CN: Channel Normalization For Point Cloud Recognition

Zetong Yang[1*], Yanan Sun[2*], Shu Liu[3], Xiaojuan Qi[4], and Jiaya Jia[1,3]

[1] The Chinese University of Hong Kong
[2] Hong Kong University of Science and Technology
[3] SmartMore
[4] The University of Hong Kong
{tomztyang, now.syn}@gmail.com sliu@smartmore.com xjqi@eee.hku.hk
leojia@cse.cuhk.edu.hk

Abstract. In 3D recognition, to fuse multi-scale structure information, existing methods apply hierarchical frameworks stacked by multiple fusion layers for integrating current relative locations with structure information from the previous level. In this paper, we deeply analyze these point recognition frameworks and present a factor, called difference ratio, to measure the influence of structure information among different levels on the final representation. We discover that structure information in deeper layers is overwhelmed by information in shallower layers in generating the final features, which prevents the model from understanding the point cloud in a global view. Inspired by this observation, we propose a novel channel normalization scheme to balance structure information among different layers and avoid excessive accumulation of shallow information, which benefits the model in exploiting and integrating multilayer structure information. We evaluate our channel normalization in several core 3D recognition tasks including classification, segmentation and detection. Experimental results show that our channel normalization further boosts the performance of state-of-the-art methods effectively.

Keywords: 3D recognition, point cloud, object detection, classification

1 Introduction

Recently, 3D point cloud recognition has attracted much attention in computer vision, since it benefits many real-life applications, such as autonomous driving [4] and robot manipulation. Compared to 2D recognition, this task is challenging because of several unique characteristics of point cloud for its sparse, unordered and locality sensitive properties.

To deal with raw point-cloud data, PointNet [19] extracts features for each point and aggregates them by max-pooling. Though effective, this method does

*Equal Contribution.

not capture multi-scale structure information, which is of great importance in point cloud recognition considering the diversity of 3D object size.

To fill this gap, later methods utilize a hierarchical structure stacked by several fusion layers to exploit multi-scale structure information. There are a variety of fusion layers. In general, they can be classified into two main streams. The first one is explicit fusion layer who applies concatenation, multiplication or summation to explicitly fuse current relative locations with previous features. *Set Abstraction* (SA) layers [20] and RS-CNN [14] are two representative structures of this track who apply concatenation or multiplication after multi-layer perceptron (MLP) encoding network to fuse current relative locations with previous features. Another track is implicit fusion layer. These layers [11, 29, 26, 16] utilize continuous convolution to encode relative locations to dynamic weights and merge previous features by matrix multiplication.

These fusion layers yield consistent performance boost for classification [30, 26, 14, 16], segmentation [30, 20, 16], and object detection [18, 22, 2]. Albeit performance improvement, existing fusion layers either require heavy computation [29, 14, 16, 26] or have their performance bottlenecks [20]. In this paper, we instead aim at a light-weight parameter-free and yet effective fusion layer.

Motivation Despite intensive research on fusion layer structures [14, 16, 26, 20, 13], it is rare to see systematic analysis to understand these operations for principled design. Importantly, we propose to evaluate its ability in aggregating multilayer structure information by a quantitative metric, called **difference ratio**. We note that difference ratio reflects the contribution of relative locations in various fusion layers on the final generated feature. If the difference ratio of a fusion layer is greater than a threshold, information of this layer generally dominates and consequently overwhelms information of other layers, which hampers the model from capturing multilayer structure information. Our empirical observation in Table 1 manifests that fusion layers with difference ratio closer to 1 tend to yield better performance since they fill the gap of influence among different layers.

Although existing methods are capable of alleviating the imbalance, the difference ratio of these fusion layers is still large and they introduce computational overhead. In this paper, we propose a simple and effective mechanism, called channel normalization, to fully utilize multilayer structure information. In each fusion layer, we rescale previous features by their difference ratios so as to enforce the model to treat location information from each fusion layer equally.

Our channel normalization does *not* introduce any extra parameters but yields impressive improvement in several 3D recognition tasks of classification, segmentation, and detection. Experimental results on multiple datasets including ModelNet40 [30], ShapeNet3D [30], and KITTI [4] prove that our CN pushes the performance of state-of-the-art (SOTA) recognition models further. Our overall contribution is the following.

- We propose to analyze fusion layers by difference ratio, which can reflect the effect of relative locations from different fusion layers on the final generated 3D representation features.
- We analyze the bottleneck of 3D recognition frameworks by difference ratio and raise the imbalance issue between shallower and deeper layers, which hampers the model from extracting proper multilayer structure information.
- We propose channel normalization, to accomplish considerable improvement on SOTA methods for all vital 3D recognition tasks without using extra parameters.

2 Related Work

View- and Voxel-based Methods View-based methods [5, 25, 3] treat 3D shape as a set of 2D images from different views and use deep neural networks to recognize them. These methods ignore the structure information in the point cloud and demand other operations to ensure performance, which may lead to considerable computation cost.

Voxel-based methods [30, 17] subdivide the raw point cloud to equally distributed voxels and employ CNN to extract their 3D representations. These methods are straightforward and efficient; but quantization during voxelization may cause information loss and performance bottleneck. In this paper, we focus on methods dealing with raw point cloud directly.

PointNet-based Methods To extract 3D representations from the raw point cloud data, PointNet [19] applies MLP network to learn features for each point and aggregates them by a symmetric function of max-pooling to extract 3D representation. Nonetheless, it ignores multi-scale structure information that is common and effective in 2D recognition.

To address this issue, later SOTA methods utilize hierarchical structures stacked by different types of fusion layers to fuse multi-scale structure information. In general, a fusion layer consists of two steps. The first one receives relative locations of points within a specific range as structure information. It also uses previous features as shallow structure information and merges them together. In the second step, merged features are sent to a MLP network to extract high-level representation. In the following subsection, we review these different fusion layers in current PointNet-based recognition frameworks.

Fusion Layers in PointNet-based Methods There are a variety of fusion layers that differ mainly in the first step. In general, they can be classified into two types. The first type is explicit fusion layer, who applies concatenation, summation or multiplication to explicitly fuse current relative locations and former features. The most straightforward one is direct concatenation, which is used in the SA layer of PointNet++ [20]. Other methods adopt an extra MLP network to encode relative locations and aggregate these encoded results

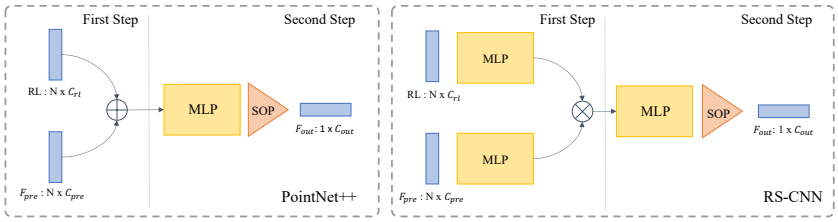

Fig. 1. Examples of explicit fusion layers. "RL", "N", F_{pre}, and F_{out} represent current relative locations, the number of interior points, previous features and output features respectively. "SOP" means symmetric operations, like max-pool.

with previous features by summation, concatenation or multiplication, like RS-CNN [14]. These methods achieve SOTA recognition performance but double the computation compared to direct concatenation.

The second type is implicit fusion layer. The most representative operation among these types of layers is continuous convolution, which is utilized in [11, 29, 26, 16]. For each point, these methods employ the relative location to generate the unique kernel weight and apply matrix multiplication based on the previous feature and the kernel weight. Multilayer structure information is obtained in an implicit way.

The recognition performance between these two types of fusion layers are comparable. In this paper, we mainly analyze the explicit ones systematically since their effectiveness and simpleness, and give an explanation behind the discrepancy of performance. We also develop a parameter-free normalization module to further boost the performance of these explicit SOTA recognition models.

3 Systematic Analysis

3.1 Background

Current 3D recognition frameworks are all hierarchical networks stacked by many fusion layers [26, 20, 16, 29]. The fusion layer is responsible for generating local structure features for target points and has a unique spherical range to cover structure information on a specific scale. The spherical range is similar to the receptive field of CNNs. Shallower fusion layers usually have smaller spherical ranges to extract local detail information, and those in deeper layers retain larger spherical ranges to capture structure information for the whole object. In this paper, we mainly focus on explicit fusion layers since they are simple and effective. All fusion layers below represent for explicit ones.

Normally, there are two parts in a fusion layer. The first combines relative locations of interior points in the current layer with features from the previous fusion layer. The second is an MLP network to extract high-dimension features. We illustrate two representative ones in Fig. 1.

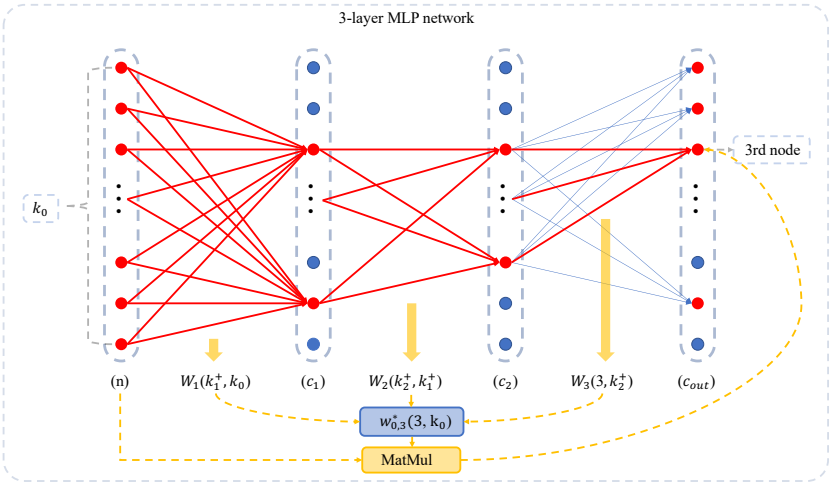

Fig. 2. A 3-layer MLP network. Nodes in "red" and "blue" represent activated (greater than 0) and inactivated (equal to 0) neuron nodes respectively. "Red" and "Yellow" lines illustrate the complete computation flow as Eq. (2) and merged computation flow as Eq. (3) respectively from input features to the certain activated node (the 3rd node) in final generated features. W_i and $W^\star_{0,3}$ stand for activated kernel weight in layer i and merged kernel weight computed by Eq. (3) respectively.

3.2 Analysis

In this subsection, we analyze the fusion layers considering the impact of relative locations in different fusion layers on the final extracted representation since it directly reflects the relative importance of different fused structure information. Our analysis further enables us to understand and explain how different fusion layers work and why they lead to varying performance. To quantify the impact, we propose an impact factor. It is followed by our development of "difference ratio" to measure impact rates, so as to compute the relative importance.

Impact Factor Since all kinds of fusion layers consist of an MLP network in the second step, we first consider a small MLP network with 3 layers in Fig. 2. Generally, a layer in an MLP network is composed of three parts: kernel multiplication, batch normalization and the non-linear unit. Since the batch normalization is a linear transformation, we can merge it with kernel multiplication (in the inference stage) and define the output of a 3-layer MLP network as

$$M_3 = \sigma(W_3 \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot X))), \tag{1}$$

where W_i, X and σ represent merged kernel weight in layer i, input feature and the non-linear unit "ReLU" respectively.

For any channel in M_3, if its value is greater than 0, there must be non-zero channels in M_2 multiplied with their corresponding kernel weights. This fact also

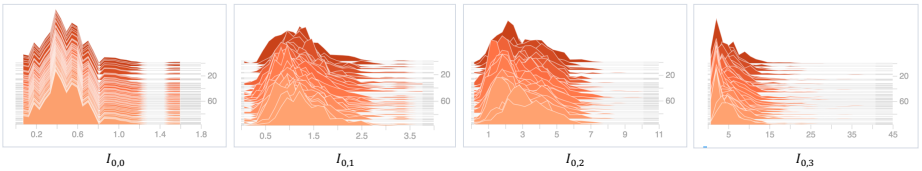

Fig. 3. We illustrate distributions of values in $I_{0,j}$ among different input features, where 0 and j represent indexes of different layers within an MLP network with 4 layers. The X-axis means the impact value. The Y-axis shows the number of channels with this impact value. The Z-axis contains indexes of different features within an MLP network.

holds for features M_2 and M_1. Generally, for an MLP network, for each non-zero channel in the final feature, there must be a computation flow from the original input X to current channel value as illustrated by "red" lines in Fig. 2.

This makes it possible to estimate the concrete contribution of input from a certain layer to each channel in the last generated features. For example, we assume that the list with k_i^+, k_i^- and k_i contains activated channel indexes (channels with positive values), inactivated channel indexes and total channel indexes for layer i respectively. This computation flow can be formulated as

$$\begin{aligned} M_1(k_1^+) &= W_1(k_1^+, k_0) \cdot X \\ M_2(k_2^+) &= W_2(k_2^+, k_1^+) \cdot M_1(k_1^+) \\ M_3(k_3^+) &= W_3(k_3^+, k_2^+) \cdot M_2(k_2^+) \end{aligned} \tag{2}$$

where (k_1, k_2) means "index operation" that gathers values by query indexes.

Based on Eq. (2), we merge these multiplication matrices, and build up a merged computation flow as

$$\begin{aligned} M_3(k_3^+) &= W_3(k_3^+, k_2^+) \cdot W_2(k_2^+, k_1^+) \cdot W_1(k_1^+, k_0) \cdot X \\ &= \mathbf{W_{0,3}} \cdot X \\ W^\star_{0,3}(k_3^+, k_0) &= \mathbf{W_{0,3}},\ W^\star_{0,3}(k_3^-, k_0) = 0 \\ M_3 &= W^\star_{0,3} \cdot X \end{aligned} \tag{3}$$

which is highlighted by yellow lines in Fig. 2. Suppose X has n input channels, $W^\star_{0,3}$ should have a shape of $[c_{out}, n]$ to transform X to features with c_{out} dimensions in a straightforward way.

Eq. (3) enables us to directly link the input to each output feature channel. Thus, we propose to use $\left\|W^\star_{0,i}\right\|$ to measure the impact of input in layer 0 on each channel imposed on the output feature in layer i , where $\|\cdot\|$ is an operation to calculate the L2-norm of each row of the matrix. $\left\|W^\star_{0,i}\right\|$ is a vector whose size is equal to the number of feature channels in layer i. For simplicity's sake, we denote this as impact factor $I_{0,i}$ and form it as

$$I_{0,i} = \left\|W^\star_{0,i}\right\| \tag{4}$$

The impact factor is a vector reflecting the impact of input over output, whose size equals to the output channel number. Each value in this vector represents the input influence on the corresponding channel of the output feature.

In Fig. 3, we count $I_{0,i}$ among a 4-layer MLP network with different input features, which illustrates that the influence of input grows rapidly with the increase of depth of layer i.

Similarly, the impact of relative locations in a certain fusion layer on different generated features can also be calculated. We adopt PointNet++ in our study since it has been widely adopted in multiple 3D computer vision tasks of detection [33], generation [9], scene flow [12], etc.

In a SA layer, it directly concatenates relative locations with the previous feature, and adopts an MLP network to extract high-level representation. In order to analyze the effect of relative locations in the current layer and previous features respectively, we rewrite the computation process in the SA layer as

$$\begin{aligned} F_i^l &= W_i^l \cdot X_i \\ F_i^f &= W_i^f \cdot F_{i-1} \\ F_i &= M_i(F_i^l + F_i^f) \end{aligned} \tag{5}$$

In each SA layer, it first transforms relative locations X_i and previous features F_{i-1} by weights W_i^l and W_i^f to obtain F_i^l and F_i^f, and then employ an MLP network M_i based on the sum of F_i^l and F_i^f to generate final features F_i.

F_i^f in Eq. (5) is derived from F_{i-1}^l and F_{i-1}^f while F_{i-1}^f is derived from F_{i-2}^l and F_{i-2}^f. In the beginning, F_0^f equals to 0, since there is no previous features in the first layer. We eliminate all F_i^f in Eq. (5) in a recursive way. Based on Eqs. (5) and (3), the recursion formula is given as

$$\begin{aligned} F_i^f &= W_i^f \cdot F_{i-1} = W_i^f \cdot M_{i-1}(F_{i-1}^l + F_{i-1}^f) \\ &= W_i^f \cdot W_{i-1,i}^\star \cdot W_{i-1}^l \cdot X_{i-1} + W_i^f \cdot W_{i-1,i}^\star \cdot F_{i-1}^f \end{aligned} \tag{6}$$

where $W_{i-1,i}^\star$ is the merged multiplication matrix in MLP network M_{i-1} by Eq. (3). This iteration process makes it possible to calculate features F_i^f generated by X_{i-1}.

We then calculate $I_{i-1,i}$ and $I_{i,i}$ as

$$\begin{aligned} I_{i-1,i} &= \left\| W_i^f \cdot W_{i-1,i}^\star \cdot W_{i-1}^l \right\|, \\ I_{i,i} &= \left\| W_i^l \right\|, \end{aligned} \tag{7}$$

and repeat the process above to obtain $I_{i,j}$ where $(i \leq j)$ for any fusion layers i and j in PointNet++ network.

In Fig. 4, we illustrate distributions of $I_{0,j}$, which measure the impact of input relative locations in fusion layer 0 imposed upon output features of fusion layer j. The contribution of relative locations in the first fusion layer grows even faster with the increase of index j.

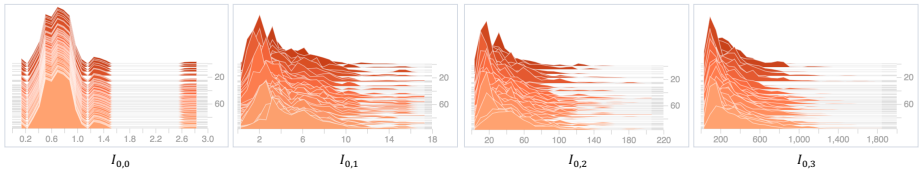

Fig. 4. Illustration of distributions of values in $I_{0,j}$ among different SA layers. The X-axis means the impact value. The Y-axis shows the number of channels with this impact value. The Z-axis represents indexes of different input points for an SA layer.

Methods		PointNet++ [20]	RS-CNN [14]	CN
Layer j	0	1	1	1
	1	16.35	1.22	1.04
	2	116.31	19.21	1.33
Accuracy (%)		90.7	92.9	93.3

Table 1. Comparison among different methods in terms of average $D_{0,j}$ and classification accuracy, where 0 indexes the first layer and j indexes other fusion layers.

Difference Ratio The impact factor quantifies the importance of input towards each channel of output, we now analyze the relative influence of input features from different layers towards the same output feature, which determines their relative importance in the final fusion result. We propose difference ratio $D_{i,j}$, which is the quotient between $I_{i,j}$ and $I_{j,j}$ as

$$D_{i,j} = \frac{I_{i,j}}{I_{j,j}}, \tag{8}$$

to judge if the relative location information from certain layer i overwhelms information from layer j. This can help us verify the quality of multilayer feature fusion. For example, if $D_{i,j}$ is close to 1 for any $(i \leq j)$, it means that all fused information is equally used, which makes the optimal fusion results.

This parameter reflects the contribution of relative location in different layers on the final generated features. In Table 1, we test $D_{0,j}$ between the first SA layer and others indexed by j in PointNet++ and average them in channels to get an overall value. As illustrated, when the network goes deeper, the influence of relative locations in the first layer not only increases on values but also enhances rapidly their proportions. That is, the features from the last SA layer mainly contain relative locations in the first SA layer, and ignore structure information from the current new neighborhood. This phenomenon is not conducive to the expansion of the receptive field and impedes the model from learning better structural information.

Table 1 also reveals the potential reason that RS-CNN draws better performance compared to original PointNet++. Because of the fusion layers, difference ratios are significantly reduced, which benefits the network in utilizing multi-scale structure hints and boost the recognition performance.

4 Channel Normalization

4.1 Approximate Difference Ratio Calculation

Even though current SOTA methods diminish difference ratios between previous and current structure information on the generated features, the gap between these two types of effect is still very large. In order to help the model exploit multiple structure information from different layers better and take full advantage of model capacity, it is a decent choice to rescale previous features i by $D_{i,j}$ to enforce this value to be 1.

However, it is intractable to calculate $D_{i,j}$ during training or inference directly because of the difficulty in calculating $W^{\star}_{i,j}$. Due to the application of ReLU, differently activated neurons lead to varying computation flow, which requires to obtain $W^{\star}_{i,j}$ for each of the flow. In a normal case, suppose there are N_t target points and for each point there are N_n interior points for calculating relative locations, we need to calculate $W^{\star}_{i,j}$ for $N_t \times N_n$ times, which is extremely large computation cost. Therefore, our main concern is to find a way to approximate the unique $W^{\star}_{i,j}$ for different points.

To this end, we propose a global approximation of $W^{\star}$. We treat all neurons to be activated. To simulate the activation of neuron nodes during inference, we utilize the probability of a to-active neuron to re-weight the corresponding weights. The probability is estimated as the ratio between the number of points whose neuron n is activated and total points in the whole scene. This approximation shares similar intuition with Dropout [23] where the not-activated neuron corresponds to the dropped neuron.

To be more specific, in order to approximate $W^{\star}_{i,j}$, for a certain layer m of the MLP network in the SA layer i, we multiply its weight matrix W_{i_m} with probability vector P_{i_m} to simulate activation of neuron nodes during inference. Considering a 3-layer MLP network in SA layer i, we approximate calculation of $W^{\star}_{i,i+1}$ by

$$M_i = (P_{i_3} \odot (W_{i_3} \cdot (P_{i_2} \odot (W_{i_2} \cdot (P_{i_1} \odot W_{i_1}))))) \cdot X = W^{\star}_{i,i+1} \cdot X, \qquad (9)$$

in which $\cdot$ and $\odot$ represent matrix- and element-wise multiplication.

Similarly, we estimate $W^{\star}_{i-1,i}$ and take it into Eq. (7) to calculate $I_{i-1,i}$ and $D_{i-1,i}$ with almost no extra effort.

4.2 Channel Normalization

After deriving $D_{i,j}$, we detail our channel normalization as follows. The main purpose of calculating $D_{i,j}$ is to balance the influence of structural information from different fusion layers on the final 3D representation. Apparently, if for all fusion layers ($i \geq 1$), $D_{i-1,i}$ equals to 1, $D_{i,j}$ always reaches 1 for all $i, j (i \leq j)$.

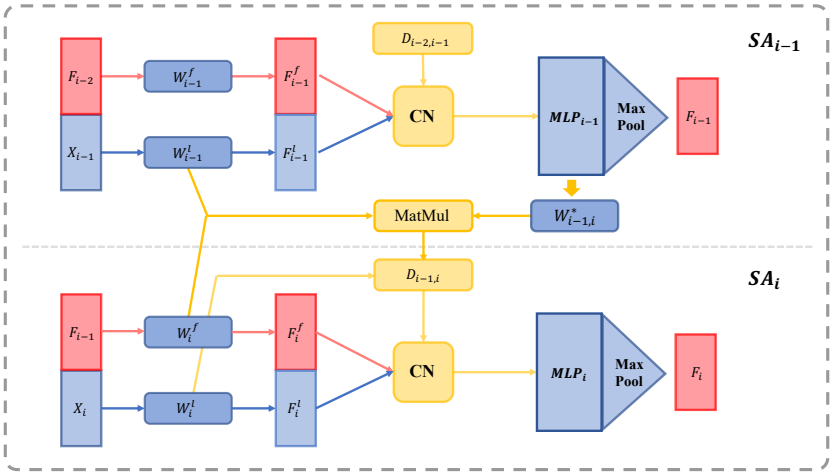

Fig. 5. Illustration of CN module in SA layer i.

Therefore, the best choice to align the influence is to calculate F_i as

$$D_{i-1,i} = \frac{I_{i-1,i}}{I_{i,i}} = \frac{\left\| W_i^f \cdot W_{i-1,i}^{\star} \cdot W_{i-1}^l \right\|}{\left\| W_i^l \right\|}, \qquad (10)$$
$$F_i = M_i(F_i^l + \frac{F_i^f}{D_{i-1,i}}),$$

in which we enforce $D_{i-1,i}$ to be 1.

We name this new operation as channel normalization (CN), which is visualized in Fig. 5. There are two main advantages of our proposed CN compared to former SOTA modules. First, compared to original SA layers in PointNet++, we do not introduce extra parameters. Second, for any fusion layer i in the network, our solution greatly reduces the gap between the values of all $D_{i,j}$ and 1, which is illustrated in Table 1.

Put differently, in any SA layer, the effect of relative locations from the current layer is the same as those from previous layers. Since relative locations from all fusion layers in the network contribute the same in the final features, they maintain structure information from every fusion layer, which benefits the model in recognizing objects with a variety of scales.

It may be ideal to set $D_{i-1,i}$ close to 1. But in this case, not all structure information from different fusion layers is equal. Actually, fusion layers are bound for different structure information regarding scales. For shallow layers, since the spherical ranges are too small to cover the entire object, the relative location information is less important compared to that in later layers. Similarly, for information in the deep fusion layers, because of the opposite reason, there is much noisy information, possibly misguiding the model in the recognition task.

Methods	Input	Point Numbers	Accuracy (%)
PointNet [19]	xyz	1024	89.2
Spec-GCN [27]	xyz	1024	91.5
PointCNN [10]	xyz	1024	91.7
DGCNN [28]	xyz	1024	92.2
PointConv [29]	xyz, norm	1024	92.5
RS-CNN [14] (SSG)	xyz	1024	92.2
RS-CNN [14] (MSG)	xyz	1024	92.9
PN2 [20]	xyz	1024	90.7
PN2 + CN (SSG)	xyz	1024	92.9
PN2 + CN (MSG)	xyz	1024	**93.3**

Table 2. Classification accuracy compared with SOTA methods on the ModelNet40 dataset. "PN2", "MSG", and "SSG" represent PointNet++ baseline, multi-scale grouping and single-scale grouping respectively. "norm" means using the normal vectors from the mesh models as input.

So the utility of structure information from a certain fusion layer depends on the scale of objects.

To resolve this problem, we extend CN by adding two learnable condition parameters α_1 and α_2. α_1 is to measure the importance of relative locations in previous layers, while α_2 is about the value of the current layer. Intuitively, if relative locations in the current fusion layer is more valuable in recognition compared to those in previous layers, α_2 gets larger than α_1 to highlight the influence of information from current layer. We express this relation as

$$F_i = M_i(\alpha_2 F_i^l + \alpha_1 \frac{F_i^f}{D_{i-1,i}}), \tag{11}$$

where F_i is the result after channel normalization in the i-th fusion layer.

5 Experiments

We conduct extensive experiments on a series of core 3D recognition tasks to verify the effectiveness of our CN and prove that reducing $D_{i,j}$, where $(i \leq j)$, is helpful for learning structure information at different scales. These experiments consist of classification on ModelNet40 [30] dataset, part segmentation on ShapeNet [30] and detection on KITTI [4] dataset. If not specified otherwise, all fusion layers in our model are the combination of SA layers and CN.

5.1 Classification on ModelNet40

We evaluate the classification accuracy of our model on the ModelNet40 dataset. This dataset contains 12,311 CAD models with 9,843 training shapes and 2,468 testing shapes in 40 different classes. For fair comparison, we follow the configuration in former papers by sampling 1,024 points uniformly for each model and

Methods	Types	Numbers	Class mIoU (%)	Instance mIoU (%)
Kd-Net [6]	points	4k	77.4	82.3
PointNet [19]	points	2k	80.4	83.7
SPLATNet [24]	-	-	82.0	84.6
KCNet [21]	points	2k	82.2	84.7
PointConv [29]	points, normal	2k	82.8	85.7
RS-CNN [14]	points	2k	84.0	**86.2**
PN2 [20]	points, normal	2k	81.9	85.1
CN	points	2k	83.9	85.8
CN2	points	2k	**84.3**	**86.2**

Table 3. Comparison among different 3D segmentation methods. "CN" is modified based on original PointNet++ baseline and "CN2" adds two more long-range connections as [14].

by normalizing them to a unit ball. During training, we apply random translation and scaling on the point cloud, and keep the same training schedule as PointNet++. We do not apply any voting operation in the testing phase and compare our model with SOTA 3D recognition frameworks in Table 2.

It shows that our CN module outperforms the PointNet++ baseline by a large margin and achieves the new SOTA results on recognition by raw point cloud data. We also provide a light-weight single scale grouping (SSG) model, in which there is only one spherical range for a single fusion layer. It also draws great improvement compared to PointNet++ baseline and outperforms RS-CNN (SSG) by 0.7%. These results clearly demonstrate the power of our CN module.

5.2 Segmentation on ShapeNet

We also experiment with 3D segmentation on the ShapeNet part segmentation dataset [30]. We take PointNet++ as our baseline and add the CN module to each SA layer. The comparison between CN models and other 3D recognition methods are listed in Table 3. We use two CN baseline models here. In the "CN" model, we directly apply CN to each SA layer in the original PointNet++ model. In "CN2", two more long-range skip connections are added to the "CN" model as that of [14] – it is for structure information loss during upsampling. The experiments show that our CN module promotes the PointNet++ baseline by 2% in terms of class mIoU without any extra parameters.

5.3 Detection on KITTI dataset

The KITTI [4] dataset contains 7,481 training point clouds and 7,518 testing point clouds in three different categories of Car, Pedestrian and Cyclist. In each class, there are three difficulty levels of "Easy", "Moderate" and "Hard", for distinguishing among objects with regard to depth to camera and occlusion. Our baseline model is 3DSSD [32], which is the SOTA single-stage 3D object

Methods	Sens.	$AP_{BEV}(\%)$			$AP_{3D}(\%)$		
		Easy	Mod	Hard	Easy	Mod	Hard
ContFuse [11]	R + L	94.07	85.35	75.88	83.68	68.78	61.67
SECOND [31]	L	91.81	86.37	81.04	84.65	75.96	68.71
PointPillars [8]		90.07	86.56	82.81	82.58	74.31	68.99
TANet [15]		91.58	86.54	81.19	84.39	75.94	68.82
HRI-VoxelFPN [7]		92.75	87.21	79.82	85.64	76.70	69.44
OHS [1]		93.59	87.95	83.21	88.12	78.34	73.49
3DSSD [32]	L	92.66	89.02	**85.86**	88.36	79.57	74.55
3DSSD [32] + CN		**94.51**	**90.50**	**85.86**	**90.55**	**79.89**	**76.31**

Table 4. Results on KITTI test set in class "Car". State-of-the-art single-stage object detector results are drawn from official benchmark. "Sens." means sensors used by the method. "L" and "R" represent using LiDAR and RGB images respectively.

detector presented in Table 4. It uses PointNet++ with multiple SA layers as the backbone.

For fair comparison, our model is trained under the same configuration as 3DSSD, including time schedule, data augmentation, input point cloud size etc. As shown in Table 4, our CN module significantly improves the detection accuracy of 3DSSD, especially on easy cases, which mainly contain cars close to the camera. For these cars with a shorter range, the LiDAR camera captures their surface points well and brings good-quality information in multiple structure levels of wheel, door, and car, for example.

The original SA layers in PointNet++ only focus on the relative locations in shallower level, which mainly include small hints like wheel or door. Larger structure information such as the whole car, is more likely to be ignored. This procedure does not make full use of the high-quality information given by the LiDAR camera and hampers the model from further improvement. With our CN module, the network exploits both local tiny structure and global objects. It greatly benefits the model in boosting detection performance.

We also compare the performance of 3DSSD with PointNet++, RS-CNN and CN. Their 3D mAPs are 82.37%, 82.88% and 85.01% respectively among KITTI moderate val set on class "Car". As listed, our CN model draws much better performance compared to its RS-CNN counterpart. We analyze its reason as below. As illustrated in Table 1, in these fusion layers, $D_{0,j}$ consistently increases with regard to j. It means the deeper the fusion layer is, the less effective its relative locations are. Since $D_{0,j}$ in RS-CNN increases rapidly, they have their limitation in gathering structure information from deep layers. In contrast, our CN rescales previous features by $D_{i-1,i}$, making the model much more capable to gather information of deep neural networks and bringing better performance compared to other fusion layers.

	layer1	layer2	layer3	layer4
Spherical Range (m)	0.8	1.6	3.2	6.4
α_1	0.148	0.721	1.392	1.295
α_2	0.686	0.868	0.293	0.143

Table 5. Condition parameters in different fusion layers. "Spherical Range" means the scale of structure information for this layer, which is usually expressed by the radius of the ball.

5.4 Effect of Condition Parameters

This ablation study is conducted on the KITTI dataset since the detection task can well reflect the model's ability in extracting multi-scale structure information. All AP results in this subsection are calculated in class "Car".

In our channel normalization, after rescaling previous features by $D_{i-1,i}$, we use two extra learnable condition parameters α_1 and α_2 to enable the model to gather structure information in a certain fusion layer. The consequence is the following. If previous structure information plays an important role in recognition, α_1 becomes large. Otherwise, if the current relative location works better, α_2 is greater.

We train a 4-layer 3DSSD with our CN module, and list the learned condition parameters in Table 5. In the first layer, since there is no structure information in previous features, α_2 is greater than α_1 to capture more relative location information in the current layer. In layer 2, since the mean size of a car in the KITTI dataset is $(l = 3.9m, h = 1.6m, w = 1.6m)$. With radius 1.6m, the layer covers the whole object. Therefore, α_2 is still greater than α_1 to capture features for the whole object. Differently, in layers 3 and 4, spherical ranges are much larger than those of cars, making α_1 much greater than α_2 to avoid these unnecessary features. We also compare the mAP performance between pure CN and CN with condition parameters. Their performances are 84.60% and 85.01% respectively which demonstrates the effectiveness of the condition parameters and manifests that always balancing is not optimal.

6 Conclusion

In this paper, we have analyzed the bottleneck of explicit fusion layers in SOTA hierarchical 3D recognition networks regarding extracting multilayer structure information. We provided "difference ratio" to measure the contribution of relative locations among different fusion layers on the final generated features. By comparing difference ratios among different SOTA methods, we find that when this factor is closer to 1, the learned features can maintain more structure information from different levels and extract more powerful 3D representations. Based on this observation, we developed a new technique "channel normalization", which enables the recognition models to fully exploit multilayer structure information and further boosts their performance without extra parameters.

References

1. Chen, Q., Sun, L., Wang, Z., Jia, K., Yuille, A.: Object as hotspots: An anchor-free 3d object detection approach via firing of hotspots (2019)
2. Chen, Y., Liu, S., Shen, X., Jia, J.: Fast point r-cnn. In: ICCV (2019)
3. Feng, Y., Zhang, Z., Zhao, X., Ji, R., Gao, Y.: GVCNN: group-view convolutional neural networks for 3d shape recognition. In: CVPR (2018)
4. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The KITTI dataset. I. J. Robotics Res. (2013)
5. Guo, H., Wang, J., Gao, Y., Li, J., Lu, H.: Multi-view 3d object retrieval with deep embedding network. IEEE Trans. Image Processing (2016)
6. Klokov, R., Lempitsky, V.S.: Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In: ICCV (2017)
7. Kuang, H., Wang, B., An, J., Zhang, M., Zhang, Z.: Voxel-FPN:multi-scale voxel feature aggregation in 3d object detection from point clouds. sensors (2020)
8. Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: Pointpillars: Fast encoders for object detection from point clouds. CVPR (2019)
9. Li, R., Li, X., Fu, C., Cohen-Or, D., Heng, P.: PU-GAN: a point cloud upsampling adversarial network. CoRR (2019)
10. Li, Y., Bu, R., Sun, M., Chen, B.: Pointcnn. CoRR (2018)
11. Liang, M., Yang, B., Wang, S., Urtasun, R.: Deep continuous fusion for multi-sensor 3d object detection. In: ECCV (2018)
12. Liu, X., Qi, C.R., Guibas, L.J.: Flownet3d: Learning scene flow in 3d point clouds. In: CVPR (2019)
13. Liu, Y., Fan, B., Meng, G., Lu, J., Xiang, S., Pan, C.: Densepoint: Learning densely contextual representation for efficient point cloud processing. In: ICCV (2019)
14. Liu, Y., Fan, B., Xiang, S., Pan, C.: Relation-shape convolutional neural network for point cloud analysis. In: CVPR (2019)
15. Liu, Z., Zhao, X., Huang, T., Hu, R., Zhou, Y., Bai, X.: Tanet: Robust 3d object detection from point clouds with triple attention. AAAI (2020)
16. Mao, J., Wang, X., Li, H.: Interpolated convolutional networks for 3d point cloud understanding. ICCV (2019)
17. Maturana, D., Scherer, S.: Voxnet: A 3d convolutional neural network for real-time object recognition. In: IROS (2015)
18. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum pointnets for 3d object detection from RGB-D data. CoRR (2017)
19. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: CVPR (2017)
20. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: NIPS (2017)
21. Shen, Y., Feng, C., Yang, Y., Tian, D.: Mining point cloud local structures by kernel correlation and graph pooling. In: CVPR (2018)
22. Shi, S., Wang, X., Li, H.: Pointrcnn: 3d object proposal generation and detection from point cloud. In: CVPR (2019)
23. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. (2014)
24. Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M., Kautz, J.: Splatnet: Sparse lattice networks for point cloud processing. In: CVPR (2018)

25. Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E.G.: Multi-view convolutional neural networks for 3d shape recognition. In: ICCV (2015)
26. Thomas, H., Qi, C.R., Deschaud, J., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: Flexible and deformable convolution for point clouds. ICCV (2019)
27. Wang, C., Samari, B., Siddiqi, K.: Local spectral graph convolution for point set feature learning. In: ECCV (2018)
28. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds. ACM Trans. Graph. (2019)
29. Wu, W., Qi, Z., Li, F.: Pointconv: Deep convolutional networks on 3d point clouds. In: CVPR (2019)
30. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. In: CVPR (2015)
31. Yan, Y., Mao, Y., Li, B.: Second: Sparsely embedded convolutional detection. Sensors (2018)
32. Yang, Z., Sun, Y., Liu, S., Jia, J.: 3dssd: Point-based 3d single stage object detector (2020)
33. Yang, Z., Sun, Y., Liu, S., Shen, X., Jia, J.: STD: sparse-to-dense 3d object detector for point cloud. ICCV (2019)