Surface Representation for Point Clouds

Supplementary Material

1. Preliminaries: Taylor Series for 2D curves

Talyor series [4] on the point $(a, f(a))$ of curve $f(\cdot)$ presents as follows:

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots, \tag{1}$$

which can be simplified as:

$$\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n, \tag{2}$$

where $f^{(n)}(a)$ is the n-th derivative of the curve $f(\cdot)$ at the point $(a, f(a))$.

We present the assumption that the formulation of Taylor Series can depict the local curve. Based on this assumption, we further develop an extension to 3D space.

2. Preliminaries: Two-Variate Taylor Series for 3D surfaces

Taylor Series depending on two variables can be defined as:

$$g(a,b)+\frac{1}{1!}(x-a,y-b)\cdot\begin{pmatrix}\frac{\partial g}{\partial x}(a,b)\\ \frac{\partial g}{\partial y}(a,b)\end{pmatrix}+\cdots, \tag{3}$$

where $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$ are the partial derivatives. This formulation presents two-variant taylor series on point $(a, b, g(a, b))$ of surface $g(\cdot, \cdot)$.

This formulation reveals the basis of RepSurf. To simply the calculation, we consider the terms of the first and second partial derivatives. Triangular RepSurf can be an instantiation.

3. Details of Polar Auxiliary

We present two types of polar auxiliary, spherical and cylindrical ones based on Spherical Polar System and Cylindrical Polar System, respectively.

For a given point (x, y, z), spherical polar auxiliary provides the corresponding polar coordinate $(\rho_s, \theta_s, \phi_s)$, where $\rho_s = \sqrt{x^2+y^2+z^2} \in [0, +\infty)$, $\theta_s = \arccos\frac{z}{\rho} \in [0, \pi]$, $\phi_s = \text{atan2}(y, x) \in [0, 2\pi)$. For stable training, we normalize the polar coordinate by θ_s divided by π and ϕ_s divided by 2π. Though ρ_s has no upper bound in theory, ρ_s is commonly limited within $[0, r]$, where r is the radius of ball query function [3]. Furthermore, to prevent the generation of NaN, we set θ_s to 0 when ρ_s is 0. The pseudo-code of spherical polar auxiliary is presented in Algorithm 1.

Accordingly, cylindrical polar auxiliary from (x, y, z) gives the polar coordinate (ρ_c, θ_c, z_c), where $\rho_s = \sqrt{x^2+y^2} \in [0, r]$, $\phi_s = \text{atan2}(y, x) \in (-\pi, \pi)$, $z_c = z \in [-r, r]$, r is the given radius of ball query function [3]. Similarly, we normalize ϕ_s and z_c into the range of $[0, 1]$. We implement polar auxiliary by concatenation of the Cartesian coordinate (x, y, z) and $(\rho_s, \theta_s, \phi_s)$ or (ρ_c, θ_c, z_c). The pseudo-code of cylindrical polar auxiliary in Algorithm 2.

Though extremely simple, our design of polar auxiliary is not an incremental method and can be insightful. Polar auxiliary is mainly relied upon the prerequisite that the models learn the local shapes within the queried balls. This prerequisite allows spherical polar coordinate to work with Cartesian coordinate more reasonably. We argue that a Cartesian coordinate is efficient to represent the location of a point numerically according to the origin or the centroid. However, it cannot obviously discriminate the locations of two neighbors. When the two points are very close, Cartesian coordinates show few clues to tell both. In this case, θ_s and ϕ_s can intuitively magnify the difference between the two points numerically. Furthermore, ρ_s is an additional ingredient to express the relationship between a neighbor point and its centroid. Both empirical results and theoretical analysis prove the effectiveness of our design of polar auxiliary.

4. Details of Channel De-differentiation

We propose channel de-differentiation to handle the obvious distribution imbalance between the maaped coordinates and the mapped last-stage features in each stage of set abstraction (SA) in a PointNet++ [3] model. An illustration is shown in Fig. 1. This may lead to an ignorance of the input of coordinates in the last few layers of MLPs. We consider this is mainly caused by the difference of the distributions of various types of input (like coordinates and

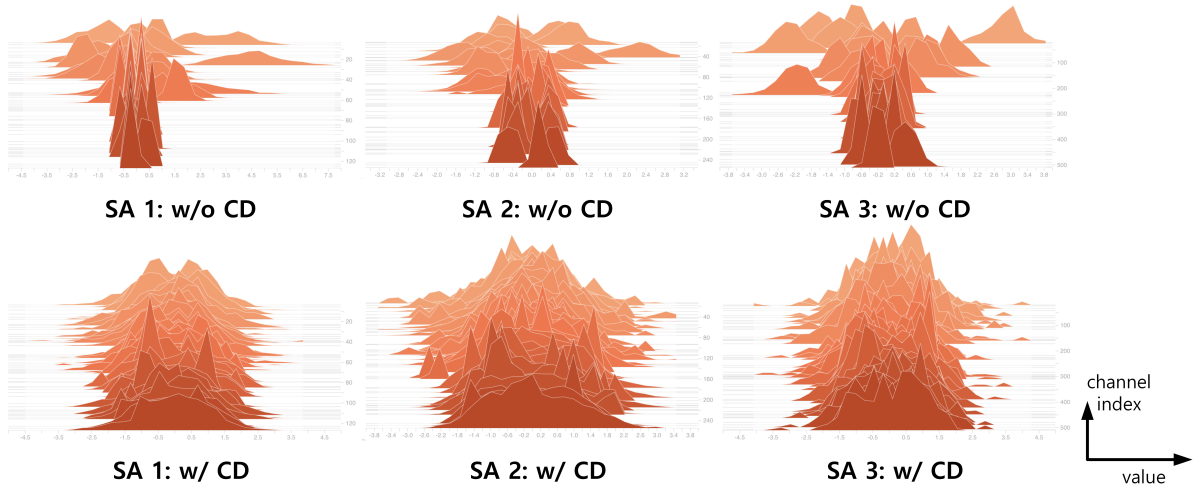

Figure 1. An example of the distributions of the mapped cooridnates (second-half channels, e.g., 64∼128 for the left images) and the mapped features (first-half channels, e.g., 0∼64 for the left images) before element-wise summation during matrix multiplication in the first layer of each stage. For an obvious comparison, we put these two modalities together in each plot, which does not mean that we perform concatenation in our CD. Note that, for the first layer of each stage, PointNet++ w/o CD performs BN **after** the summation of the mapped coordinates and features (the status like the above three images), while PointNet++ w/ CD performs BN **before** the summation (the status like the below three images). The problem of distribution imbalance will weaken the importance of one of the two kinds of input, and CD can alleviate this problem in a simple manner.

Algorithm 1 Pytorch-Style Pseudocode of Spherical Polar Auxiliary

```
# xyz: coordinates of a point set
rho = sqrt(sum(pow(xyz,2),dim=-1,keepdim=True))
rho = clamp(rho,min=0) # range: [0, inf]
theta = acos(xyz[...,2,None]/rho) # range: [0, pi]
phi = atan2(xyz[..., 1,None], xyz[..., 0,None]) #
    range: [-pi, pi]

# check nan
idx = rho==0
theta[idx] = 0

# normalize
theta = theta/pi # [0, 1]
phi = phi/(2*np.pi)+.5 # [0, 1]
out = torch.cat([rho,theta,phi],dim=-1)
return out
```

Algorithm 2 Pytorch-Style Pseudocode of Cylindrical Polar Auxiliary

```
# xyz: coordinates of a point set
rho = sqrt(sum(pow(xyz[...,:2],2),dim=-1,keepdim=
    True))
rho = clamp(rho,0,1) # range: [0, 1]
phi = atan2(xyz[...,1,None], xyz[...,0,None]) #
    range: [-pi, pi]
z = xyz[...,2,None]
z = torch.clamp(z,-1,1) # range: [-1, 1]

# normalize
phi = phi/(2*pi)+.5
z = (z+1.)/2.
out = torch.cat([rho,phi,z],dim=-1)
return out
```

high-level features).

Intuitively, we adopt batch normalization to alleviate the difference of these distributions. In the first MLP of each SA, the fused feature $\mathbf{f}_i^1$ of the i-th point can be rewrite as:

$$\mathbf{f}_i^1 = \omega^1([\mathbf{x}_i, \mathbf{f}_i]) = \omega_x^1(\mathbf{x}_i) + \omega_f^1(\mathbf{f}_i), \tag{4}$$

where ω^1 is a linear function, the concatenation of the weights of ω_x^1 and ω_f^1 equals to the weights of ω^1. $\mathbf{x}_i$ and $\mathbf{f}_i$ corresponds to the coordinate and the high-level feature from the last stage of the i-th point, respectively.

Commonly, when we add the normalization and non-linearity to this formula, the feature can be presented as:

$$\mathbf{f}_i^1 = ReLU(BatchNorm(\omega_x^1(\mathbf{x}_i) + \omega_f^1(\mathbf{f}_i))). \tag{5}$$

Empirically, the point-based models benefit from separate application of batch normalization to $\mathbf{x}_i$ and $\mathbf{f}_i$ as follows:

$$\begin{aligned}\mathbf{f}_i^1 = ReLU(&BatchNorm_x(\omega_x^1(\mathbf{x}_i)) + \\ &BatchNorm_f(\omega_f^1(\mathbf{f}_i))).\end{aligned} \tag{6}$$

This tiny modification can significantly boost the performance of point-based models as well. For our RepSurf, $\mathbf{x}_i$ may contain polar coordinates, and $\mathbf{f}_i$ may be the features of RepSurf, RGB information. An illustration of our Channel De-differentiation is shown in Fig. 2

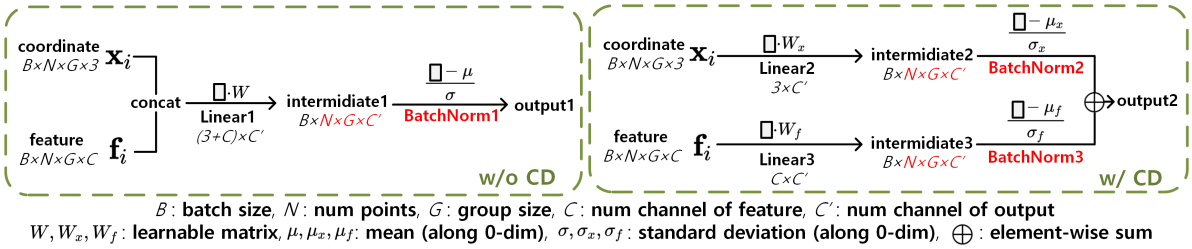

Figure 2. Illustration of our Channel De-differentiation.

5. Computation of FLOPs

To explore the efficiency of various models, we adopt the same formulas of complexity for the calculation of FLOPs. Since prior works are based on different versions of CUDA point cloud operations or non-CUDA ones, it may lead to an unfair comparison of efficiency based on FLOPs. Therefore, we treat the point cloud operations, including farthest point sampling, indexing, ball querying, knn querying, the same for the final estimation of FLOPs of different models. Following the common rules of FLOPs calculation, We count for the addition and multiplication of float points only.

For other basic operations, such as Convolution, ReLU, MLP, we adopt the default settings of THOP [1].

6. Computation of Speed

We test all methods with one V100 GPU and four cores Intel Xeon @ 2.50GHz CPU. The speed may vary with different sizes of input due to the parallelism of GPU. In this case, we set the batch size to 16 for all methods on the tasks of classification and segmentation. For detection, we set the batch size to 1 on the same experimental workstation in [1].

The FLOPs of one model can present the efficiency radically and theoretically. For an overall view of the efficiency, we adopt the practical method by testing the speed during the process of training and inference.

7. Implementation details

Classification. We implement Triangular and Umbrella RepSurf on PointNet++ [3] (SSG version). For both the datasets of ModelNet and ScanObjectNN, we set the initial learning rate to 0.001 with a decay rate of 0.7 for every 20 iterations. We use Adam for optimization. We apply data augmentation (including random scale, random shift, random dropout) when training on ModelNet, while we do not apply any augmentation methods for ScanObjectNN. Considering the quality of surface reconstruction, we sample 1024 points with farthest point sampling (FPS) method before input. We normalize the point clouds into the range of $[-1, 1]$ for ModelNet. We apply label smoothing with a ratio of 0.1.

Segmentation. We implement RepSurf on PointNet++ [3] (SSG segmentation version). For both the datasets of S3DIS and ScanNet, we set the initial learning rate to 0.5, with a decay rate of 0.1 on the 60th and 80th iteration. We use SGD, with a weight decay of $1e^{-4}$ for optimization. We apply data augmentation (including point cloud scaling, color contrasting, color shifting, and color jittering) when training on S3DIS and ScanNet. Considering the quality of surface reconstruction, we sample points with grid sampling method before input. We weight the loss with the ratio of classes.

Detection. We implement RepSurf on ScanNet V2 and SUN RGB-D following the practice of GroupFree [1].

[1]Official THOP repository: https://github.com/Lyken17/pytorch-OpCounter.

8. Detailed Experimental Results

We reveal the details of detection on the datasets of ScanNet V2 (mAP@0.25 in Tab. 1 and mAP@0.5 in Tab. 2) and SUN RGB-D (mAP@0.25 in Tab. 3 and mAP@0.5 in Tab. 4).

9. Visualization

9.1. Surface Reconstruction of RepSurf

We visualize the results after the process of surface reconstruction in Fig. 3. Different from prior methods, we only need to reconstruct discrete surfaces before calculating the features of Triangular and Umbrella RepSurf.

9.2. Geometry Sensitivity on Triangular RepSurf

We visualize the output of each channel of Triangular RepSurf on ScanObjectNN in Fig 4. Triangular RepSurf is eligible to perceive the local geometries numerically. Thus, the points on a flat shape have similar color, while the color of points on an edge changes obviously.

9.3. Geometry Sensitivity on Umbrella RepSurf

We visualize the output of each channel of Umbrella RepSurf on ScanObjectNN in Fig 5. Intuitively, Umbrella

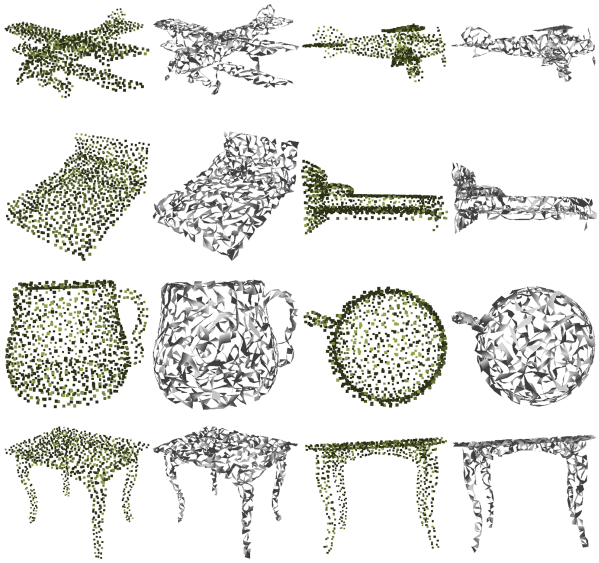

Figure 3. Visualization of surface reconstruction for RepSurf.

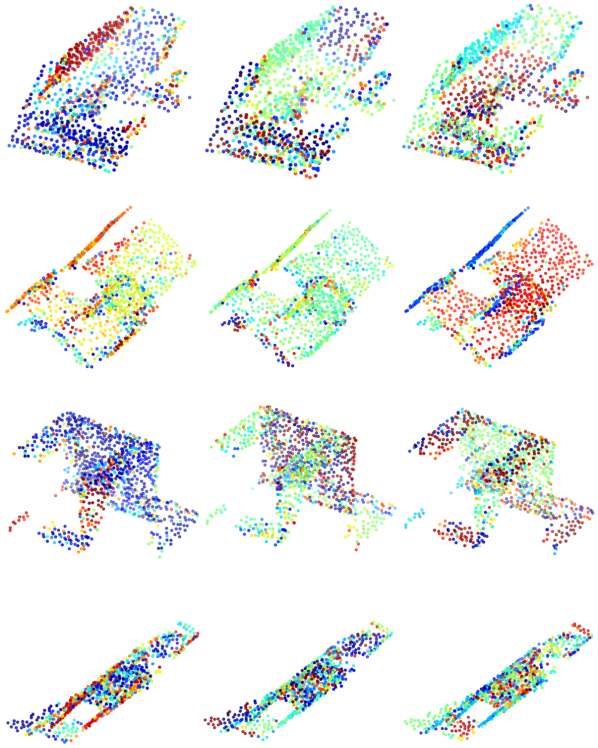

Figure 4. Visualization of the values of 3 channels from the normal vectors of Triangular RepSurf.

RepSurf can recognize the local geometries, including the edges and the planes of objects.

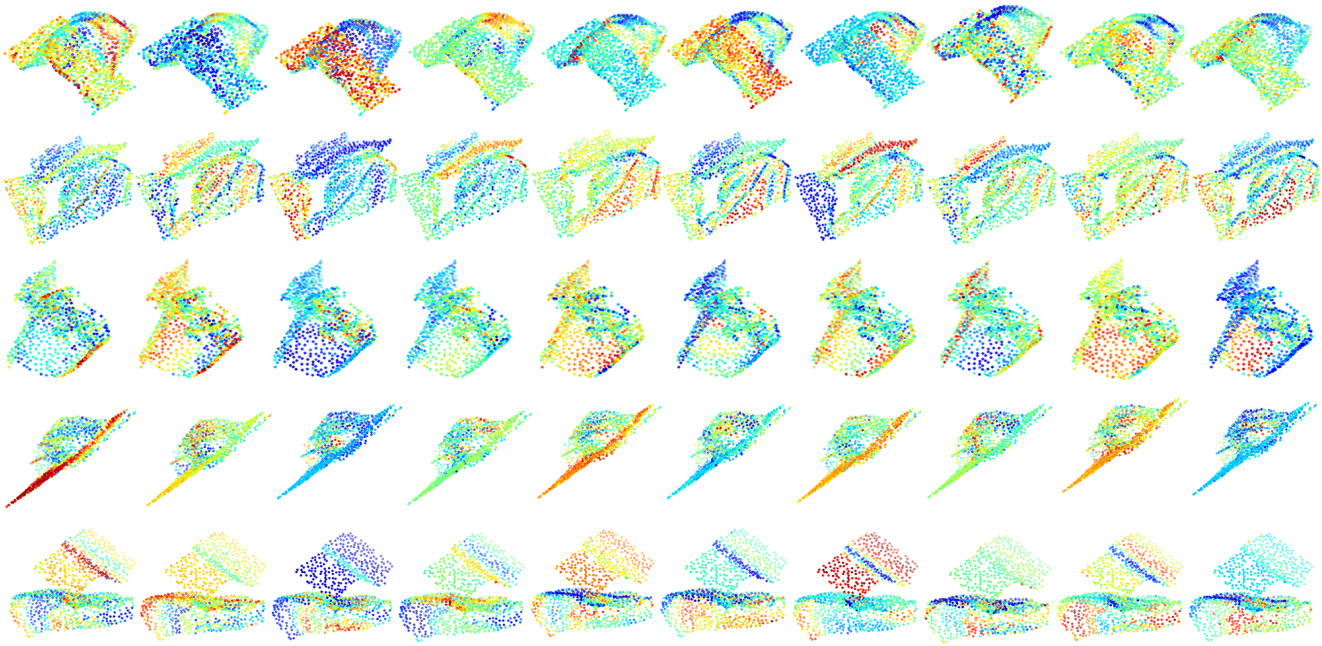

Figure 5. Visualization of the values of 10 channels from Umbrella RepSurf.

methods	backbone	cab	bed	chair	sofa	tabl	door	wind	bkshf	pic	cntr	desk	curt	fridg	showr	toil	sink	bath	ofurn	mAP
VoteNet [2]	PointNet++	47.7	88.7	89.5	89.3	62.1	54.1	40.8	54.3	12.0	63.9	69.4	52.0	52.5	73.3	95.9	52.0	92.5	42.4	62.9
H3DNet [5]	4×PointNet++	49.4	88.6	91.8	90.2	64.9	61.0	51.9	54.9	18.6	62.0	75.9	57.3	57.2	75.3	97.9	67.4	92.5	53.6	67.2
GroupFree^{6,256}	PointNet++	54.1	86.2	92.0	84.8	67.8	55.8	46.9	48.5	15.0	59.4	80.4	64.2	57.2	76.3	97.6	76.8	92.5	55.0	67.3
GroupFree^{6,256}	RepSurf-U	55.5	87.7	93.4	85.9	69.1	57.3	48.8	50.0	16.5	61.0	81.6	66.2	59.0	77.5	99.2	78.2	94.0	56.8	68.8
GroupFree^{12,512}	PointNet++^{2}	52.1	91.9	93.6	88.0	70.7	60.7	53.7	62.4	16.1	58.5	80.9	67.9	47.0	76.3	99.6	72.0	95.3	56.4	69.1
GroupFree^{12,512}	RepSurf-U^2	54.6	94.0	96.2	90.5	73.2	62.7	55.7	64.5	18.6	60.9	83.1	69.9	49.4	78.4	99.4	74.5	97.6	58.3	71.2

Table 1. Performance of mAP@0.25 for each category on the ScanNet V2 dataset.

methods	backbone	cab	bed	chair	sofa	tabl	door	wind	bkshf	pic	cntr	desk	curt	fridg	showr	toil	sink	bath	ofurn	mAP
VoteNet [2]	PointNet++	14.6	77.8	73.1	80.5	46.5	25.1	16.0	41.8	2.5	22.3	33.3	25.0	31.0	17.6	87.8	23.0	81.6	18.7	39.9
H3DNet [5]	4×PointNet++	20.5	79.7	80.1	79.6	56.2	29.0	21.3	45.5	4.2	33.5	50.6	37.3	41.4	37.0	89.1	35.1	90.2	35.4	48.1
GroupFree^{6,256}	PointNet++	23.0	78.4	78.9	68.7	55.1	35.3	23.6	39.4	7.5	27.2	66.4	43.3	43.0	41.2	89.7	38.0	83.4	37.3	48.9
GroupFree^{6,256}	RepSurf-U	24.9	79.6	80.1	70.4	56.4	36.7	25.5	41.4	8.8	28.7	68.0	45.2	45.0	42.7	91.3	40.1	85.1	39.2	50.5
GroupFree^{12,512}	PointNet++^{2}	26.0	81.3	82.9	70.7	62.2	41.7	26.5	55.8	7.8	34.7	67.2	43.9	44.3	44.1	92.8	37.4	89.7	40.6	52.8
GroupFree^{12,512}	RepSurf-U^2	28.5	83.5	84.8	72.6	64.0	43.6	28.3	57.8	9.6	37.0	69.7	45.9	46.4	46.1	94.9	39.1	92.1	42.6	54.8

Table 2. Performance of mAP@0.5 for each category on the ScanNet V2 dataset.

methods	backbone	bathtub	bed	bkshf	chair	desk	drser	nigtstd	sofa	table	toilet	mAP
VoteNet [2]	PointNet++	75.5	85.6	31.9	77.4	24.8	27.9	58.6	67.4	51.1	90.5	59.1
H3DNet [5]	4×PointNet++	73.8	85.6	31.0	76.7	29.6	33.4	65.5	66.5	50.8	88.2	60.1
GroupFree^{6,256}	PointNet++	80.0	87.8	32.5	79.4	32.6	36.0	66.7	70.0	53.8	91.1	63.0
GroupFree^{6,256}	RepSurf-U	81.1	89.3	34.4	80.4	33.5	37.3	68.1	71.4	54.8	92.3	64.3
GroupFree^{12,256}	RepSurf-U^2	81.9	89.9	35.3	81.2	33.5	38.1	68.8	71.5	55.6	93.2	64.9

Table 3. Performance of mAP@0.25 for each category on the SUN RGB-D validation set.

methods	backbone	bathtub	bed	bkshf	chair	desk	drser	nigtstd	sofa	table	toilet	mAP
VoteNet [2]	PointNet++	45.4	53.4	6.8	56.5	5.9	12.0	38.6	49.1	21.3	68.5	35.8
H3DNet [5]	4×PointNet++	47.6	52.9	8.6	60.1	8.4	20.6	45.6	50.4	27.1	69.1	39.0
GroupFree^{6,256}	PointNet++	64.0	67.1	12.4	62.6	14.5	21.9	49.8	58.2	29.2	72.2	45.2
GroupFree^{6,256}	RepSurf-U	65.2	67.5	13.2	63.4	15.0	22.4	50.9	58.8	30.0	72.7	45.9
GroupFree^{12,512}	RepSurf-U^2	66.5	70.0	14.9	64.7	17.0	24.7	52.0	60.7	31.7	74.4	47.7

Table 4. Performance of mAP@0.5 for each category on the SUN RGB-D validation set.

References

[1] Ze Liu, Zheng Zhang, Yue Cao, Han Hu, and Xin Tong. Group-free 3d object detection via transformers. *arXiv preprint arXiv:2104.00678*, 2021. 3

[2] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9277–9286, 2019. 6

[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. 1, 3

[4] Brook Taylor. *Methodus incrementorum directa et inversa*. Innys, 1717. 1

[5] Zaiwei Zhang, Bo Sun, Haitao Yang, and Qixing Huang. H3dnet: 3d object detection using hybrid geometric primitives. In *European Conference on Computer Vision*, pages 311–329. Springer, 2020. 6