PointCaps: Raw Point Cloud Processing using Capsule Networks with Euclidean Distance Routing

Dishanika Denipitiyage*, Vinoj Jayasundara†, Ranga Rodrigo*, Chamira U. S. Edussooriya*,#

*Department of Electronic and Telecommunication Engineering, University of Moratuwa, Sri Lanka
#Department of Electrical and Computer Engineering, Florida International University, Miami, FL, USA
†Department of Computer Science, University of Maryland, College Park, MD, USA

Abstract

Raw point cloud processing using capsule networks is widely adopted in classification, reconstruction, and segmentation due to its ability to preserve spatial agreement of the input data. However, most of the existing capsule based network approaches are computationally heavy and fail at representing the entire point cloud as a single capsule. We address these limitations in existing capsule network based approaches by proposing PointCaps, a novel convolutional capsule architecture with parameter sharing. Along with PointCaps, we propose a novel Euclidean distance routing algorithm and a class-independent latent representation. The latent representation captures physically interpretable geometric parameters of the point cloud, with dynamic Euclidean routing, PointCaps well-represents the spatial (point-to-part) relationships of points. PointCaps has a significantly lower number of parameters and requires a significantly lower number of FLOPs while achieving better reconstruction with comparable classification and segmentation accuracy for raw point clouds compared to state-of-the-art capsule networks.

Keywords: Point cloud reconstruction, classification, capsule networks, error routing

1. Introduction

Point clouds have been widely adopted in computer vision due to their applications in autonomous driving, augmented reality, robotics, and drones. A large variety of 3D sensors (e.g., LiDARs) used in such applications produce raw point clouds as their default output, requiring no additional processing. Even though raw point clouds are disordered and irregular, they are still a popular choice for 3D processing due to their ability to preserve the geometric information in 3D space without any discretization.

Deep learning based raw point cloud processing has gained wide adaptation in object classification, reconstruction, and segmentation. A prominent early attempt at directly processing raw point clouds is Pointnet [1], which learns a spatial representation of a point cloud and aggregates individual features to generate a global representation. One limitation of this work is that it discards the spatial arrangements of the points in local regions while aggregating features through the pooling operation. However, consideration of spatial arrangements is important because similar local regions have distinct spatial arrangements due to permutation invariance. Following the PointNet [1] architecture, PointNet++[2] proposed a hierarchical network architecture to combine local features. Furthermore, EdgeConv [3]

and PointCNN [4] proposed a convolutional local feature aggregator based on the neighbourhood graph. Above methods have mainly focused on improving global feature vector via different local feature extracting methods, predominately using k-NN-like clustering techniques to represent the point-to-part relationship. Learnable point-to-part relationship—manifested in the form of capsules [5]—is more powerful in this context. Therefore, we extend the capsule networks [5] to identify spatial relationship in local regions while considering the feature existence of local regions.

A capsule's ability to learn the spatial relationship of local regions stems from dynamic routing algorithm which establishes the mapping between the lower level capsules and higher level capsules. In other words, a capsule's activity vector is able to represent a specific type of object or an object part through this routing agreement. 3D-PointCapsNet [6] is the first architecture to formulate capsules with raw point clouds, generating the latent representation through fully-connected capsules resembling the multi-layer perceptron architecture. This approach is computationally intensive resulting in longer training and testing time and 3D-PointCapsNet [6] failed to identify proper point to part representation in unordered point clouds. Furthermore, 3D-PointCapsNet [6] fails at representing the entire point cloud as a single capsule causing the latent representation to be not linearly separable and requires a separate SVM for classification. A recent work [7] uses such a representation, but still suffers from high complexity due to feature aggregation through clustering. Nevertheless, there are two main problems associated with the capsule networks: 1) Since the logits values in the dynamic routing is bounded, dissimilarity between capsules ranges between -1 and 0. As a result, similarity gap of dissimilar capsules and similar capsules is reduced. 2) The original capsule network implementation assumes static pixel locations, However, point clouds are irregular.

In order to address these limitations, we propose *PointCaps*: a novel capsule based auto-encoder architecture, which has two novel *convolutional* capsule layers, to capture point-to-part spatial relationships and vice versa. Instead of using a traditional transformation matrix to transform low dimensional features to high dimensional features, our approach adopts the 2D convolution capsule idea into sparse 3D point clouds by creating capsules along the feature axis. More importantly, there is a significant reduction in the number of parameters in the capsule layer due to parameter sharing in *convolutional* capsules. This leads to a significant reduction in the *computational complexity* while providing better identification of geometric and spatial relationships between the parts. Furthermore, we employ a novel routing algorithm: dynamic Euclidean distance ($\mathcal{L}_2$ based routing (ER) in multiple capsule layers instead of dynamic routing (DR) as a solution to the lower similarity gap in dynamic routing. This increases the resolution of highly dissimilar capsules in between $-\infty$ and 0 instead of -1 to 0. Moreover, we represent the entire point cloud as a single capsule by adopting the approach of Sabour *et al.* [5] and replacing the decoder with a class-independent decoder proposed by Rajasegaran *et al.* [8]. PointCaps's ability to compress single point cloud to a vector of instantiation parameters enables us to explore the robustness of the model to noise while completing clasification and reconstruction tasks simultaniously. To recover lost fine-grained spatial information, we introduce a skip connection between the encoder and the decoder. **Our contributions are three-fold:**

- We propose a novel capsule auto-encoder architecture to classify, reconstruct, and segment raw point clouds. Further, we propose a novel convolution capsule layer with dynamic Euclidean routing instead of dynamic routing to capture part-whole relationships.

- To the best of our knowledge, PointCaps is among the first to adapt a class-independent decoder to reconstruct 3D point clouds.

- We evaluate classification accuracy, reconstruction error, and segmentation accuracy (in terms of mean intersection over union (IoU)) of PointCaps using standard benchmarks, where our approach surpasses the current state-of-the-art of reconstruction error and provides comparable performance in point cloud classification and segmentation despite having 85% less parameters and requiring 72% less floating point operations per second (FLOPs) compared to previous capsule based architectures.

2. Related Work

Deep learning applications of point clouds include 3D object detection, object classification [1, 2, 9, 10], reconstruction [11, 1, 12], scene labeling [13, 14], segmentation [1, 2, 15], point cloud completion [16, 17], layout inference [18], and point cloud registration [19, 20, 21]. The three main categories of 3D object classification based on the input to the deep learning network are volumetric representation [15, 22], view-based [23] and raw point cloud methods [1, 7]. In this paper, we will be focusing on raw 3D point cloud object classification and reconstruction.

Deep networks on point clouds: The capability of processing irregular, unordered point clouds through point-wise convolution and permutation invariant pooling proposed by PointNet [1] paved way for various point cloud-specific architectures such as PointNet++ [2], spherical convolution [24], Monte-Carlo convolution [25], graph convolution [9, 26] and SO-Net [27]. Unlike PointNet, PontNet++ [2] aggregate local features into a global feature vector forming a hierarchical feature learning architecture through farthest point sampling. Thereafter, improved convolotion operations [4] has been proposed to group local region features. SO-Net [27] proposed self-organization networks where spatial distribution of point clouds are used in an auto-encoder architecture to enhance the performance. A better upsampling method was introduced in PU-Net [28], and conversion of 2D grid into 3D surface was proposed by FoldingNet [29]. AtlasNet [30] is an extension of FoldingNet [29] which uses multiple data patches. PPF-FoldNet[31], which is based on supervised PPFNet [32], uses FoldingNet decoder [29] to enhance local feature extraction. However, all of the above methods use pooling operation to learn global features from local features based on the feature existence. We focus both existence of features in local regions and their spatial relationship through a capsule-based architecture and employ a new routing algorithm to aggregate the geometric features and spatial relationships in the local region.

Capsule networks: Hinton *et al.* [33] proposed capsule networks, a novel method to group neurons, which greatly impacted object classification in deep learning. Sabour *et al.* [5] extended this idea by proposing dynamic

routing between capsules. The success of capsule networks in object classification translates well into 3D point cloud classification due to its ability to capture spatial relationships through dynamic routing. Moreover, the instantiating parameters available in these networks are capable of capturing various properties (e.g., size, position, and texture) of a particular entity. In view of this, our work focuses on a novel auto-encoder architecture to classify and segment raw point clouds, and achieves minimum reconstruction error using capsule networks.

Several recent works address the use of capsule networks in point cloud classification, reconstruction, and segmentation. The 3D-PointCapsNet [6] is the first to devise a capsule network for raw point clouds where part segmentation classification is completed in an unsupervised way. However, it fails to model an entire point cloud as a single capsule. Several previous works [34, 12] have proposed supervised capsule architectures for point cloud classification. Cheraghian *et al.* [12] applies capsule networks as a drop-in replacement for a fully connected classifier. However, these models are trained in a supervised manner, in contrast to our auto-encoder architecture in PointCaps. Point2SpatialCapsule [7] uses capsule networks to encode fixed spatial locations into capsules. These capsule network architectures[12, 6, 7] directly use classification capsule layer with fully-connected capsule architecture for feature representation. This and the presence of k-NN clustering [7] lead to high computational complexity. On the other hand, convolutional capsule layer (PointCapA and PointCapB described in Sec. 3) and the absence of k-NN clustering in PointCaps provides significant reduction in computational complexity.

3. Method

The 3D point capsule Network [6], has proposed an end to end trainable auto encoder architecture for several common point cloud-related tasks. Inspired by the benefits of capsule network, we propose PointCaps, for processing a point cloud by simultaneous classification and reconstruction, and later achieve segmentation. Point cloud processing differs significantly from regular deep networks based vision tasks due to the irregular and unordered nature of point clouds. The high level processing pipeline of PointCaps: 1) reduces the size of the original data using convolutional capsule networks 2) generates the latent vector representations followed by reconstructing the point cloud using deconvolutions.

In the following sections, we first describe the overall PointCaps architecture. Second, we describe the different types of capsule layers we employ. Finally, we elaborate the routing mechanisms with Euclidean distance.

3.1. PointCaps Architecture

The proposed point-cloud classifier reconstructor network comprises an encoder with Euclidean and dynamic routing, and a class-independent decoder. The encoder architecture contains three types of capsule layers to learn spatial and geometric features of irregular, unordered 3D data. we designed the overall architecture as Fig. 1. The input to PointCaps is a 3D point cloud with N points. We chose $N = 2048$ following the work by Zhao *et al.* [6]. Two 1D convolution layers process this input to produce a feature vector of length 64, one for each point. This layer

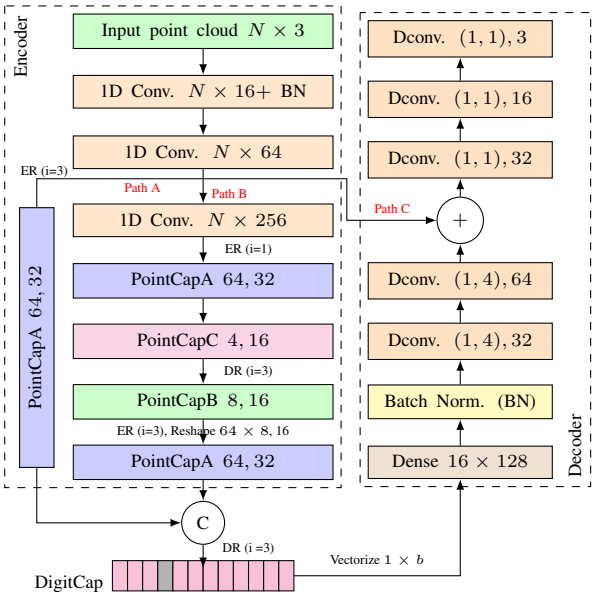

Figure 1: Model Architecture, PointCapA and PointCap B are convolutional capsule layers where PointCapA uses Euclidean routing and in the other capsules dynamic routing is applied. DR denotes the dynamic routing whereas ER denotes dynamic Euclidean distance routing

is followed by a *PointCapA* capsule layer: (Fig. 1.: path A) which creates the point-to-part relationships. The second path (Fig. 1.: path B) generates point-to-part relationship through sparse subspace. The direct use of 2D capsule layer gives larger parameter space which increases the use of computational resources. Therefore we use PointCapA as a dimension reduction technique in the second path and follow *PointCapC* and *PointCapB* capsule layers to retain the essential properties of parts. Then the parts are regenerated using a PointCapA capsule layer. The two paths: path A and path B are concatenated. Using the concatenated output, the DigitCap capsule layer generates the latent representation of a point cloud. Note that *PointCapB* is an upsampling layer, deployed as an intermediate capsule layer after the *PointCapA* capsule layer, which generates various properties of a given part that is available in the point cloud. Furthermore, *PointCapC* layer is a generic convolutional layer, with squashed output.

It is important to note that the original argument of dynamic routing in capsule network [5] was the capturing of intrinsic geometric properties of the object. These capsule representations sometimes may not correspond to human visible part segments in the object. However, we expect to centralize semantically similar regions in the object so that a human can identify. Furthermore, the coupling coefficient determines the agreement between the current output and the prediction using cosine similarity. As the coupling coefficient and logits are bounded, the gap between highly dissimilar capsules lies within -1 and 0. We increase this gap by making logits unbounded while keeping the coupling coefficient bounded using Euclidean distance. This increases the dissimilarity range between $-\infty$ and 0. We employ novel *PointCapA*, which predicts the possible point-to-part representation for each point using a given dynamic Euclidean ($\mathcal{L}_2$) distance routing algorithm (see Sec. 3.2).

3.1.1. PointCapA

Let $\Psi^l \in \mathbb{R}^{c^l \times n^l}$ be the input to the PointCapA layer, where c^l and n^l denote the number of input capsules and the input number of atoms (capsule dimension), respectively. Initially, each point with its features is considered as an input to the layer, and $\Psi^{l+1} \in \mathbb{R}^{c^{l+1} \times n^{l+1}}$ is considered as the output capsule from the layer l. The output capsules correspond to different local regions in the point cloud. The activity vector parameter interprets different properties of local region such as size, orientation, and texture.

The operation of the PointCapA 1D convolutional capsule is as follows. First Ψ^l is convolved with $(c^{l+1} \times n^{l+1})$ number of ψ_l kernels, forming $\Psi_A^{conv} \in \mathbb{R}^{c^l \times (c^{l+1} \times n^{l+1})}$, where $\psi_l \in [1, n^l]$. Then swish activation function [35] is applied as the pre-activation function to Ψ_A^{conv}, and then reshaped to generate the vote matrix V_A which has the shape of (c^l, c^{l+1}, n^{l+1}). Using a 1D convolution and choosing the kernel height as one in the transformation matrix has two advantages: 1) it provides a solution to the order invariance problem 2) it allows the network to keep the value of c^l (input number of capsules) unchanged. Then we feed the vote matrix to the routing algorithm as described in Sec. 3.2.

Similar to the approach used in [5], the transformation matrix learns the part-to-whole relationships between the lower and higher level capsules by updating the logits based on the similarity between the input capsule and the output capsule.

3.2. Routing Algorithm

Routing is a standard method that is used in capsule networks to identify the relevance between a lower level capsule and an upper level capsule [5]. In PointCaps, we employ routing to generate point to parts relationships. Unlike Sabour *et al.* [5], where the agreement between the current output V_j and prediction $v_{A_{j|i}}$ is the dot product between two quantities and the logits are updated based on the measurements for the next iteration, our novel Euclidean distance routing employs the Euclidean distance to find the relevance between capsule layers, and experimentally proving that Euclidean distance provides better performance compared to cosine similarity.

The operation of the routing algorithm is as follows. The routing algorithm maps a block of capsules in the child capsule to the parent capsule. Let the vote tensor (votes) be denoted by $V \in \mathbb{R}^{c^l \times c^{l+1} \times n^{l+1}}$. Following the [5], we initialize logits B_s as 0 where $B \in \mathbb{R}^{c^l \times c^{l+1}}$. Then the corresponding coupling coefficients K are generated by applying the *Softmax* function, defined as

$$k_{ij} = \frac{\exp(b_{ij})}{\sum_r \exp(b_{ir})}, \tag{1}$$

on logits B, where $i \in c^l$ and $j \in c^{l+1}$ This results the iterative dynamic routing process. Here the logits are normalized over all the predicted capsules in layer l because each single capsule in layer l predicts the outputs for all the capsules in layer $l+1$. Then these predictions are weighted by $k_{ij} \in K$ as

$$s_j = \sum_i k_{ij}.v_{j|i} \tag{2}$$

and s_j is applied to the squash function, given by

$$\hat{s}_{ij} = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \cdot \frac{s_j}{\|s_j\|}. \tag{3}$$

The squash function is a non linear function that ensure the higher probability of the existence of an entity to converge to a length nearly 1 and lower probabilities t0 get a length of almost 0.

Sabour et al. [5] proposed cosine similarity as an agreement between the current output V_j and the prediction $v_{A_{j|i}}$. The logits are updated based on this similarity measure. We used Euclidean distance to calculate error between two quantities and updated the logits values using,

$$b_{ij} \leftarrow b_{ij} - \|v_{Aj|i} - \hat{s}_j\|_2^2 \tag{4}$$

Algorithm 1 Dynamic Euclidean Routing Algorithm

procedure ROUTING
Require: $\mathbf{V}_A \in \mathbb{R}^{c^l \times c^{l+1} \times n^{l+1}}$, r and l
    $\mathbf{B} \leftarrow \mathbf{0} \in \mathbb{R}^{c^{l+1} \times c^l}$ Let $i \in c^l, j \in c^{l+1}$
    for r iterations **do**
        for all i, $k_i \leftarrow \texttt{softmax}(b_i)$
        for all j, $s_j \leftarrow \sum_i k_{ij} \cdot v_{j|i}$
        for all j, $\hat{s}_j \leftarrow \texttt{squash}(s_j)$
        for all j, $b_{ij} \leftarrow b_{ij} - \|v_{Aj|i} - \hat{s}_j\|_2^2$
    end for
    return $\hat{s}_j$
end procedure

3.2.1. PointCapB

PointCapB is a 2D convolutional capsule layer. We use PointCapB in PointCaps architecture to identify the properties of the entities such as length, elongation and texture. PointCapB operates as follows. Let the input tensor to the PointCapB be $\Psi^l \in \mathbb{R}^{E \times c^l \times n^l}$, where E is the number of entities, c^l is the number of capsules in the l^{th} layer and n^l is the capsule dimension. First, the input tensor Ψ^l is reshaped into $(E, c^l \times n^l, 1)$, where $(E, c^l \times n^l, 1)$ is the standard format of the input for the 2D convolution (H_{in}, W_{in}, C). Then the reshaped tensor is convolved with $(c^{l+1} \times n^{l+1})$ number of ψ_i 2D kernels, where the size of ψ_i is $(1 \times n^l)$. Note that the height and width of the input feature map for the 2D convolution are represented by E and $(c^l \times n^l)$, respectively. Maintaining the kernel height as 1, width and stride as n^l enable PointCapB to get a vote for single capsule from layer l. This process generates intermediate votes $\Psi_B^{conv} \in \mathbb{R}^{E \times c^l \times (c^{l+1} \times n^{l+1})}$, where the width of the output can be calculated as

$$W_{out} = \frac{c^l \times n^l - n^l + 0}{n^l} + 1 = c^l. \tag{5}$$

The intermediate votes are then reshaped into votes V_B. The vote tensor V_B has the shape of $(1, E, c^l, c^{l+1}, n^{l+1})$. We apply the pre-activation swish function [35]. Then the votes are fed to the routing algorithm as proposed by Rajasegaran *et al.* [8].

During routing, *softmax* function is applied on logits $B_s \in \mathbb{R}^{1\times E\times c^{l+1}\times c^l}$ for each $s \in c^l$ (initialized as 0) to generate coupling coefficients K_s. Here, we normalize the logits among all the predicted capsules from capsule tensor S in layer l. Each generated prediction in V_B is weighted by a factor $k_{prs} \in [0, 1]$, which results in a single prediction S_{pr}. Then the *squash* function is applied to the single prediction S_{pr}. The level of agreement between S and V_B is measured using cosine similarity to update the logits in the next iteration of the routing.

3.2.2. PointCapC

Now we describe the architecture of the PointCapC. Let $\Psi^l \in \mathbb{R}^{E\times c^l\times n^l}$ be the input to the PointCapC and $\Psi^{l+1} \in \mathbb{R}^{E\times c^{l+1}\times n^{l+1)}}$ be the output, where E is the number of entities or points, and c^l and n^l have usual meaning. First Ψ^l is reshaped into a matrix of shape $(E, c^l \times n^l)$, and 1D convolution is applied with $(c^{l+1} \times n^{l+1})$ kernels having the shape $(1, c^l \times n^l)$. Then the output is reshaped into a tensor of shape (E, c^{l+1}, n^{l+1}), followed by the *squash* function to produce the output.

3.3. Class Independent Decoder with Skip Connection

The decoder network is used to reconstruct the input point cloud using the instantiation vector extracted from the DigitCap in the encoder network. In the original capsules [5], DigitCap is masked to extract activity vectors and then used with three fully-connected layers to reconstruct the input image. During the training, they mask the digit capsule output with the true label, and the activity vector of maximum length is used for the inference stage. This vectorization results in a $\mathbb{R}^{a\times b}$ matrix with zeros except for the row corresponding to the true class or predicted class. Here a is the number of classes and b is the classification capsule dimension. Hence, the network gets class information which indirectly makes the decoder class dependent. DeepCaps [8] has claimed that class independent decoder based capsule networks are better for regularization.

To address this issue, in this paper, we use the class independent decoder proposed by Rajasegaran *et al*. [8] which provides better regularization in terms of capsule encoding. The proposed decoder network uses a class independent network by passing only the activity vector. Here, the masked activity vector is $P_t \in [1, b]$, where t is equivalent to true prediction in the training stage whereas $t = \arg\max_i(\|P_i\|_2^2)$ for testing. The decoder learns different distributions of different physical parameters irrespective of the class which makes the decoder class-independent. The network consists of a single fully connected network followed by five deconvolution layers. Moreover, the convolution layer from the encoder is skip-connected to the intermediate layer in the decoder as shown in Fig. 1. Further, Chamfer distance loss is used as the reconstruction loss and the input point cloud is recreated at the final layer.

Loss function: The total auto-encoder loss is defined as the summation of classification and reconstruction losses,

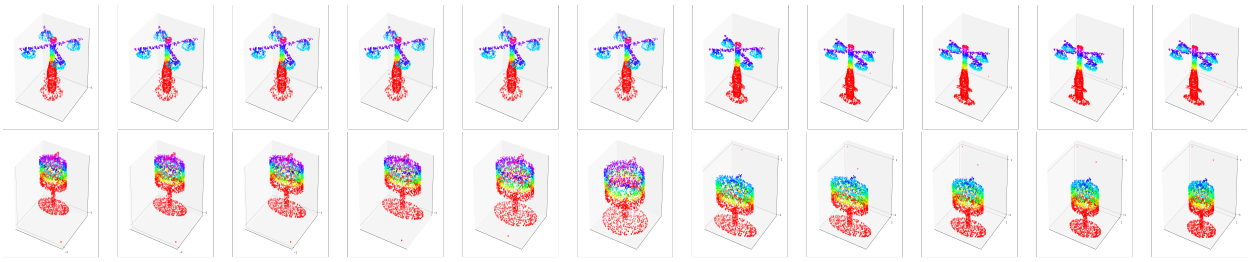

Figure 2: Point clouds generated by our decoder. The first row of point clouds is generated using ShapeNet core13 [36] and the second raw of the point clouds is generated using ShapeNetPart [37] dataset. This shows the PointCaps ability to capture geometric properties of the point cloud. For example, we can observe the compression along the y-axis when the 26^{th} dimension of the instantiation vector of ShapeNetPart lamp is changed between $[-5, 5]$.

i.e.,

$$Loss = \sum_{k \in a} L_k + \gamma\, CD(X, Y) \tag{6}$$

where $\gamma = 0.5$, For a each class k we use margin loss L_k as indicated in the Eq. 7, where $T_k = 1$ if the k^{th} class is present and otherwise zero and $m^+ = 0.9$ and $m^- = 0.1$ are the lower bound and upper bound of the correct and incorrect class. We use $\lambda = 0.5$ to reduce the effect of absent classes. The sum of the losses of all digit capsules is defined as total classification loss.

$$\begin{aligned} L_k = T_k \ \max(0, m^+ - \|v_k\|)^2 \\ + \lambda(1 - T_k) \ \max(0, \|v_k\| - m^-)^2 \end{aligned} \tag{7}$$

Similar to the other auto encoders, we use Chamfer Distance loss to measure the similarity between point clouds where X and Y are two different point clouds with the same number of points.

$$\begin{aligned} CD(X, Y) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y} \|x - y\|_2^2 \\ + \frac{1}{|Y|} \sum_{y \in Y} \min_{x \in X} \|x - y\|_2^2 \end{aligned} \tag{8}$$

4. Results and Discussion

Implementation: We implemented our system using Keras and Tensorflow. For training, we used RAdam optimizer [38] with an initial learning rate of 0.001 and decayed it by $1/10$ over several steps. We conducted an extensive evaluation of our network for point cloud classification and reconstruction using four datasets: ModelNet10 [22], ModelNet40 [22], ShapeNetPart [37], and ShapeNet Core13 [36]. For ShapeNetPart and ShapeNet Core13, we used 2048 points, and for ModelNet10 and ModelNet40 we used normal vectors with 1024 points. The model was trained using two GPUs: Nvidia P100 GPU and Nvidia V100 GPU.

Table 1: Reconstruction quality of ShapeNet Core13 [36] dataset. Reconstruction quality is reported by Chamfer Distance (CD) (multiplied by 10^3). *pc (#)* denotes point cloud with # points and *mesh (#)* denotes mesh with # vertices

Method	Input	CD
OccNet [42]	mesh (1511)	2.54
IM-NET [43]	mesh (1204)	2.36
BSPNet [41]	mesh (1073)	1.43
AtlasNetV2 [39]	pc (2048)	1.22
3D-PointCapsNet [6]	pc (2048)	1.49
Canonical Cap [40]	pc (2048)	0.97
PointCaps	pc (1024)	0.56
PointCaps	pc (2048)	**0.25**

4.1. Quantitative Evaluations

3D Reconstruction: We compare our method with both point-cloud based [39, 6, 40] and mesh-based [41, 42, 43] reconstruction methods using the Chamfer Distance matrix. We report quantitative results in Table 1. Our method *outperforms the state-of-the-art methods* on the ShapeNet Core13 [36] dataset. PointCaps (2048 points) surpassed the results of AtlasNet [30] by 79%, 3D-PointCapsNet [6] by 83%, and Canonical Capsules [40] by 74%. It is worthwhile to note that PointCaps is better than all the mesh-based reconstruction methods and achieves 61% improvement compared to BSPNet [41].

Point Cloud Classification: We test our model with ModelNet10 [22] and ModelNet40 [22] datasets and compare the model with existing network architectures. As shown in Table 2 PointCaps achieves second best performance with *significantly lower computational cost* (See Table 5) compared to capsule based methods. Moreover, PointCaps is slightly lower than the baseline method PointNet++ [2] by 0.2%.

Segmentation: In this section we evaluate the part segmentation of PointCaps. We use the ShapeNet Part dataset to train our model. Following the approach of 3D-PointCapsNet, we train two models: 1) using 1% of dataset (hereafter referred to as 1% training set) and 2) using 5% of dataset (hereafter referred to as 5% training set) as the training set. We used the complete testing dataset set to test our model. We use the same part segmentation evaluation method that is used in SO-Net [27] and 3D-PointCapsNet [6] to evaluate PointCaps: accuracy and IoU. As seen in the Table 4, PointCaps surpasses 3D-PointCapsNet [6] and SO-Net [27] with respect to accuracy and IoU. PointCaps achieves an accuracy of 0.85 and 0.87 for the 1% training set and 5% training set, where as the respective values for SO-Net [27] and 3D-PointCapsNet [6] are 0.78 and 0.85 for 1% training set and 0.84 and 0.86 for the 5% training set. We also observe that PointCaps achieves better IoU compared to SO-Net [27] and 3D-PointCapsNet [6]. For the 1% training set, PointCaps achieves an IoU that is 7.81% higher than SO-Net [27] and 2.98% higher than 3D-PointCapsNet [6]. For the 5% percent training set, Pointcaps surpasses SO-Net [27] and 3D-PointCapsNet [6] by

Table 2: ModelNet40 and modelNet10 classification accuracy comparison

Method	Input	ModelNet10	ModelNet40
PointNet [1]	1024×3	-	89.2%
PointNet++ [2]	$1024 \times 3 + n$	-	91.9%
DGCNN [3]	1024×3	-	92.2%
SAF-Net [10]	$1024 \times 3 + n$	-	93.4%
Kd-Net [44]	$2^{15} \times 3$	94.0%	91.8%
SO-Net [27]	2048×3	94.1%	90.9%
PointCNN [4]	1024×3	-	91.7%
RS-CNN [45]	1024×3	-	93.6%
Grid-CNN [46]	1024×3	**97.5** %	93.1%
CurveNet [47]	1024×3	96.3%	**94.2**%
3DCapsule [12]	1024×3	94.7%	91.5%
P2SCapsule [7]	$1024 \times 3 + n$	**95.9**%	**93.7**%
PointCaps	$1024 \times 3 + n$	94.7%	91.7%

Table 3: Comparison of classification accuracy and Chamfer Distance (CD) error ($\times 10^3$) of PointCaps with different routing algorithms for three datasets where PointCaps provides better reconstruction and comparable accuracy improvement. Here, *PointCaps* denotes Euclidean routing at PointCapA while dynamic routing (DR) is used in other Caps (see Fig. 1.), *All DR* uses DR for all the capsules, and *All ER* uses ER for all the capsules in the model. The performance of *PointCaps* model without using skip connection between encoder and decoder is denoted as *without skip connection*

Dataset	Input	PointCaps		All DR		All ER		W/o skip connection	
		Accuracy	CD	Accuracy	CD	Accuracy	CD	Accuracy	CD
ShapeNet core13 [36]	2048×3	**94.12**%	**0.25**	**94.12**%	0.34	93.84%	0.315	94.08%	17.42
ShapeNet part [37]	2048×3	98.33%	**0.117**	**98.43**%	0.337	98.29%	0.294	98.15%	4.07
ModelNet10 [22]	$2048 \times 3 + n$	**95.13**%	1.71	94.59%	**1.19**	94.69%	2.11	93.90%	9.67
ModelNet40 [22]	2048×3	**87.6**%	**0.891**	86.52%	1.023	87.5%	0.928	86.6%	18.13

Table 4: Part segmentation on ShapeNet-Part by learning only on the $x\%$ of the training data.

Method	1% data		5% Data	
	Acc	IoU	Acc	IoU
SO-Net [27]	0.78	0.64	0.84	0.69
3D-PointsCapsNet [6]	**0.85**	0.67	0.86	0.70
PointCaps	**0.85**	**0.69**	**0.87**	**0.72**

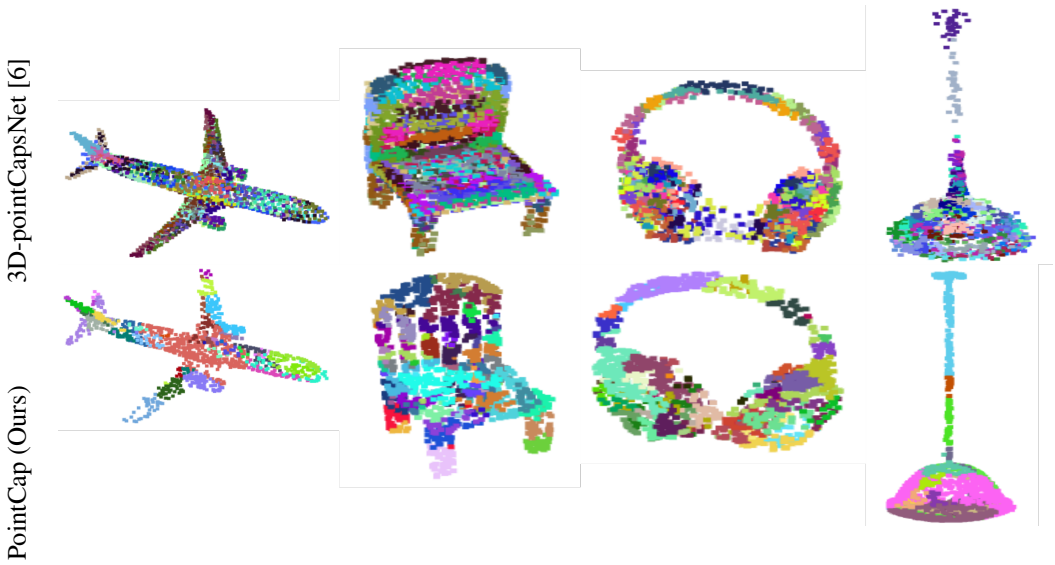

Figure 3: Part representation with dynamic routing in 3D-pointCapsNet [6] (Row 1) and with Euclidean routing in PointCapA (Row 2). 3D-pointCapsNet interprets 32 parts, each having 64 points, whereas PointCaps has 64 parts with different number of points. Note that PointCaps captures spatial relationships to form more human annotated local regions compared to 3D-pointCapsNet [6]

.

Table 5: The number of model parameters for the ModelNet40 dataset

Method	Params	FLOPs
PointNet [1]	3.48 M	957 M
PointNet++ [2]	1.99 M	3136 M
3D-PointCapsNet [6]	69.38 M	2231 M
P2SCapsule [7]	22.95 M	2251 M
PointCaps	**3.52 M**	**615 M**

4.34% and 2.85% respectively. These results prove that PointCaps achieves better segmentation quality than 3D-PointCapsNet [6] and SO-Net [27].

Computational Complexity: In this section, we compare the number of model parameters and FLOPs of PointCaps for ModelNet40 classification to recent capsule domain state-of-the-art models. Even though Point2SpatialCapsule outperforms in terms of accuracy, PointCaps achieves second best performance whilst *significantly reducing the number of FLOPs*, i.e., by 72%. Moreover, PointCaps has 3.5 Million parameters that is 85% lower than Point2SpatialCapsule [7] and 95% lower than 3D-PointCapsNet [6]. Table 5 summarizes the results. It is worthwhile to note that, compared to the backbone structure PointNet++, PointCaps has 35% reduction in number of FLOPs. Overall, PointCaps achieves performance comparable to the state-of-the-art models, with significantly lower computational complexity.

Robustness to Noise: To evaluate the robustness of our architecture to noise, we train a noise-free version of ModelNet10 [22] dataset using two augmentation techniques; 1) point perturbation and 2) adding outliers, and evaluate

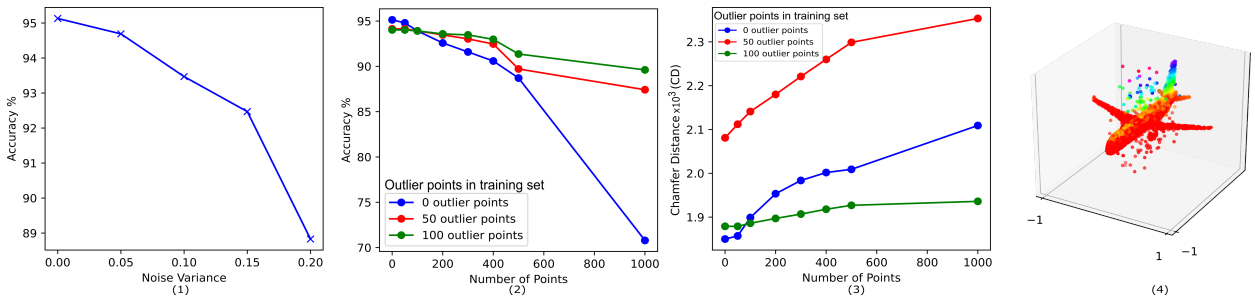

Figure 4: Noise analysis on ModelNet10 dataset. (1) The network is trained without any perturbation and tested with Gaussian noise, with variance in the range $0 - 0.2$. (2-3) The network is subjected to different number of anomaly points (in the X axis we increase the number of outlier points in the test set) and the performance (accuracy and reconstruction) is analyzed on ModelNet10 dataset. The network is trained with various number of outliers with Gaussian noise $\mathcal{N}(0, 0.2)$. (4) An example of 100 Points replaced with Gaussian noise $\mathcal{N}(0, 0.2)$.

the reconstruction loss and the accuracy matrix. In the perturbation test, Gaussian noise $\mathcal{N}(0, \sigma)$ is added to the points where $\sigma \in [0, 0.2]$. As shown in Fig. 4.(1), even though the network shows a considerable accuracy drop when $\sigma >= 0.15$, the network still achieves a minimum of 89.1% accuracy. Our outlier test replaces various numbers of points in both training and testing sets. Fig. 4. (2,3) depicts this behaviour. In Fig. 4. (2, 3) the $X-$axis denotes the number of outlier points in the test set. The three colours represent different number of outliers in the training set. As shown in Fig. 4. (2, 3), PointCaps delivers more than 90% accuracy up to 400 outliers in the test set. We also observe that the accuracy increases when we add outliers during the training phase. Hence, we conclude that the PointCaps is significantly more robust to Gaussian noise and to anomalies and provides good reconstruction.

Data Generation by Perturbation: We analyze the ability of PointCaps to generate data by perturbing the instantiation parameters. To experiment on that, we add random noise to only one non-zero instantiation parameter at a time. As seen in Fig. 2, we can observe that the instantiation parameter creates specific changes in the reconstructed point cloud. Furthermore, we observe that the new data samples are not distorted. This proves that latent representation of the PointCaps is capable of capturing interpretable geometric properties and the PointCaps augmenting data with less distortion. We achieve low distortion in data augmentation by applying an upper bound of noise to the instantiation parameter where the maximum variance of noise was manually inspected.

Points to Part Capsule: Here we analyse the capability of PointCapA (Path A in Fig. 1) at representing point-to-part relationship with dynamic Euclidean routing. We compare the ability to represent point to part relationships of PointCaps (Euclidean routing based) with 3D-PointCapsNet (dynamic routing based). The 3D-PointCapsNet generates symmetric local regions. This is due to the fact that they only consider existance of geomatrical information and disregard spatial relationships. This problem rises when pooling based methods are used to agregate features in local regions. It filters out features of different areas which represent the existence of characteristics in local regions while ignoring spatial relationships among local regions. However, in PointCaps capable of identifing different spatial arrangements in geometrically similar local regions through dynamic ER routing. The PointCapA is responsible for representing points-to-part relationships. Fig. 3 illustrates the local part representation of capsules. As indicated in

the Sec. 3.2 of the paper, each parent capsule has a logit which increases for the possible parent during routing. This represents the contribution of the lower level capsules to the higher level capsule. We use this logits to identify the relevant part labels for each point. As shown in Fig. 3, PointCapA is capable of specializing on the local regions compared to 3D-PointCapsNet (dynamic routing based).

4.2. Ablation Study

We first evaluate the impact of our novel routing algorithm, Euclidean distance routing (ER) for the accuracy and CD error. To evaluate this, we compare the accuracy and CD error of three implementations; 1) *PointCaps*: where PointCapA uses ER while PointCapB and Digitcap use DR, 2) *All DR*: where all capsule layers use DR and 3) *All ER*: where all capsule layers use ER. As shown in Table 3, the accuracy of PointCaps is slightly above or on-par compared to two other routing techniques. More notably, PointsCaps considerably surpasses All DR network (both accuracy and Chamfer distance) for the benchmark dataset, ModelNet40. Further, except for ModelNet10 data set, PointCaps achieves the best CD. These observations confirm that the use of ER for PointCaps achieves better accuracy and CD. Moreover, we observe that PointCaps provides faster convergence for all the datasets.

Secondly, we evaluate the impact of skip connection on accuracy and CD error. We compare two implementations; 1) *PointCaps*: where a skip connection is used between the encoder and decoder and 2) *W/o-skip-connection*: which does not contain a skip connection. In Table 3, we show that PointCaps concurrently achieves better CD error (80% improvement) for all the datasets. This observation proves our previous intuition for using a skip connection; the use of a skip connection results in lower reconstruction error.

5. Conclusion

In this work, we presented a novel capsule network based architecture for raw point cloud reconstruction, classification, and segmentation. Our approach of using 1D convolutional capsule architecture helps to significantly reduce computational complexity while retaining the global context. Our PointCapsA layers is capable of representing human-interpretable point-to-part relationships. We also introduced a novel routing mechanism, dynamic Euclidean distance routing (as opposed to dynamic routing), and class-independent latent representation. These improved reconstruction, classification, and segmentation accuracy of raw point clouds. Further, our proposed architecture is capable of augmenting data by perturbing instantiation parameters with no distortion.

References

[1] C. R. Qi, H. Su, K. Mo, L. J. Guibas, Pointnet: Deep learning on point sets for 3D classification and segmentation, in: CVPR, 2017, pp. 652–660.

[2] C. R. Qi, L. Yi, H. Su, L. J. Guibas, Pointnet++: Deep hierarchical feature learning on point sets in a metric space, in: NIPS, 2017, pp. 5099–5108.

[3] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, Dynamic graph CNN for learning on point clouds, Acm Transactions On Graphics 38 (5) (2019) 1–12.

[4] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, B. Chen, Pointcnn: Convolution on x-transformed points, in: NIPS, 2018, pp. 820–830.

[5] S. Sabour, N. Frosst, G. E. Hinton, Dynamic routing between capsules, in: NIPS, 2017, pp. 3856–3866.

[6] Y. Zhao, T. Birdal, H. Deng, F. Tombari, 3D point capsule networks, in: CVPR, 2019, pp. 1009–1018.

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

[8] J. Rajasegaran, V. Jayasundara, S. Jayasekara, H. Jayasekara, S. Seneviratne, R. Rodrigo, Deepcaps: Going deeper with capsule networks, in: CVPR, 2019, pp. 10725–10733.

[9] W. Wang, Y. You, W. Liu, C. Lu, Point cloud classification with deep normalized reeb graph convolution, Image and Vision Computing 106 (2021) 104092.

[10] S.-H. Lee, C.-S. Kim, SAF-Nets: Shape-adaptive filter networks for 3D point cloud processing, Journal of Visual Communication and Image Representation 79 (2021) 103246.

[11] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, R. Urtasun, 3D object proposals for accurate object class detection, in: NIPS, 2015, pp. 424–432.

[12] A. Cheraghian, L. Petersson, 3Dcapsule: Extending the capsule architecture to classify 3D point clouds, in: WACV, 2019, pp. 1194–1202.

[13] K. Lai, L. Bo, D. Fox, Unsupervised feature learning for 3D scene labeling, in: ICRA, 2014, pp. 3050–3057.

[14] N. Qin, X. Hu, H. Dai, Deep fusion of multi-view and multimodal representation of als point cloud for 3d terrain scene recognition, ISPRS journal of photogrammetry and remote sensing 143 (2018) 205–212.

[15] D. Maturana, S. Scherer, Voxnet: A 3D convolutional neural network for real-time object recognition, in: IROS, 2015, pp. 922–928.

[16] W. Yuan, T. Khot, D. Held, C. Mertz, M. Hebert, Pcn: Point completion network, in: 2018 International Conference on 3D Vision (3DV), IEEE, 2018, pp. 728–737.

[17] H. Wu, Y. Miao, R. Fu, Point cloud completion using multiscale feature fusion and cross-regional attention, Image and Vision Computing 111 (2021) 104193.

[18] A. Geiger, C. Wang, Joint 3D object and layout inference from a single RGB-D image, in: German Conference on Pattern Recognition, 2015, pp. 183–195.

[19] Y. Wang, J. M. Solomon, Prnet: Self-supervised learning for partial-to-partial registration, arXiv preprint arXiv:1910.12240.

[20] S. Zhang, H. Wang, J.-g. Gao, C.-q. Xing, Frequency domain point cloud registration based on the Fourier transform, Journal of Visual Communication and Image Representation 61 (2019) 170–177.

[21] B. Maiseli, Y. Gu, H. Gao, Recent developments and trends in point set registration methods, Journal of Visual Communication and Image Representation 46 (2017) 95–106.

[22] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, J. Xiao, 3D shapenets: A deep representation for volumetric shapes, in: CVPR, 2015, pp. 1912–1920.

[23] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, L. J. Guibas, Volumetric and multi-view CNNs for object classification on 3D data, in: CVPR, 2016, pp. 5648–5656.

[24] H. Lei, N. Akhtar, A. Mian, Spherical convolutional neural network for 3D point clouds, arXiv preprint arXiv:1805.07872.

[25] P. Hermosilla, T. Ritschel, P.-P. Vázquez, À. Vinacua, T. Ropinski, Monte Carlo convolution for learning on non-uniformly sampled point clouds, ACM Transactions on Graphics (TOG) 37 (6) (2018) 1–12.

[26] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering. 2016, No. Nips.

[27] J. Li, B. M. Chen, G. Hee Lee, SO-Net: Self-organizing network for point cloud analysis, in: CVPR, 2018, pp. 9397–9406.

[28] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, P.-A. Heng, Pu-net: Point cloud upsampling network, in: CVPR, 2018, pp. 2790–2799.

[29] Y. Yang, C. Feng, Y. Shen, D. Tian, Foldingnet: Point cloud auto-encoder via deep grid deformation, in: CVPR, 2018, pp. 206–215.

[30] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, M. Aubry, A papier-mâché approach to learning 3D surface generation, in: CVPR, 2018, pp.

216–224.

[31] H. Deng, T. Birdal, S. Ilic, Ppf-foldnet: Unsupervised learning of rotation invariant 3D local descriptors, in: ECCV, 2018, pp. 602–618.

[32] H. Deng, T. Birdal, S. Ilic, Ppfnet: Global context aware local features for robust 3D point matching, in: CVPR, 2018, pp. 195–205.

[33] G. E. Hinton, A. Krizhevsky, S. D. Wang, Transforming auto-encoders, in: International conference on artificial neural networks, Springer, 2011, pp. 44–51.

[34] Y. Zhao, T. Birdal, J. E. Lenssen, E. Menegatti, L. Guibas, F. Tombari, Quaternion equivariant capsule networks for 3D point clouds, arXiv preprint arXiv:1912.12098.

[35] P. Ramachandran, B. Zoph, Q. V. Le, Searching for activation functions, arXiv preprint arXiv:1710.05941.

[36] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al., Shapenet: An information-rich 3D model repository, arXiv preprint arXiv:1512.03012.

[37] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, L. Guibas, A scalable active framework for region annotation in 3D shape collections, ACM Transactions on Graphics (ToG) 35 (6) (2016) 1–12.

[38] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, J. Han, On the variance of the adaptive learning rate and beyond, arXiv preprint arXiv:1908.03265.

[39] T. Deprelle, T. Groueix, M. Fisher, V. Kim, B. Russell, M. Aubry, Learning elementary structures for 3d shape generation and matching, in: Advances in Neural Information Processing Systems, 2019, pp. 7433–7443.

[40] W. Sun, A. Tagliasacchi, B. Deng, S. Sabour, S. Yazdani, G. Hinton, K. M. Yi, Canonical capsules: Unsupervised capsules in canonical pose, arXiv preprint arXiv:2012.04718.

[41] Z. Chen, A. Tagliasacchi, H. Zhang, Bsp-net: Generating compact meshes via binary space partitioning, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[42] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, A. Geiger, Occupancy networks: Learning 3d reconstruction in function space, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4460–4470.

[43] Z. Chen, H. Zhang, Learning implicit fields for generative shape modeling, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 5939–5948.

[44] R. Klokov, V. Lempitsky, Escape from cells: Deep kd-networks for the recognition of 3d point cloud models, in: ICCV, 2017, pp. 863–872.

[45] Y. Liu, B. Fan, S. Xiang, 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] Q. Xu, X. Sun, C.-Y. Wu, P. Wang, U. Neumann, Grid-gcn for fast and scalable point cloud learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 5661–5670.

[47] T. Xiang, C. Zhang, Y. Song, J. Yu, W. Cai, Walk in the cloud: Learning curves for point clouds shape analysis, arXiv preprint arXiv:2105.01288.