \in \mathbb{R}^{N\times 3}, F \in \mathbb{R}^{N\times C_{in}}\}$, we want to obtain a output of $\{P' \in \mathbb{R}^{M\times 3}, F' \in \mathbb{R}^{M\times C_{out}}\}$. Table I and Table II show the computation complexity and memory consumption for the APP block and a single scale single MLP PointNet++ block, respectively. K denotes the number of nearest neighbors. The dominating computation is $2 * N \times C_{in} \times C_{out}$ for APP block, $M \times K \times C_{in} \times C_{out}$ for the PointNet++ block. Due to $M \times K \gg 2 \times N$ in practice, the APP block owns lower computations. In the stable implementation of PointNet++ [40], $M \times K = 16 \times N$, which induces $8\times$ computations. As to the memory, the dominating part is $3 \times N \times C_{in} + N \times C_{out}$ for APP block, $M \times K \times C_{in} + M \times K \times C_{out}$ for PointNet++. Similarly, the APP block owns several times lower memory consumption. Meanwhile, the above analyses are based on the single scale and single MLP settings. If employing the commonly used multiple scales or multiple MLP for the PointNet++, the overhead advantages for APP block would be more obvious.

TABLE I
APP BLOCK OVERHEAD ANALYSIS.

Step	Computation	Memory
Position Encoding	$N \times 3 \times C_{in}$	$N \times C_{in}$
Push	$N \times C_{in}$ $\frac{N}{r_a} \times C_{in}$	$N \times C_{in}$ $\frac{N}{r_a} \times C_{in}$
Pull	$N \times C_{in}$	$N \times C_{in}$
Channel Mixing	$N \times C_{in} \times C_{out}$	$N \times C_{out}$
Block Pool	-	$M \times C_{out}$

TABLE II
POINTNET++ BLOCK OVERHEAD ANALYSIS.

Step	Computation	Memory
Group	-	$M \times K \times (C_{in} + 3)$
MLP	$M \times K \times (C_{in} + 3) \times C_{out}$	$M \times K \times C_{out}$
Pooling	-	$M \times C_{out}$

IV. EXPERIMENTS

To explore the characteristics of our method, we conduct extensive experiments on a synthetic dataset and a real scene dataset. Our method has achieved competent accuracy while maintaining very low overhead and high efficiency compared with existing methods. We further conduct some ablation studies on Section IV-D to test how every module works.

A. Settings

Our experiments are evaluated on a server with one Tesla V100 GPU. Most of the projects are implemented with PyTorch [41]. For all the tasks, we use the default Adam [42] to update the network parameters. And we use the cosine learning rate scheduler [43] to update the learning rate, with an initial learning rate of 2e-3. The minimum learning rate threshold is set to 2e-4. The cycle for the cosine scheduler is $T_{max} = 200$. For all the experiments, we train the network for 300 epochs with a training batch size of 32 and a test batch size of 16; we use the first epoch to warm up the training process. All the results of the comparison methods are obtained in three ways: 1. careful reproduction in our environment to prevent the unfairness caused by machine development (For fairness, if we fail to reproduce the public results, we will adopt the following two ways); 2. the reported results in the original papers; 3. the updated results on the public websites or other published papers.

TABLE III
CLASSIFICATION ON MODELNET40. WE REPORT THE OVERALL ACCURACY, TRAIN SPEED, TEST SPEED, AND THE NUMBER OF PARAMETERS OF SOME BASELINES. 5K DENOTES 4096 POINTS, AND THE 7K FOR KPCONV MEANS USING AROUND 7,000 POINTS. 'P' AND 'N' MEANS USING POINT AND GROUND TRUTH NORMAL, RESPECTIVELY. **Bold** NUMBER DENOTES THE BEST ONE.

Model	Inputs	Train Speed (samples/s)	Test Speed (samples/s)	Param.	OA(%)
PointNet [7]	1k P	960.9	1422.4	3.5M	89.2
Pointnet++ [8]	1k	352.2	730.2	1.41M	90.7
PointNet++ [8]	5k P+N	-	-	1.41M	91.9
PointCNN [44]	1k P	-	-	-	92.5
PointConv [31]	1k P+N	104.5	76.4	18.6M	92.5
KPConv [30]	7k P	211.7	717.7	15.2M	92.9
DGCNN [27]	1k P	-	-	-	92.9
RS-CNN [45]	1k P	-	-	-	92.9
DensePoint [46]	1k P	-	-	-	92.8
ShellNet [13]	1k P	551.3	1077.5	**0.48M**	93.1
PointASNL [28]	1k P+N	285.9	316.4	3.2M	93.2
PosPool [10]	5k P	51.0	59.5	18.5M	93.2
Point Trans. [47]	1k P	-	-	-	92.8
GBNet [48]	1k P	17.7	175.9	8.39M	93.8
GDANet [49]	1k P	29.8	273.3	0.93M	93.4
PA-DGC [32]	1k P	-	-	-	93.6
MLMSPT [50]	1k P	-	-	-	92.9
PCT [12]	1k P	115.7	1044.9	2.88M	93.2
Point Trans. [14]	1k P	67.1	82.3	12.9M	93.7
CurveNet [51]	1k P	89.9	112.9	2.04M	93.8
PointMLP [15]	1k P	60.4	169.0	12.6M	**94.1**
PointMLP-elite [15]	1k P	240.1	632.8	0.68M	93.6
APP-Net(Exp+AW)	5k P	785.9	1451.8	0.77M	93.0
APP-Net(Exp+AW)	5k P+N	**1440.6**	**2155.5**	0.77M	94.0
APP-Net(Cos+AW)	5k P+N	1174.1	1971.2	0.79M	93.5
APP-Net(Cos+PW)	5k P+N	1274.2	2107.6	0.77M	93.4
APP-Net(Sin+AW)	5k P+N	1224.3	1995.1	0.79M	93.2
APP-Net(Sin+PW)	5k P+N	1264.3	2095.1	0.77M	93.4

TABLE IV
TIME COST ANALYSIS FOR EACH MODULE OF THE APP-NET.

	Normal Estimation	Feature Embedding	Layer 1	Layer 2	Layer 3	Classifier
APP-Net	$2ms$	$0.13ms$	$1.2ms$	$1.4ms$	$1.1ms$	$0.07ms$
PointNet++ [8]	-	-	$20.01ms$	$12.8ms$	$0.88ms$	$0.27ms$

For simplicity, we denote each variant of APP-Net with the combination of "basic operator+aggregator type" in the following part. The basic operators contain {Exp, Sin, Cos} and the aggregator type contains {AW, PW}. 'AW' means to use the adaptive weight style, and 'PW' denotes the point-wise MLP style.

B. Shape Classification on ModelNet40

ModelNet40 [9] is the most influential benchmark for 3D classification task, consisting of 40 common categories. The point-cloud-type data is synthesized by randomly sampling the surface of 3D CAD models, with a training set of 9,843 samples and a testing set of 2,468 samples. We report the most widely used metric, Overall Accuracy, on the testing set. For fairness, we do not adopt the voting strategy for all the methods(which often improves the accuracy by about 0.4% for some methods). Besides, we also report the computation overhead. As shown in Table III, we achieve comparable accuracy to these SOTAs and maintain a very efficient running speed. The speed is measured by $\frac{Total\ Samples}{Total\ Inference\ Time}$. Most of the baseline methods obtain a faster speed than the one reported in the previously published papers. We believe it is mainly attributed to the machine difference and the adoption of the optimized CUDA operations. Using 1024 points is a standard setting for this task. There are also some methods choosing to input more (4096 or more) points to boost the result at the price of a heavier burden. For ModelNet40, the proposed APP-Net is fed with 4096 points while still running faster than all the other baselines of 1024 points. And it's $3\times$ faster than the PointNet++ [8], $19\times$ faster than CurveNet [51] during test, which is coherent with the analysis in section III-E3. APP-Net has a clear speed advantage over the other methods even with the online estimated normal.

C. Shape Classification on ScanObjectNN

Considering the saturation of ModelNet40 [9] and challenging real-world cases, Uy et al. propose the ScanObjectNN [52], collecting by scanning the indoor objects. The real-world data often face some annoying issues, like the cluttered or occluded by fore/background. So ScanObjectNN reveals the great potential to promote the classification application in the real world.

TABLE V
CLASSIFICATION ON SCANOBJECTNN. THE INPUT FOR APP-NET CONTAINS 1024 POINTS. WE RUN THE EXPERIMENT FIVE TIMES AND REPORT THE MEAN±STD. **Bold** NUMBER DENOTES THE BEST ONE. *‡ USES MORE LEARNABLE PARAMETERS (STILL LESS THAN MOST OF THE BASELINES).

Methods	Inputs	OA(%)	Train Speed (samples/s)	Test Speed (samples/s)
PointNet [7]	Point	68.2	960.9	1422.4
SpiderCNN [53]	Point	73.7	-	-
PointNet++ [8]	Point	77.9	352.2	730.2
DGCNN [27]	Point	78.1	-	-
PointCNN [44]	Point	78.5	-	-
BGA-DGCNN [52]	Point	79.7	-	-
BGA-PN++ [52]	Point	80.2	-	-
DRNet [54]	Point	80.3	-	-
GBNet [48]	Point	80.5	-	-
SimpleView [55]	Multi-view	80.5 ± 0.3	-	-
PRANet [56]	Point	82.1	-	-
MVTN [22]	Multi-view	82.8	-	-
PointMLP [15]	Point	85.4 ± 0.3	60.4	169.0
PointMLP-elite [15]	Point	83.8 ± 0.6	240.1	632.8
APP-Net(Exp+AW)	Point	84.3 ± 0.3	**1633.0**	2442.0
APP-Net(Cos+AW)	Point	84.2 ± 0.1	1377.3	2343.1
APP-Net(Cos+PW)	Point	84.4 ± 0.2	1405.4	2395.7
APP-Net(Sin+AW)	Point	84.7 ± 0.1	1394.2	2387.7
APP-Net(Sin+PW)	Point	84.4 ± 0.1	1359.4	**2471.7**
APP-Net(Cos+AW)‡	Point	86.1 ± 0.2	1239.7	2069.9
APP-Net(Sin+AW)‡	Point	$\mathbf{87.0 \pm 0.2}$	1153.9	2023.9

According to the common practice in other work, we use the hardest variant PB_T50_RS to conduct our experiments. The whole dataset contains a training set with 11416 samples and a testing set with 2882 belonging to 15 categories. We choose the most representative point-based and multi-view methods as the baselines. The normal we put into the network is computed online, and the duration of normal estimation is considered in the speed test. The overall accuracy is shown in Table V. Following SimpleView [55], we report the mean±std. We outperform all the baseline methods, including the PointMLP [15]. The lightweight "Exp+AW" version is $14\times$ faster than PointMLP [15] and $3.8\times$ faster than the lightweight PointMLP-elite. With more parameters, the APP-Net achieves the best accuracy among all the methods. And the large version is still faster than all the other methods. In Table IV, we report the time cost of each module under test mode, with batch size=16 and N =1024. Each layer only costs $1.1 \sim 1.4$ms.

D. Ablation Studies

There are some fine-designed structures in APP. To test their functionality and substitutability, we conducted some ablation studies and analyses based on the ScanObjectNN.

The Advantages of the Reducible Operation One of the core designs of the APP is the reducible operation pair: Push-Step and Pull-step. They make each point's representation independent of the auxiliary point. To verify the effectiveness, we design two types of non-reducible operations: 1. use two different mapping functions to compute the spatial relation for the Push-step and Pull-step, respectively; 2. add non-linear operation, i.e., BatchNorm and Leaky ReLU, to the spatial relation. The results in Table VI show that the reducible operation is non-trivially superior to the others in accuracy.

TABLE VI
THE REDUCIBLE OPERATION. WE EXPLORE WHETHER THE REDUCIBLE ATTRIBUTE IS NECESSARY AND A NEW INSTANTIATION OF THE REDUCIBLE PHILOSOPHY. THE BEST ONE IS COLORED WITH **Bold**. TESTED WITH THE "EXP+AW" AGGREGATOR.

	OA(%)
Reducible	**84.3**
Not Reducible [Different Mapping Function]	82.8
Not Reducible [Non-linear Mapping Function]	83.2

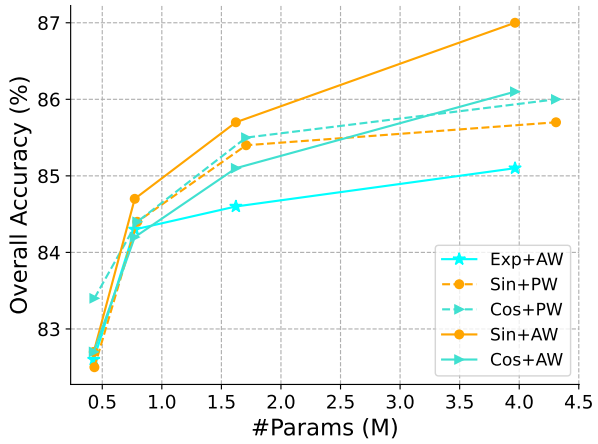


Fig. 8. Performance gains with the growing model size.

The Input to the Network Unlike previous work, which takes the original coordinates as the input, we use the estimated normal and curvature as explicit descriptors to represent the local geometry. Table VII shows the comparisons among different inputs to different configured networks. The results clearly indicate that the estimated normal and curvature significantly

TABLE VII
DIFFERENT INPUT TO THE NETWORK.

	Sin+AW	Cos+AW	Sin+PW	Cos+PW
xyz	76.1	78.2	79.8	79.4
$normal$	83.8	83.0	83.3	83.7
$normal$+ $curvature$	**84.7**	**84.2**	**84.4**	**84.4**

TABLE VIII
COMBINATIONS OF THE APP LAYER AND POINTNET++ LAYER. 'A' DENOTES USING APP LAYER AND 'P' DENOTES USING POINTNET++ LAYER. TESTED WITH THE "EXP+AW" AGGREGATOR.

Layer 0	Layer 1	Layer 2	OA (%)	Speed (samples/s)	Memory (MB)
A	A	A	**84.3**	2442.0	1393
A	A	P	83.7	2058.6	3361
A	P	P	83.7	1762.4	5535
P	P	P	84.1	1353.3	5535

TABLE IX
DOWN SAMPLE METHODS. 'RS' DENOTES RANDOM SAMPLE AND 'FPS' DENOTES FARTHEST POINT SAMPLE.

Configs	OA(%)		Speed (samples/s)	
	Exp+AW	Sin+AW	Exp+AW	Sin+AW
RS	**84.3**	**84.7**	**2442.0**	**2387.7**
FPS	83.8	84.1	2320.4	2235.2

TABLE X
POINTWISE MLP STYLE ANALYSIS

Pulls	OA(%)		Speed (samples/s)	
	Sin +PW	Cos +PW	Sin +PW	Cos +PW
$W \times [f_i, \phi(A(p_i) - \phi(p_i))]$	84.4	84.4	**2471.7**	**2395.7**
$W \times [0, \phi(A(p_i)) - \phi(p_i)]$	**84.7**	**85.0**	2143.2	1992.0

TABLE XI
DIFFERENT WAYS TO MODEL THE RELATIONS AMONG NEIGHBORS. TESTED WITH THE "EXP+AW" AGGREGATOR.

Configs	OA(%)
Global Position Encoding $\phi(\mathbf{p}_i)$	**84.3**
Local Position Encoding $\phi(\mathbf{p}_i - A(\mathbf{p}_i))$	82.8
No Position Encoding $\mathbf{p}_i$	82.5
Feat+Global Position Encoding $\mathbf{f}_i + \phi(\mathbf{p}_i)$	83.6
Concat[Feat, Global Position] Encoding $[\mathbf{f}_i, \phi(\mathbf{p}_i)]$	82.1

improve the performance. Meanwhile, when fed with the xyz, the APP-Net achieves PointNet++'s [8] and DGCNN's [27] performance with a clear overhead advantage.

Model Scalability In this part, we scale up the network by enlarging the number of feature channels to test the scalability of the APP-Net. According to Figure 8, the performance grows along with the number of channels. The Sin-based and Cos-based networks outperform the Exponential Function; we think its due to the value range of the Exponential Function being unbounded and growing rapidly, thus making it hard to optimize. In the point-wise manner, the Cos-based method is marginally better than the Sin-based method, while in the adaptive weighting variant, the relation are reversed.

Co-operation with other operators According to Table IV, the first layer in PointNet++ occupies a large ratio of time. We layer-wisely replace the APP layer in our network with the PointNet++ layer to explore the effect of combining different layers. According to Table VIII, the overall accuracy of every combination is comparable, while replacing the PointNet++ layer with the APP layer greatly accelerates the network and reduces memory consumption. Meanwhile, the results of the 3 layers PointNet++ is more effective and efficient than the one in Table V. We believe it is because we implement it with the single MLP layer and the input of the network is the normal and curvature rather than the coordinates. The identical memory consumption in "A+P+P" and "P+P+P" settings are caused by the internal memory allocation mechanism of PyTorch.

Farthest Point Sample versus Random Sample We use a random sample in the blockdown sample module. According to Table IX, although the farthest point sample produces a more uniform subset of the input point cloud, it does not lead to better accuracy. And it is slightly slower than the random sample in our experiments.

Whether to use feature difference in the Point-wise MLP aggregator To eliminate the calculation for the point-wise MLP style aggregator, we reuse the concatenation of feature and position encoding for the push and pull steps. This induces the aggregator to model the feature difference. To explore the effect of modeling the feature difference and the original feature, we decouple the calculation of the push and pull steps. According to Table X, the modeling of the original feature marginally hits a better accuracy. However, it slows down the network. Taking the results in Table V into account, the speed around 2000 samples/s can be also met through enlarging the channels, and the enlarged version achieved even better accuracy. To keep the architecture simple, we model the feature difference as the standard version.

Different Relation Modeling Methods We compare different ways of encoding points' relations in this part. The variants include using the global position encoding, local position encoding, or directly computing the spatial relation without using position encoding. The local position is computed by subtracting the corresponding center point. Considering that the transformer measures the relation between features, we also try to combine the feature relation in the APP. Results in Table XI show that the global position encoding obtains better results than the local one. We think it is due to the instability of the local position caused by the random block partition, which hinders the network convergence during training. Besides, the feature relation does not provide a positive effect. We guess it wrongly builds the dependencies between the position and feature. This also explains why the "Feat+Global Position Encoding" performs better than "Concat[Feat, Global Position] Encoding" since the former decouples the process of learning feature relation and spatial relation.

Feature Updating Style At the end of the pull step, every

TABLE XII
DIFFERENT WAYS TO UPDATE THE FEATURE AFTER THE PULL-STEP. TESTED WITH THE "EXP+AW" AGGREGATOR.

	Concat $\mathbf{g}_i = \delta([\mathbf{g}_i, \mathbf{f}_i])$	Not Concat $\mathbf{g}_i = \delta(\mathbf{g}_i)$	Res Feature $\mathbf{g}_i = \delta(\mathbf{g}_i) + \mathbf{f}_i$	Identity $\mathbf{g}_i = \mathbf{g}_i$
OA (%)	**84.3**	77.8	83.7	79.3

TABLE XIII
THE INFLUENCE OF THE NETWORK DEPTH. WITH DIFFERENT LAYERS, THE RATE WILL BE ADJUSTED ADAPTIVELY.

Configs	OA(%)
2 layers, r_d=[8,8], r_a=[64,64]	83.0
3 layers, r_d=[8,8,8], r_a=[64,64,64]	**84.3**
4 layers, r_d=[4,4,8,8], r_a=[64,64,16,16]	83.9

TABLE XIV
DIFFERENT RATE CONFIGURATIONS FOR THE APP-NET.

Rate	OA(%)
r_d=[8,8,8],r_a=[64,64,64]	**84.3**
r_d=[4,4,4],r_a=[64,64,64]	84.1
r_d=[16,16,16],r_a=[64,64,64]	83.5
r_d=[8,8,8],r_a=[32,32,32]	82.6
r_d=[8,8,8],r_a=[32,64,64]	83.7
r_d=[8,8,8],r_a=[64,64,32]	83.1

TABLE XV
DIFFERENT POOLING POLICIES.

Configs	OA(%)
AVG+MAX Pooling	**84.3**
MAX Pooling	82.1
AVG Pooling	79.7
Position Adaptive Pooling	79.9

point's feature is updated by concatenating the output with the original feature and sending it to a $\delta(*)$ function. Among the comparisons, we try to remove some parts of it or leverage a residual structure. As shown in Table XII, The concatenation manner is superior to the other configurations. Moreover, in the last two rows, we remove the $\delta(*)$ function or the whole APP block; the results clearly state their indispensability.

Network Depth We explore how the network depth affects the performance. In the 2-layer and 3-layer version, we adopt the same r_a and r_d. In the 4-layer version, due to the original number of the point being 1024, after two downsample operations, the remaining points are insufficient to support a large r_a, so we adopt a smaller $r_a = 16$ in the last two layers. According to Table XIII, the 3-layer version achieves the best performance.

Down Sample Rate The rate r_a and r_d serve as the network's kernel size. They control the receptive field for each center point. For simplicity, we adopt the same r_d for all layers in each variant. Results in Table XIV imply that a large r_a for the auxiliary point generation is critical to a better representation, especially at the high level (according to the last two rows). And the result is relatively not sensitive to the rate r_d for aggregation.

Pooling Policies in the Aggregation We have tested with the common pooling policies to explore a proper aggregation operation for the second block partition. The position-adaptive pooling aggregates local points weighted by the reciprocal distance in Euclidean space. The results in Table XV show that combining the local mean context and the most representative feature can achieve better performance for the classification task.

E. Limitations

We note the following limitations of this work:

1) The proposed operator is designed for the classification task. However, this paper does not explore its generalization to other dense estimation tasks, like segmentation. Although it does not harm the global descriptor, the random receptive field may introduce noise to the local descriptor. So the generalization to other tasks is challenging.
2) In the synthetic dataset, the ground truth normal is necessary for the APP-Net to classify some hard cases (confusing with similar categories). This is because the geometry information is too dependent on the estimated noisy normal and curvature. A better estimation method or a lightweight geometric learning layer may alleviate it.

V. CONCLUSIONS AND FUTURE WORK

This paper proposes a novel APP operator aggregating local features through auxiliary points. It avoids redundant memory consumption and re-computation of the source point. Furthermore, the auxiliary points' influence is reducible, allowing the method to preserve more details. Experiments on the synthetic and real-scene datasets show a good balance between performance, speed, and memory consumption in the classification task. Especially in speed, it outperforms all the previous methods significantly. Our future goal is to extend this method to more tasks, like semantic segmentation and object detection.

REFERENCES

[1] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[2] T. Yu, J. Meng, and J. Yuan, "Multi-view harmonized bilinear network for 3d object recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 186–194.

[3] X. Wei, R. Yu, and J. Sun, "View-gcn: View-based graph convolutional network for 3d shape analysis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1850–1859.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] C. R. Qi, H. Su, K. Mo, and L. 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*, 2017, pp. 652–660.

[8] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[9] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," *2015 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR)*, pp. 1912–1920, 2015.

[10] Z. Liu, H. Hu, Y. Cao, Z. Zhang, and X. Tong, "A closer look at local aggregation operators in point cloud analysis," in *European Conference on Computer Vision*. Springer, 2020, pp. 326–342.

[11] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," *arXiv preprint arXiv:1907.03739*, 2019.

[12] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, "Pct: Point cloud transformer," *Computational Visual Media*, vol. 7, no. 2, pp. 187–199, 2021.

[13] Z. Zhang, B.-S. Hua, and S.-K. Yeung, "Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1607–1616.

[14] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16 259–16 268.

[15] X. Ma, C. Qin, H. You, H. Ran, and Y. Fu, "Rethinking network design and local geometry in point cloud: A simple residual MLP framework," in *International Conference on Learning Representations*, 2022.

[16] T. Yu, J. Meng, and J. Yuan, "Multi-view harmonized bilinear network for 3d object recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 186–194.

[17] Z. Yang and L. Wang, "Learning relationships for multi-view 3d object recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7505–7514.

[18] K. Sun, J. Zhang, J. Liu, R. Yu, and Z. Song, "Drcnn: Dynamic routing convolutional neural network for multi-view 3d object recognition," *IEEE Transactions on Image Processing*, vol. 30, pp. 868–877, 2020.

[19] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, "Gvcnn: Group-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 264–272.

[20] S. Chen, L. Zheng, Y. Zhang, Z. Sun, and K. Xu, "Veram: View-enhanced recurrent attention model for 3d shape classification," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 12, pp. 3244–3257, 2018.

[21] Z. Yang and L. Wang, "Learning relationships for multi-view 3d object recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7505–7514.

[22] A. Hamdi, S. Giancola, and B. Ghanem, "Mvtn: Multi-view transformation network for 3d shape recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1–11.

[23] C. Esteves, Y. Xu, C. Allen-Blanchette, and K. Daniilidis, "Equivariant multi-view networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1568–1577.

[24] X. Liu, Z. Han, Y. Liu, and M. Zwicker, "Fine-grained 3d shape classification with hierarchical part-view attention," *IEEE Trans. Image Process.*, vol. 30, pp. 1744–1758, 2021. [Online]. Available: https://doi.org/10.1109/TIP.2020.3048623

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[26] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[27] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3693–3702.

[28] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui, "Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5589–5598.

[29] X. Gu, K. Nahrstedt, and B. Yu, "Spidernet: An integrated peer-to-peer service composition framework," in *Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 2004.* IEEE, 2004, pp. 110–119.

[30] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6411–6420.

[31] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9621–9630.

[32] M. Xu, R. Ding, H. Zhao, and X. Qi, "Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3173–3182.

[33] X. Wen, Z. Han, X. Liu, and Y.-S. Liu, "Point2spatialcapsule: Aggregating features and spatial relationships of local regions on point clouds using spatial-aware capsules," *IEEE Transactions on Image Processing*, vol. 29, pp. 8855–8869, 2020.

[34] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *Advances in neural information processing systems*, vol. 30, 2017.

[35] Y. Liu, Y. Cong, G. Sun, T. Zhang, J. Dong, and H. Liu, "L3doc: Lifelong 3d object classification," *IEEE Transactions on Image Processing*, vol. 30, pp. 7486–7498, 2021.

[36] F. Wang, W. Li, and D. Xu, "Cross-dataset point cloud recognition using deep-shallow domain adaptation network," *IEEE Transactions on Image Processing*, vol. 30, pp. 7364–7377, 2021.

[37] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "Randla-net: Efficient semantic segmentation of large-scale point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 108–11 117.

[38] H. Zhao, L. Jiang, C.-W. Fu, and J. Jia, "Pointweb: Enhancing local neighborhood features for point cloud processing," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5565–5573.

[39] "Estimating surface normals in a pointcloud," https://pcl.readthedocs.io/projects/tutorials/en/latest/normal_estimation.html?highlight=normal%20estimation Accessed July 31, 2022.

[40] E. Wijmans, "Pointnet2 pytorch," https://github.com/erikwijmans/Pointnet2_PyTorch Accessed July 31, 2022.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[43] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[44] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," *Advances in neural information processing systems*, vol. 31, pp. 820–830, 2018.

[45] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8895–8904.

[46] Y. Liu, B. Fan, G. Meng, J. Lu, S. Xiang, and C. Pan, "Densepoint: Learning densely contextual representation for efficient point cloud processing," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5239–5248.

[47] N. Engel, V. Belagiannis, and K. Dietmayer, "Point transformer," *IEEE Access*, vol. 9, pp. 134 826–134 840, 2021.

[48] S. Qiu, S. Anwar, and N. Barnes, "Geometric back-projection network for point cloud classification," *IEEE Transactions on Multimedia*, 2021.

[49] M. Xu, J. Zhang, Z. Zhou, M. Xu, X. Qi, and Y. Qiao, "Learning geometry-disentangled representation for complementary understanding of 3d object point cloud," *arXiv preprint arXiv:2012.10921*, vol. 2, 2021.

[50] X.-F. Han, Y.-J. Kuang, and G.-Q. Xiao, "Point cloud learning with transformer," *arXiv preprint arXiv:2104.13636*, 2021.

[51] A. Muzahid, W. Wan, F. Sohel, L. Wu, and L. Hou, "Curvenet: Curvature-based multitask learning deep networks for 3d object recognition," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 6, pp. 1177–1187, 2020.

[52] M. A. Uy, Q.-H. Pham, B.-S. Hua, T. Nguyen, and S.-K. Yeung, "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1588–1597.

[53] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "Spidercnn: Deep learning on point sets with parameterized convolutional filters," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 87–102.

[54] S. Qiu, S. Anwar, and N. Barnes, "Dense-resolution network for point cloud classification and segmentation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 3813–3822.

[55] A. Goyal, H. Law, B. Liu, A. Newell, and J. Deng, "Revisiting point cloud shape classification with a simple and effective baseline," in *International Conference on Machine Learning*. PMLR, 2021, pp. 3809–3820.

[56] S. Cheng, X. Chen, X. He, Z. Liu, and X. Bai, "Pra-net: Point relation-aware network for 3d point cloud analysis," *IEEE Transactions on Image Processing*, vol. 30, pp. 4436–4448, 2021.