PointAlign: Towards Rotation Invariant Point Cloud Representation via Tangent Space Alignment

Anonymous CVPR submission

Paper ID 5911

Abstract

Point cloud is an important data structure in 3D vision applications that precisely describes the surface of objects using sampling points. Though we are witnessing dramatic growth of applications using point cloud, one of the fundamental problems lying in this kind of data is not properly solved, that is the variance of rotation both at overall and local levels of the data. This unexpected variance messes up the patterns in the data and makes the underlying networks unnecessarily hard to train. In this paper, we propose a convolution strategy that remedies this problem by aligning the points in the local tangent space lying in the whole points manifold while preserving relative poses among different local spaces. By hierarchically applying this method, the variance of rotation is reduced layer by layer and an overall rotation-invariant result can be obtained in the final output. Furthermore, this method is designed as a flexible building block named PointAlign that can be easily applied to many existing networks without any extra learnable parameters in itself. Both theoretical and empirical analyses are provided to demonstrate our method is effective to achieve rotation invariant results and is highly efficient in terms of data utilization.

1. Introduction

Point cloud is a promising data structure representing 3D objects' surfaces by unordered sample points set. It is widely used in many industrial scenarios and many high-performance models have been proposed to deal with it. Nevertheless, the rotational variance problem that lies in point cloud data is often ignored by state-of-the-art methods, which makes the pattern of the data unnecessarily hard to learn. This variance commonly exists at different levels and scales on the point cloud surfaces (Figure 1 top). At the object level, due to the uncertain viewpoint, the target object may be in different rotational poses. At the local level, located at different sides of an object, similar local structures such as corners or planes may vary in orientation. After rotation, a bottle remains a bottle, and a corner is still a corner. However, without explicitly addressing the rotational variance problem, many existing methods have to treat them as a set of separated training samples, ignoring their unchanged symmetric properties under rotation transformations. Rotational variance does mess up the patterns of the input data, making the training process unnecessarily inefficient. The desired invariant result cannot be guaranteed even with a tremendous amount of training data.

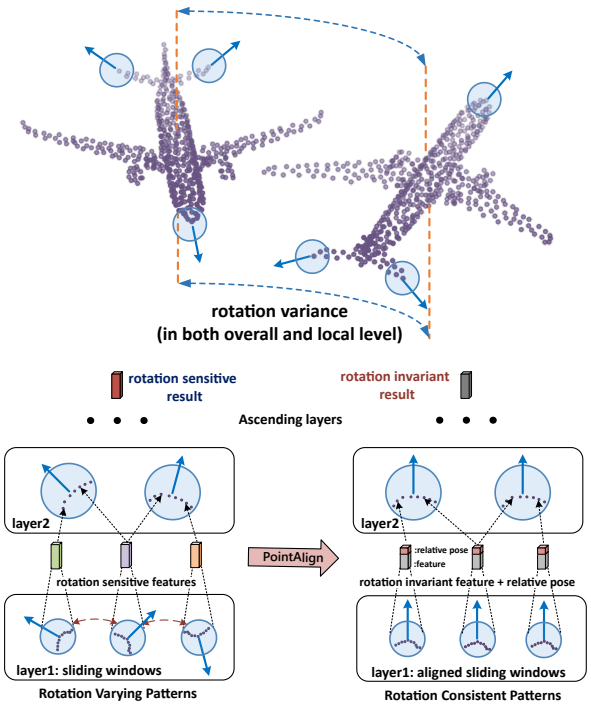

Figure 1: Rotation variance existing in both local and overall levels. The blue circles denote the sliding windows (clusters) of similar local structures that vary in orientation. PointAlign reduces this variance by hierarchically aligning the local points without losing relative pose information by pose encoding. A rotation invariant result is obtained in the final output.

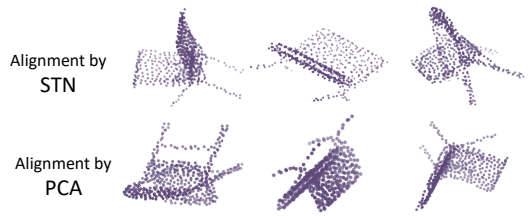

Figure 2: STN (spatial transform network[10]) and PCA fail to align the objects right. The STN here is the point cloud transform block in DGCNN[21]. We train it on a 360° augmented dataset. For the PCA alignment, the minimal eigenvalue's eigenvector is used as the orientation vector.

To tackle this issue, researchers have developed many strategies. They can be coarsely categorized into three major types (none of them has dominated the practice) : (1) Handcrafted features that are invariant to rotation. (2) Convolution that guarantees an invariant result (3) Orientation Alignment. For the first kind of strategy, a carefully designed feature can produce a rotation-invariant result [2] but can not be end-to-end optimized, and the overall performance highly depends on the design of the feature. It's also hard to learn a deep feature based on this feature, since the rotation-invariance may not hold with other types of input. For the second kind of strategy, the convolution operator is specially designed to be equivariant or steerable under rotation transformation. Though an overall invariant result can be obtained in the final layer, these special convolutions either constrain the expression ability of the convolution filters or require a complicated and carefully designed network structure. For the third kind of strategy, some works try to reduce the rotation variance by aligning the orientation of objects globally with some carefully designed algorithms or massive data trained deep models. However, it is theoretically difficult to do this perfectly, since the canonical orientation of a 3D object is hard to define and sometimes even ambiguous. For example, how to tell the correct direction of a cell phone? Portrait or landscape, or just lying down. As an intuitive example shown in Figure 2, many existing algorithms are therefore unstable in practice.

Considering the drawbacks of handcrafted features and special convolutions mentioned above, we try to develop a method that can be flexibly inserted into existing networks without requiring special network design. Based on the fact that most deep models dealing with point cloud are using hierarchical local feature extraction in a sliding window (cluster) way and the local points are easier to align than the global ones, we achieve the rotation invariant result by local points alignment and relative pose restoring. As we can see at the bottom of Figure 1, our method, named PointAlign, locally aligns the points according to their tangent space(the plane orthogonal to its normal) where most of the points are lying around it due to the local smoothness of point cloud surface. This alignment makes the local clusters (sliding windows) invariant under object rotations and thus reduces the variance of rotation at the local level. Furthermore, to obtain a correct result, we need to restore the relative pose. In this paper, we achieve this by encoding relative pose into the feature fed to following layers. Both the feature learned from the aligned clusters and the encoding of the relative pose are invariant to object rotation. Hence, the invariant property is kept layer by layer and a rotation-invariant result in the final output is obtained. Our method is supposed to be effective to achieve a rotation-invariant result and efficient in terms of data utilization by reducing the variance of data in both local and overall levels. To demonstrate this, both theoretical and empirical analysis is provided in this paper. In summary, the contributions of this paper are as follows:

- We propose a flexible building block (PointAlign) which achieves the rotation invariant result and efficient data utilization by reducing the local rotation variance layer by layer. It is compatible with many existing deep models adopting hierarchical local feature extraction.
- By applying PointAlign to PointNet++[19] and RS-CNN [16], a stable and superior performance is achieved in rotated data benchmarks.

2. Related Works

2.1. Point Cloud

Point cloud is a promising data structure which provides precise object surface sample points in the raw data form of LiDARs, structural lights and many other widely used 3D sensors in industrial applications. Though easy to gather, the raw point cloud data is irregular and sparse in the 3-dimension space. This makes some variance such as rotation transformation severer than that in 2D images. Point cloud based methods potentially require more training data. As the proliferation of deep learning, we have witnessed a tremendous amount of methods been developed utilizing the large data of point cloud to do many complex tasks such as object classification, segmentation, detection, etc [18, 19, 11, 15, 21]. However, large, richly-labeled datasets are expensive and hard to produce, and methods that can reduce or properly model this variance are eagerly demanded.

2.2. Tackling the Problem of Rotation Variance

The rotation variance is common in point cloud data both at local and whole-object level. The whole-object level variance makes the underlying networks hard to learn a rotation-invariant result while the local level variance makes many local weight sharing models inefficient to learn. In literature, some methods have been developed to tackle this variance of rotation, we will go through these methods.

2.2.1 Handcrafted features

Handcrafted features are widely investigated and used before the era of deep learning. Facing the challenge of the variance of rotation, researchers have come up with methods using this old fashion. ClusterNet[2] designs a feature that encodes the relative pose of local clusters (analog to sliding windows in 2D CNNs [13, 12]) before the input of a deep network. This handcrafted feature is proven to be rotation-invariant in object-level and effectively encodes the pose of points without losing the pose relation information among clusters. However, this method has many limitations such as relying on the original point selection, relying on the local feature orders, etc. To make the final network output invariant to rotation, some useful elements that are rotation varied are not allowed to use. This includes the local relative position widely used in PointCNN[15], PointNet++[19], RS-CNN[16], PCNN[1], PointConv[25]. It plays the role of pixel grid in 2D CNNs and characterizes the local geometric relations among features. In the future, more complicated handcrafted features are supposed to develop, they should concern more about the compatibility with the most majorities of existing networks.

2.2.2 Rotation Equivariant/Invariant Convolution

One way to achieve rotation invariance is to design a convolution operation that generates rotation equivariant result for the intermediate layers of deep networks. This idea is mainly achieved by G-CNNs, the extension of conventional CNNs using group theory. Regular G-CNN obtains a rotation equivariant result in the first layer by copying the conventional kernel(filters) in different orientations. A group convolution is used to keep the rotation-equivariance property in ascending layers. In the last layer, a rotation-invariant result is achieved by coset pooling. Regular G-CNN[3] is an important attempt to achieve the rotation-invariant result which properly models the variance of the rotation in the whole object level. This method has inspired many works to obtain a rotation-invariant result in different domains such as Hexaconv[8], Spherical CNNs[5], CubeNet[23], etc.

Regular G-CNN achieving the equivariant result is implemented by the filters copying, which is discrete and cannot produce a continuous result. It is also hard to apply to point cloud data which is not aligned in any regular grid. Without this exhaustive and problematic copying, steerable G-CNN [4, 24, 22] represents features in terms of more general fields over a homogeneous space and achieves a continuous rotation-equivariant result for intermediate layers. A rotation-invariant result in the object-level is achieved in the final layer by carefully selected convolution layers' fusion. Though achieving convincing results, it has been theoretically proven that if we want to make the linear convolution layers equivariant to rotation, the convolution kernel must be the linear combination of some irreducible bases [22, 4], which means this kind of method constrains the convolution kernels in a feature space expanded by some irreducible bases. To maintain the equivariant properties in between layers, carefully designed activation functions and other equivariance preserving blocks are required, limiting the flexibility. Based on the idea of steerable CNNs, 3D Steerable CNN[22] learns an equivariant feature for volumetric data in SE(3) (the group of rigid transformation). Tensor Field Network [6] achieves translation-, permutation- and rotation- equivariance for point cloud data representation. Beyond the framework of G-CNN, some rotation invariant convolutions are developed. For example, RIConv [29] achieves the invariant result effectively by constructing local rotation-invariant features, but they may not fully characterize the local geometry and the relative pose among sliding windows is not explicitly modeled.

2.2.3 Orientation Alignment

Another way to achieve the rotation invariant result is to align the object to a canonical orientation. PCA can be used to align an object according to its principle eigenvectors' directions. However, these eigenvectors are computed from the object points' covariance matrix, which does not fully characterize the geometric properties of an object. Spatial transformer network (STN)[10], first introduced in 2D CNNs, tries to learn the canonical orientation of the input object in an unsupervised manner. Though achieving some intuitive results in some datasets such as MNIST[14] and SVHN[17], it does not guarantee the objects can be correctly aligned. Due to its unsupervised property, many deep models[18, 21] adopt STN as a branch of their networks. Supervised orientation learning methods have also been developed in some tasks such as 3D pose estimation, 3D bounding box detection, etc. However, manually labeled data is expensive and hard to obtain. More complicated and better methods are expected to be developed, but aligning objects in the whole object level may be a long-standing difficulty since it's hard to find an algorithm that generates consistent global alignment results for different instances of the same category. However, local orientation alignment may be much easier. Taking this into consideration, [20] defines an interpolated convolution operating on the aligned local tangent space. Though achieving invariant results, it does not explicitly model the relative pose among clusters and requires special convolution operation.

3. Method

For point cloud data, points are distributed on the surfaces of objects. Similar to the successful 2D CNNs, many works are learning features hierarchically from local clus-

ters (sliding windows) located on this surface. A problem here is that clusters similar in structure may vary in orientations, which makes the local learning subnetworks unnecessarily inefficient to train. By aligning the tangent space of different local clusters (sliding windows) hierarchically, we achieve both efficient training and overall rotation-invariant result.

In the following part of this section, we first define the underlying networks that our method is compatible with. Then, the formal definition and theoretical analysis are given in the latter part of this section.

3.1. Compatible Models

We find that many existing deep models dealing with point cloud data can be recognized by a framework of generalized convolution given by the following formula, which means in each layer we use the local learning block h_θ to extract a new feature $f^{(out)}$ out of a position x_i based on the local input elements $f^{(in)}$ in a sliding window (local cluster):

$$f_{x_i}^{(out)} = h_\theta \left(\left\{ f_{x_j}^{(in)} \right\}_{x_j \in \mathcal{N}(x_i)} \right) \tag{1}$$

where $\mathcal{N}$ denotes the elements in the sliding window located at x_i. This is usually determined by kNN or ball query in point cloud networks and by pixel grid neighbors in conventional CNNs. In this formula, it should be noticed that $f^{(out)}$ may not be the input of the next convolutional layer's input $f^{(in)}$. Some non-convolutional layers can be inserted between them, such as the layers to extract the edge features with its corresponding node x_j [21], the layers to do batch normalization [9], etc.

Based on this framework, we need to explicitly separate the terms that are required to align, leaving the input feature rotation agnostic. Most works are using relative coordinates r inside a local cluster which indicates the spatial distribution of local features. This term is critically important, without which the underlying local subnetworks may fail to extract a good feature characterizing the local points structure. Without this term, the network may be more vulnerable and less convincing, since we can change the spatial distribution of local features without affecting the output result. Considering this importance, we explicitly add this term to our framework. Moreover, to be more compatible, we also add the normal of points $\mathbf{n}_{x_j}$ to it.

$$f_{x_i}^{(out)} = h_\theta \left(\left\{ r_{x_j}, \mathbf{n}_{x_j}, f_{x_j}^{(in)} \right\}_{x_j \in \mathcal{N}(x_i)} \right) \tag{2}$$

where $r_{x_j} = x_j - x_i$ denotes the local position or relative position of x_j. In practice, this relative position may be adequately enough to describe the local structure. However, to cope with more methods that use global coordinates in each local cluster. We further add the global term into our framework:

$$f_{x_i}^{(out)} = h_\theta \left(\left\{ x_j, r_{x_j}, \mathbf{n}_{x_j}, f_{x_j}^{(in)} \right\}_{x_j \in \mathcal{N}(x_i)} \right) \tag{3}$$

It should be noted that we add this global coordinates term is just to make our framework more compatible with existing networks. The global term itself is not recommended since it may hinder the efficient learning of sharing weight local learning blocks h_θ. Moreover, problems such as the translation sensitiveness may be introduced by this global term. In literature, we even see that some works try to use the global coordinate x_j as an alternative of relative position r_{x_j}. It is also not recommended since the sharing weight local learning block h_θ can not learn the relative position in some networks where the cluster $\mathcal{N}(x_i)$ does not contain the center point x_i itself.

In this subsection, we have defined the compatible networks we are going to cope with, in the following part, we will give the formal definition of our method.

3.2. PointAlign: via Tangent Space Alignment

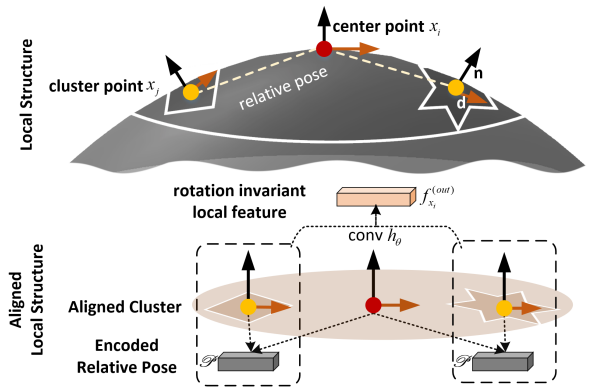

Figure 3: Demonstration of PointAlign. The local clusters(sliding windows) are aligned according to their tangent spaces while the relative pose among clusters is preserved by encoding the relative pose of the cluster points x_j and center point x_i.

The tangent space of a surface point x_i is regarded as the plane orthogonal to its normal $\mathbf{n}_{x_i}$. We can find that in most smooth places, the points in the local cluster are distributed near this plane. If we can align this plane, then we can reduce the variance of rotation at the local level. To do this, we first denote the unit vector of x and z axes as $\mathbf{i} = (1, 0, 0)^T$ and $\mathbf{k} = (0, 0, 1)^T$. Our method tries to align all the tangent spaces containing points in local clusters to a canonical orientation. Without losing generality, we choose the direction of $\mathbf{k}$. Denoting the rotation matrix $\mathcal{R}^{\mathbf{n}_{x_i} \to \mathbf{k}}$ that is determined by the unit vector $\mathbf{n}_{x_i}$ and $\mathbf{k}$, we can align

the local points in the following way: (We'll provide an efficient way to compute the rotation matrix $\mathcal{R}^{\mathbf{n}_{x_i}\to\mathbf{k}}$ in the supplementary materials.)

$$r'_{x_j} = \mathcal{R}^{\mathbf{n}_{x_i}\to\mathbf{k}}\, r_{x_j} \tag{4}$$

This formula aligns the tangent space in the normal direction which reduces much variance of rotation. Without this alignment, the underlying sub-networks h_θ have to learn the local points varying in the full 3D space. Since most of the local points approximately lie around the tangent space, our method makes the learning space of underlying networks much smaller. Furthermore, to make a better alignment, we align the orientation inside the plane. It can be determined by the direction of the massive center inside a cluster:

$$m_{x_i} = \sum_{x_j\in\mathcal{N}(x_i)} \frac{r_{x_j}}{|\mathcal{N}(x_i)|} \tag{5}$$

$$\mathbf{d}^{(out)}_{x_i} = \frac{m_{x_i}}{||m_{x_i}||_2} \tag{6}$$

$$\mathbf{d}'_{x_i} = \frac{\mathbf{proj_{xy}}\ \mathcal{R}^{\mathbf{n}_{x_i}\to\mathbf{k}}\ \mathbf{d}^{(out)}_{x_i}}{||\mathbf{proj_{xy}}\ \mathcal{R}^{\mathbf{n}_{x_i}\to\mathbf{k}}\ \mathbf{d}^{(out)}_{x_i}||_2} \tag{7}$$

where m_{x_i} denotes the local massive center of cluster $\mathcal{N}(x_i)$. $\mathbf{d}^{(out)}_{x_i}$ denotes the unit vector of the massive center. $.^{(out)}$ is used to identify the variables that are passed among layers. $\mathbf{d}'_{x_i}$ is the aligned unit massive center which is calculated by aligning $\mathbf{d}^{(out)}_{x_i}$ and projecting onto xy plane. It should be noted that the projection $\mathbf{proj_{xy}}$ can be simply implemented by setting the z component to zero without the explicit matrix multiplying. Obtaining $\mathbf{d}'$, we can we align the tangent space in its plane by the rotation matrix $\mathcal{R}^{\mathbf{d}'_{x_i}\to\mathbf{i}}$ determined by $\mathbf{d}'_{x_i}$ and $\mathbf{i}$.

$$\bar{r}_{x_j} = \mathcal{R}^{\mathbf{d}'_{x_i}\to\mathbf{i}}\, r'_{x_j} \tag{8}$$

Furthermore, we can align the local normal vector $\bar{\mathbf{n}}_{x_j}$:

$$\bar{\mathbf{n}}_{x_j} = \mathcal{R}^{\mathbf{d}'_{x_i}\to\mathbf{i}}\ \mathcal{R}^{\mathbf{n}_{x_i}\to\mathbf{k}}\ \mathbf{n}_{x_j} \tag{9}$$

As we discussed before, some works are using global coordinates as every layer's input which unfortunately cannot be perfectly aligned since they are likely to introduce variance for similar local structures located in different positions. We are not sure if we can simply discard this global term or change it to the relative one since the networks may prefer learning features globally to sharing knowledge among local clusters (sliding windows). Here, we provide a simple alignment for the global coordinates to make sure it's unchanged when the whole object is rotated against the origin (but cannot reduce the local level rotation variance):

$$\bar{x}_j = \mathcal{R}^{\mathbf{d}'\to\mathbf{i}}\ \mathcal{R}^{\mathbf{n}_{x_i}\to\mathbf{k}}\ \bar{x}_j \tag{10}$$

After the alignment, to make the result correct and preserve relative information, we need to restore the relative pose information among clusters for the following layers. Though each pair of clusters has a relative pose, it suffices to restore the relative pose only between the next layer's center x_i and the peripheral x_j. Taking this into consideration, we encode the relative pose into the feature $f^{(in)}_{x_j}$ of the cluster points. Here, for simplicity, we just concatenate them (denoted by $\oplus$), the underlying deep models are responsible to learn a better feature. More sophisticated encoding functions may be developed, but it is beyond the concerns of this paper. Following, we give the formal definition of this procedure:

$$\mathscr{P}^{normal}_{x_j} = \bar{\mathbf{n}}_{x_j} \tag{11}$$

$$\mathscr{P}^{plane}_{x_j} = \mathcal{R}^{\mathbf{d}'\to\mathbf{i}}\ \mathcal{R}^{\mathbf{n}_{x_i}\to\mathbf{k}}\ (\mathbf{d}^{(in)}_{x_j} - \mathbf{d}^{(out)}_{x_i}) \tag{12}$$

$$\tilde{f}^{(in)}_{x_j} = f^{(in)}_{x_j} \oplus \mathscr{P}^{normal}_{x_j} \oplus \mathscr{P}^{plane}_{x_j} \tag{13}$$

where we represent the relative pose of x_j in a cluster by two independent components, $\mathscr{P}^{normal}_{x_j}$ and $\mathscr{P}^{plane}_{x_j}$. $\mathscr{P}^{normal}_{x_j}$ encodes the relative rotation of normal which is function of $\mathbf{n}_{x_i}$ and $\mathbf{n}_{x_j}$. For simplicity, we choose $\bar{\mathbf{n}}_{x_j}$ to represent it. $\mathscr{P}^{plane}_{x_j}$ encodes the relative rotation in the plane orthogonal to the normal vector which can be determined by $\mathbf{d}^{(in)}_{x_j}$ and $\mathbf{d}^{(out)}_{x_i}$. Here we just use the vector subtraction to encode it. Finally, we can use these aligned components and relative pose augmented feature feeding for the underlying learning blocks: (the overall procedure is vividly depicted in Figure 3)

$$f^{(out)}_{x_i} = h_\theta\left(\left\{\bar{x}_j, \bar{r}_{x_j}, \bar{\mathbf{n}}_{x_j}, \tilde{f}^{(in)}_{x_j}\right\}_{x_j\in\mathcal{N}(x_i)}\right) \tag{14}$$

It should be noted that in the training process, the gradients of components computed outside the learning block h_θ, such as $\mathscr{P}$, $\bar{r}_{x_j}$ can be pruned, since we do not need to optimize the pose information. Moreover, it should be minded that in the first layer, no relative pose is defined (single points do not have pose), $\tilde{f}^{(in)}_{x_j}$ simply equals $f^{(in)}_{x_j}$, which can be empty or some rotation invariant features of points such as the RGB colors, reflection rates, etc.

3.3. Effectiveness of Rotation Invariant Result

Although the aligning operation shown in section 3.2 seems to be complicated, the overall architecture is clear that our method decorrelates the absolute rotation transformation in each local cluster of every convolutional layer and preserves the relative rotational pose at the same time to produce a correct result. We can easily check that $f^{(out)}_{x_i}$ remains invariant to arbitrary input rotations by induction. The first input $\tilde{f}^{(in)}_{x_j}$ (usually an empty vector) is constraint to be invariant to arbitrary rotations, $\bar{x}_j$, $\bar{r}_{x_j}$, $\bar{\mathbf{n}}_{x_j}$ are all

Method	Input	Input size	z/z	z/SO3	SO3/SO3
PointNet[18]	xyz	1024×3	87.0	12.8	80.3
PointNet++[19]	xyz	1024×3	89.3	28.6	85.0
PointCNN[15]	xyz	1024×3	**91.3**	41.2	84.5
RS-CNN[16]	xyz	1024×3	88.5	31.4	81.0
Spherical CNN[7]	voxel	2×64^2	88.9	78.6	86.9
RIConv[29]	xyz	1024×3	86.5	86.4	86.4
ClusterNet[2]	xyz	1024×3	87.1	87.1	87.1
PointNet++(PointAlign)	xyz+normal(provided)	1024×6	88.3	88.3	88.5
PointNet++(PointAlign)	xyz+normal(estimated)	1024×6	88.9	88.9	88.8
RS-CNN(PointAlign)	xyz+normal(provided)	1024×6	88.6	88.6	88.5
RS-CNN(PointAlign)	xyz+normal(estimated)	1024×6	89.3	89.3	89.3
RS-CNN*(PointAlign)	xyz+normal(estimated)	1024×6	90.8	**90.8**	**90.7**

* Modified RS-CNN with four layers and double scale neighborhood (The original one is with three layers and single scale neighborhood).

Table 1: Experiments on ModelNet40[26] in three different (z/z, z/SO3 and SO3/SO3) train/test rotation setups. The top 4 methods are not explicitly designed to be rotation invariant, and we can see the dramatic performance drop in z/SO3 setting. Judging from the SO3/SO3 performance, it is also hard for them to perform well when fully augmented data is provided. The 3 methods in the middle are claimed to be rotation-invariant. As shown in the last five rows, PointNet++ and RS-CNN assembled with our method show superior and stable performance.

aligned terms which can not change under arbitrary rotations. Thus, the output feature $f_{x_i}^{(out)}$ remains invariant to arbitrary rotations. This property remains true through the layers, leading to an overall rotation invariant result in the last layer.

3.4. Data Efficiency

We have demonstrated we can obtain a convincing rotation invariant result by hierarchically applying our method. Furthermore, we can prove that our method can do this in a data-efficient way. This efficiency mainly comes from the fact that our method reduces the variance of rotation in both local and overall object levels. The underlying learning blocks can concentrate on learning the features without the disturbance of unrelated variance. This potentially reduces the demanding for massive augmented training data.

4. Experiments

Applying our method *PointAlign* to two mainstream networks, i.e. PointNet++ and RS-CNN, we conducted experiments to verify its effectiveness to achieve a robust rotation invariant result and efficiency in terms of data utilization.

4.1. Effectiveness of Rotation Invariant Property

We evaluate the effectiveness of our method in both object classification and part segmentation tasks. Following the work of [7] and [29], we test our method on datasets with rotation transformation in three experimental setups: (1) training and testing with azimuthal rotations (z/z), (2) training with azimuthal rotations and testing with arbitrary rotations (z/SO3), (3) training and testing with arbitrary rotations (SO3/SO3). The experiments with the z/SO3 setup characterize the rotation invariant properties of the comparing models. The other two setups are used as the control groups. The z/z experiments are used to check the baseline performance of the testing methods in well-aligned datasets while the SO3/SO3 experiments are used to demonstrate how well the methods can learn with a fully augmented dataset.

4.1.1 Experiments on Classification Tasks

To verify our method being effective to achieve a robust rotation invariant result, we first test our method on ModelNet40[26], a widely-used 3D object classification dataset consisting of 12,311 CAD models from 40 categories. Following Qi et al.[18, 19, 16], a train/test split of 9,843/2,468 is adopted. As for the input, our method requires the normal vector of input points, but this doesn't limit our method in datasets where the surface normal is provided. In practice, we can use PCA and many computer graphics algorithms to estimate the local normal vectors. Although ModelNet40 provides the surface normals, for comparison, we also test our method using the PCA estimated normal data. The experiment results, shown in Table 1, demonstrate that our method, assembled with either PointNet++ or RS-CNN, effectively achieves a rotation-invariant result and outperforms other rotation-invariant models by a large margin in z/SO3 and SO3/SO3 scenarios. Our method also has stable performance across three scenarios. This agrees with the theoretical analysis that our method is rotation invariant.

Detail Analysis of the Estimated Normal As mentioned before, our method does not rely on the normal provided by the dataset, as they can be estimated by the local points' distribution. To do this, we apply PCA to local clusters to extract them, where the ambiguous direction of the eigenvector of the PCA covariance matrix is not a problem since we can tell the back or front of a surface by viewpoint relations. To study the effectiveness of estimated normals, we set up experiments using different cluster configurations in Table 2. We can observe that our method on RS-CNN has consistent or even better performance with estimated normals than that with the provided normal. The best estimation is made by clustering with the same ball query radius as that of the first layer of our network (0.23). We consider that normal vectors estimated from clustered points (matched with the cluster of network input) better represent the local structure of the neighborhoods than the real normal which is defined on the mesh rather on the points.

Normal	acc. (z/SO3)
provided by dataset	88.6
k=32 kNN	87.1
k=64 kNN	88.3
r=0.23 ball query	**89.3**

Table 2: z/SO3 accuracy of RS-CNN(PointAlign) on the ModelNet40 object classification task with different sets of normal vectors. It shows we can obtain stable and even better normal vectors by simple estimating methods.

4.1.2 Experiments on Segmentation Tasks

A similar experiment is conducted on part segmentation task to verify the rotation invariant property of our method. ShapeNet Parts [28] is used in this experiments, which consists of 16,880 CAD models in 16 categories. Each model is annotated with 2-6 part labels. We follow the conventional train/test split of 14,006/2,874, and evaluate our method in z/SO3 setting. Following previous works, we report the standard evaluation metrics including mean IoU across all classes (Table 3,4) and IoU for each class. An intuitive visualization result is also provided in Figure 4. Our method is very effective to make a rotation-invariant result and achieves superior performance in z/SO3 scenario.

4.1.3 Implementation Details

We assemble PointNet++ and RS-CNN with our method. For PointNet++, it is simple to do this using the formula defined in section 3.2. We will give more details on RS-CNN. We adopt the hyperparameters and training settings from the original paper. The relative pose information is restored by concatenating them with the low-level relation h in RS-CNN. To demonstrate the effect of our method,

Method	Input	z/SO3
PointNet[18]	xyz	37.8
PointNet++[19]	xyz+normal	48.2
PointCNN[15]	xyz	34.7
DGCNN[21]	xyz	37.4
SpiderCNN[5]	xyz+normal	42.9
RS-CNN[16]	xyz	47.5
RIConv[29]	xyz	75.3
PointNet++(PointAlign)	xyz+normal	75.4
RS-CNN(PointAlign)	xyz+normal	**78.6**

Table 3: Part segmentation performed on ShapeNet part dataset [28] . We use the mean per-class IoU (mIoU, %) to measure the performance in z/SO3 scenario.

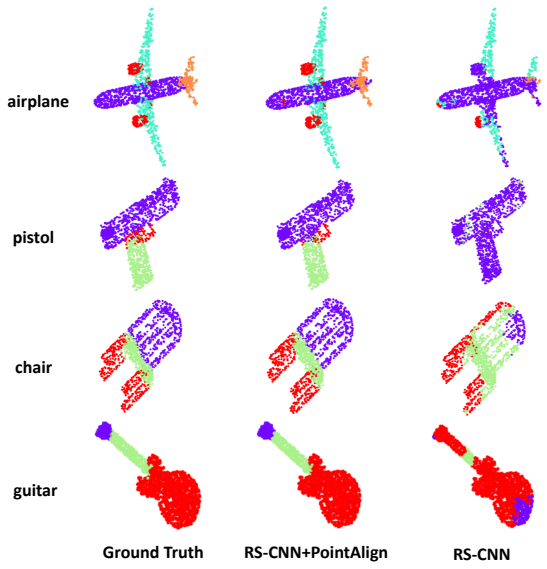

Figure 4: Visualization of some examples from part segmentation results. All models are trained with azimuthal augmented ShapeNet Parts dataset. As we can see, when the input objects are rotated in the SO3 group, the original RS-CNN generates unsatisfactory results while RS-CNN(PointAlign) performs well.

we conduct most experiments using the simplest RS-CNN network structure (single-scale neighbourhood, three layers and without global coordinates). Besides, we also tested with other configurations mentioned by the original paper, which showed stable performance improvement as the network went deeper and more complex. Moreover, without explicit specification, we report the performance of RS-CNN without global coordinates, with which the network shows minor performance difference but becomes sensitive to potential translation variance.

Network	aero.	bag	cap	car	chair	ear.	guitar	knife	lamp	laptop	motor.	mug	pistol	rocket	skate.	table
PointNet[18]	40.4	48.1	46.3	24.5	45.1	39.4	29.2	42.6	52.7	36.7	21.2	55.0	29.7	26.6	32.1	35.8
PointNet++[19]	51.3	66.0	50.8	25.2	66.7	27.7	29.7	65.6	59.7	70.1	17.2	67.3	49.9	23.4	43.8	57.6
PointCNN[15]	21.8	52.0	52.1	23.6	29.4	18.2	40.7	36.9	51.1	33.1	18.9	48.0	23.0	27.7	38.6	39.9
DGCNN[21]	37.0	50.2	38.5	24.1	43.9	32.3	23.7	48.6	54.8	28.7	17.8	74.4	25.2	24.1	43.1	32.3
SpiderCNN[27]	48.8	47.9	41.0	25.1	59.8	23.0	28.5	49.5	45.0	83.6	20.9	55.1	41.7	36.5	39.2	41.2
RS-CNN[16]	34.5	66.5	70.3	29.9	46.8	64.6	56.7	51.9	61.5	54.3	17.3	53.7	29.0	24.7	43.6	54.5
RIConv[29]	80.6	**80.0**	70.8	**68.8**	**86.8**	70.3	87.3	**84.7**	77.8	80.6	57.4	91.2	71.5	52.3	66.5	78.4
PointNet++(PointAlign)	79.7	76.6	80.7	65.1	85.3	68.7	89.1	81.2	79.2	79.3	52.3	91.6	73.1	54.6	71.9	78.1
RS-CNN(PointAlign)	**82.2**	78.5	**84.9**	67.0	86.6	**70.7**	**89.6**	81.2	**80.8**	**85.6**	**68.0**	**92.0**	**79.2**	**57.5**	**73.6**	**80.1**

Table 4: Per-class IoU of object part segmentation on ShapeNet parts dataset in z/SO3 scenario. Our method is effective to make the underlying networks performs well in rotated dataset.

4.1.4 Ablation Studies

As discussed in section 3.2, our method aligns the local point twice, first the alignment of tangent space normal (operated by $R^{\mathbf{n}_{x_i} \to \mathbf{k}}$) and second the alignment of tangent space plane (operated by $R^{\mathbf{d}'_{x_i} \to \mathbf{i}}$). In this subsection, we conduct ablation studies on these two alignments. All models are trained and tested in z/SO3 scenario on ModelNet40 dataset. As shown in Table 5, both alignments are critical to achieving a rotation invariant result. Moreover, it's interesting that without the first alignment, the second alignment alone actually worsens the performance (compare model A with C). This implies that the tangent space normal alignment, which eliminates most of the variance of rotation by reducing the three-dimensional space to an approximate plane, is of major importance.

model	align1	align2	acc. (z/SO3)
A			31.4
B	✓		63.5
C		✓	25.0
D	✓	✓	**89.3**

Table 5: Ablation study of our method applied on RS-CNN. align1 means the first tangent space aligning with $R^{\mathbf{n}_{x_i} \to \mathbf{k}}$, and align2 means the second aligning with $R^{\mathbf{d}'_{x_i} \to \mathbf{i}}$

4.2. Data Efficiency

Besides the strong performance of our method on rotation augmented benchmarks, by reducing the variance of rotation, our method is supposed to be efficient in terms of data utilization. To verify this, we train the models in datasets with different data augmentation strengths and test the models in different strengths too. This strength is measured by the rotation angle range, from which the rotation angles around three axes are sampled uniformly. The experiment results, as shown in Figure 5, demonstrate that RS-CNN not assembled with our method fails to generalize in unseen cases when the augmentation is not strong enough, while our method makes the underlying model stable even without any augmentation. This shows our method is efficient in terms of data.

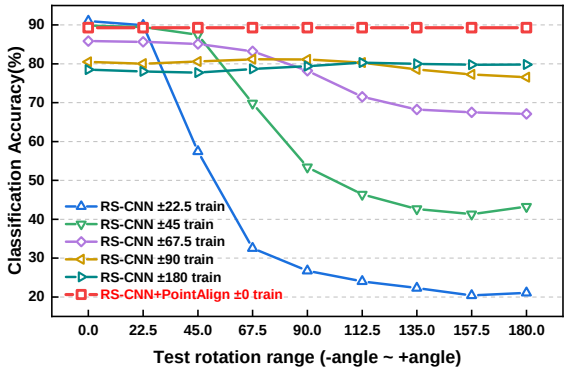

Figure 5: Experiments on ModelNet40 of different rotation augmentation settings. In the legend area, $\pm angle$ denotes the augmentation range from $-angle$ to $+angle$.

5. Conclusion

In this paper, we propose a flexible plugin, i.e. PointAlign, which is compatible with many existing models. Bypassing the difficulties of global alignment, it aligns the local points in tangent spaces instead, which well preserves the relative pose at the same time. This method effectively reduces the rotation variance in point cloud data at both local and overall levels. This not only helps us achieve the rotation invariant result but also enables the underlying networks to concentrate on learning patterns without task-unrelated disturbance, thus ridding the networks of massive augmented data and leading to a better generalization. The variance of data is common in many datasets, behind which there may be some invariant and symmetric properties lying. In future works, to make the deep models less demanding for large data, we are obligated to mine more symmetric properties and explicitly reduce more kinds of variance.

References

[1] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Trans. Graph.*, 37:71:1–71:12, 2018. 3

[2] Chao Chen, Guanbin Li, Ruijia Xu, Tianshui Chen, Meng Wang, and Liang Lin. ClusterNet: Deep Hierarchical Cluster Network With Rigorously Rotation-Invariant Representation for Point Cloud Analysis. In *CVPR*, 2019. 2, 3, 6

[3] Taco Cohen and Max Welling. Group Equivariant Convolutional Networks. In *International conference on machine learning*, pages 2990–2999, 2016. 3

[4] Taco Cohen and Max Welling. Steerable CNNs. *ArXiv*, abs/1612.08498, 2017. 3

[5] Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *International Conference on Learning Representations*, 2018. 3, 7

[6] Philippe Corboz and Frédéric Mila. Tensor Network Study of the Shastry-Sutherland Model in Zero Magnetic Field. *Physical Review B*, 87(11):115144, 2013. 3

[7] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) Equivariant representations with Spherical CNNs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–68, 2018. 6

[8] Emiel Hoogeboom, Jorn W.T. Peters, Taco S. Cohen, and Max Welling. HexaConv. In *International Conference on Learning Representations*, 2018. 3

[9] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015. 4

[10] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial Transformer Networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015. 2, 3

[11] Mingyang Jiang, Yiran Wu, and Cewu Lu. Pointsift: A Sift-like Network Module for 3D Point Cloud Semantic Segmentation. *arXiv preprint arXiv:1807.00652*, 2018. 2

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 3

[13] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In *NIPS*, 1989. 3

[14] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. 3

[15] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution On X-Transformed Points. In *Advances in Neural Information Processing Systems 31*, pages 820–830. Curran Associates, Inc., 2018. 2, 3, 6, 7, 8

[16] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-Shape Convolutional Neural Network for Point Cloud Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019. 2, 3, 6, 7, 8

[17] Yuval Netzer, Tiejie Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, and Andrew Y. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. 2011. 3

[18] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. 2, 3, 6, 7, 8

[19] 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 5105–5114, 2017. 2, 3, 6, 7, 8

[20] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent Convolutions for Dense Prediction in 3D. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018. 3

[21] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics (TOG)*, 2019. 2, 3, 4, 7, 8

[22] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data. In *Advances in Neural Information Processing Systems*, pages 10402–10413, 2018. 3

[23] Daniel E. Worrall and Gabriel J. Brostow. CubeNet: Equivariance to 3D Rotation and Translation. In *ECCV*, 2018. 3

[24] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Harmonic Networks: Deep Translation and Rotation Equivariance. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7168–7177, 2017. 3

[25] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019. 3

[26] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. 6

[27] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018. 8

[28] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qi-Xing Huang, Alla Sheffer, and Leonidas J. Guibas. A Scalable Active Framework for Region Annotation in 3D Shape Collections. *ACM Trans. Graph.*, 35:210:1–210:12, 2016. 7

[29] Zhiyuan Zhang, Binh-Son Hua, David W. Rosen, and Sai-Kit Yeung. Rotation Invariant Convolutions for 3D Point Clouds Deep Learning. In *International Conference on 3D Vision (3DV)*, 2019. 3, 6, 7, 8