Effective Rotation-invariant Point CNN with Spherical Harmonics kernels

Adrien Poulenard, Marie-Julie Rakotosaona, Yann Ponty, Maks Ovsjanikov

Effective Rotation-invariant Point CNN with Spherical Harmonics kernels

Adrien Poulenard
LIX, Ecole Polytechnique
adrien.poulenard@inria.fr

Marie-Julie Rakotosaona
LIX, Ecole Polytechnique
mrakotos@lix.polytechnique.fr

Yann Ponty
LIX, Ecole Polytechnique
yann.ponty@lix.polytechnique.fr

Maks Ovsjanikov
LIX, Ecole Polytechnique
maks@lix.polytechnique.fr

Abstract

We present a novel rotation invariant architecture operating directly on point cloud data. We demonstrate how rotation invariance can be injected into a recently proposed point-based PCNN architecture, at all layers of the network, achieving invariance to both global shape transformations, and to local rotations on the level of patches or parts, useful when dealing with non-rigid objects. We achieve this by employing a spherical harmonics based kernel at different layers of the network, which is guaranteed to be invariant to rigid motions. We also introduce a more efficient pooling operation for PCNN using space-partitioning data-structures. This results in a flexible, simple and efficient architecture that achieves accurate results on challenging shape analysis tasks including classification and segmentation, without requiring data-augmentation, typically employed by non-invariant approaches.

1. Introduction

Analyzing and processing 3D shapes using deep learning approaches has recently attracted a lot of attention, inspired in part by the successes of such methods in computer vision and other fields. While early approaches in this area relied on methods developed in the image domain, e.g. by sampling 2D views around the 3D object [21], or using volumetric convolutions [27], recent methods have tried to directly exploit the 3D structure of the data. This notably includes both mesh-based approaches that operate on the surface of the shapes [15, 16], and point-based techniques that only rely on the 3D coordinates of the shapes without requiring any connectivity information [19, 20].

Point-based methods are particularly attractive, since they are both very general and often more efficient than volumetric or mesh-based methods as they do not require maintaining complex and expensive data-structures. As a result, starting from the seminal works on PointNet [19] and PointNet++ [20], many point-based learning approaches have been proposed, often achieving remarkable accuracy in tasks such as shape classification and segmentation among many others. A key challenge when applying these methods in practice, however, is to ensure *invariance* to different kinds of transformations, and especially to rigid motions. Common strategies include either using spatial transformer blocks [10] as done in the original PointNet architecture and its extensions, or applying extensive *data augmentation* during training to learn invariance from the data. Unfortunately, when applied to shape collections that are not pre-aligned, these solutions can be either expensive, requiring unnecessarily long training, or incomplete in cases when *local* rotation invariance is required, e.g. for non-rigid shapes, undergoing articulated motion, which is difficult to model through data augmentation alone.

In this paper, we propose a different approach for dealing with both global and local rotational invariance for point-based 3D shape deep learning tasks. Instead of learning invariance from data, we propose to use a different kernel that is theoretically guaranteed to be invariant to rotations, while remaining informative. To achieve this, we leverage the recent PCNN by extension operators [2], which provides an efficient framework for point-based convolutions. We extend this approach by introducing a rotationally invariant kernel and making several modifications for improved efficiency. We demonstrate on a range of difficult experiments that our method can achieve high accuracy directly, without relying on data augmentation.

2. Related work

A very wide variety of learning-based techniques have been proposed for 3D shape analysis and processing. Below we review methods most closely related to ours, focusing on point-based approaches, and various ways of incorporating rotational invariance and equivariance in learning. We

refer the interested readers to several recent surveys, e.g. [28, 16], for a more in-depth overview of geometric deep learning methods.

Learning in Point Clouds. Learning-based approaches, and especially those based on deep learning, have recently been proposed specifically to handle point cloud data. The seminal PointNet architecture [19] has inspired a large number of extensions and follow-up works, notably including PointNet++ [20] and Dynamic Graph CNNs [23] for shape classification and segmentation, and more recently PCPNet [7] for normal and curvature estimation, PU-Net [30] for point cloud upsampling, and PCN for shape completion [31] among many others.

While the original PointNet approach is not based on a convolutional architecture, instead using a series of classic MLP fully connected layers, several methods have also tried to define and exploit meaningful notions of convolution on point cloud data, inspired by their effectiveness in computer vision. Such approaches notably include: basic pointwise convolution through nearest-neighbor binning and a grid kernel [9]; Monte Carlo convolution, aimed at dealing with non-uniformly sampled point sets [8]; learning an $\mathcal{X}$-transformation of the input point cloud, which allows the application of standard convolution on the transformed representation [14]; and using extension operators for applying point convolution [2]. These techniques primarily differ by the notion of neighborhood and the construction of the kernels used to define convolution on the point clouds. Most of them, however, share with the seminal PointNet architecture the lack of support for invariance to rigid motions, mainly because the kernels are applied to point coordinates, and defining invariance on the level of a single point is generally not meaningful.

Invariance to transformations. Addressing invariance to various transformation classes has been considered in many areas of Machine Learning and Geometric Deep Learning in particular. Most closely related to ours are approaches based on designing *steerable filters* which can learn representations that are equivariant to the rotation of the input data [25, 24, 1, 26]. A particularly comprehensive overview of the key ideas and results in this area is presented in [13]. In closely related works, Cohen and colleagues have proposed group equivariant networks [5] and rotation-equivariant spherical CNNs [6]. While the theoretical foundations of these approaches are well-studied, in the context of 3D shapes, they have primarily been applied to either volumetric [24] or spherical (e.g. by projecting shapes onto an enclosing sphere via ray casting) [6] representations. Instead, we apply these constructions directly in an efficient and powerful point-based architecture.

Perhaps most closely related to ours are two very recent unpublished methods, [29, 22], that also aim to introduce invariance into point-based networks. Our approach is different from both, since unlike the PRIN method in [29] our convolution operates directly on the point clouds and avoids the construction of spherical voxel space. As we show below, this gives our method greater invariance to rotations and higher accuracy. At the same time while the authors of [22] explore similar general ideas and describe related constructions, including the use of spherical harmonics kernels, they do not describe a detailed architecture, and only show results with dozens of points (the released implementation is also limited to toy examples), rendering both the method and its exact practical utility unclear. Nevertheless, we stress that both [29] and [22] are very recent unpublished methods and thus concurrent to our approach.

Contribution: Our key contributions are as follows:

1. We develop an effective rotation invariant point-based network. To the best of our knowledge, ours is first such method achieving higher accuracy than PointNet++ [20] *with data augmentation* on a range of tasks.
2. We significantly improve the efficiency of PCNN by Extension Operators [2] using space partitioning.
3. We demonstrate the efficiency and accuracy of our method on tasks such as shape classification, segmentation and matching on standard benchmarks and introduce a novel dataset for RNA molecule segmentation.

3. Background

In this section we first introduce the main notation and give a brief overview of the PCNN approach [2] that we use as a basis for our architecture.

3.1. Notation

We use the notation from [2]. In particular, we use tensor notation: $a \in \mathbb{R}^{I\times I\times J\times L\times M}$ and the sum of tensors $c = \sum_{ijl} a_{ii'jlm} b_{ijl}$ is defined by the free indices: $c = c_{i'm}$. $C(\mathbb{R}^3, \mathbb{R}^K)$ represent the collections of volumetric functions $\psi : \mathbb{R}^3 \to \mathbb{R}^K$.

3.2. PCNN

The PCNN framework consists of three simple steps. First a signal is extended from a point cloud to $\mathbb{R}^3$ using an extension operator $\mathcal{E}_X$. Then standard convolution on volumetric functions O is applied. Finally, the output is restricted to the original point cloud with a restriction operator $\mathcal{R}_X$. The final convolution on point clouds is defined as:

$$O_X = \mathcal{R}_X \circ O \circ \mathcal{E}_X \tag{1}$$

In the original work [2], the extension operators and kernel functions are chosen so that the composition of the three operations above, using Euclidean convolution in $\mathbb{R}^3$ can be computed in closed form.

Extension operator. Given an input signal represented as J real-valued functions $f \in \mathbb{R}^{I\times J}$ defined on a point cloud, it can be extended to $\mathbb{R}^3$ via a set of volumetric basis functions $l_i \in C(\mathbb{R}^3, \mathbb{R})$ using the values of f at each point f_i. The extension operator $\mathcal{E}_X : \mathbb{R}^{I\times J} \to C(\mathbb{R}^3, \mathbb{R}^J)$ is:

$$(\mathcal{E}_X[f])_j(x) = \sum_i f_{ij} l_i(x) \tag{2}$$

The authors of [2] use Gaussian basis functions centered at the points of the point cloud so that the number of basis functions equals to the number of points.

Convolution operator. Given a kernel $\kappa \in C(\mathbb{R}^3, \mathbb{R}^{J\times M})$, the convolution operator $O : C(\mathbb{R}^3, \mathbb{R}^J) \to C(\mathbb{R}^3, \mathbb{R}^M)$ applied to a volumetric signal $\psi \in C(\mathbb{R}^3, \mathbb{R}^J)$ is defined as:

$$(O[\psi])_m(x) = (\psi * \kappa)_m(x) = \int_{\mathbb{R}^3} \sum_j \psi_j(y)\kappa_{jm}(x-y)dy \tag{3}$$

The kernel can be represented in an RBF basis:

$$\kappa_{jm}(z) = \sum_l k_{jml}\Phi(|z - v_l|), \tag{4}$$

where k_{jml} are learnable parameters of the network, Φ is the Gaussian kernel and $v_{l=1}^L$ represent translation vectors in $\mathbb{R}^3$. For example they can be chosen to cover a standard $3 \times 3 \times 3$ grid.

Restriction operator. The restriction operator $R_X : C(\mathbb{R}^3, \mathbb{R}^J) \to \mathbb{R}^{I\times J}$ is defined as simply as the restriction of the volumetric signal to the point cloud:

$$(R_X[\psi])_{i,j} = \psi_j(x_i) \tag{5}$$

Architecture With these definitions in hand, the authors of [2] propose to stack a series of convolutions with non-linear activation and pooling steps into a robust and flexible deep neural network architecture showing high accuracy on a range of tasks.

4. Our approach

4.1. Overview

Our main goal is to extend the PCNN approach to develop a rotation invariant convolutional neural network on point clouds. We call our network SPHNet due to the key role that the spherical harmonics kernels play in it. Figure 1 gives an overview. Following PCNN we first extend a function on point clouds to a volumetric function by the operator $\mathcal{E}_X$. Secondly, we apply the convolution operator SPHConv to the volumetric signal. Finally, the signal is restricted by R_X to the original point cloud.

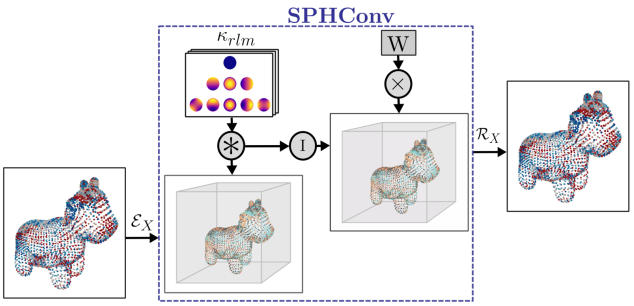

Figure 1. SPHNet framework. A signal on point cloud is first extended to $\mathbb{R}^3$ (left). Our convolution operator is applied to the extended function (center). The signal is restrained to the original point cloud (right).

Section 4.2 describes the spherical harmonics kernels and we describe our rotation invariant framework in Section 4.3.

4.2. Spherical harmonics kernel

In this work, we propose to use spherical harmonics-based kernels to design a point-based rotation-invariant network. In [24], the authors define spherical harmonics kernels with emphasis on rotation equivariance, applied to volumetric data. We adopt these constructions to our setting to define rotation-invariant convolutions.

Spherical harmonics is a family of real-valued functions on the unit sphere which can be defined, in particular, as the eigenfunctions of the spherical Laplacian. Namely, the ℓ^{th} spherical harmonic space has dimension $2\ell + 1$ and is spanned by spherical harmonics $(Y_{\ell,m})_{\ell>=0, m\in-\ell\dots\ell}$, where ℓ is the degree. Thus, each $Y_{\ell,m}$ is a real-valued function on the sphere $Y_{\ell,m} : \mathcal{S}_2 \to \mathbb{R}$.

Spherical harmonics are rotation equivariant. For any rotation $R \in \mathrm{SO}(3)$ we have:

$$Y_{\ell,m}(Rx) = \sum_{n=-\ell}^{\ell} D^\ell_{mn}(R)Y_{\ell,n}(x) \tag{6}$$

Where $D^\ell(R)$ is the so-called Wigner matrix of size $2\ell + 1 \times 2\ell + 1$. Importantly, Wigner matrices are *orthonormal* for all ℓ.

The spherical harmonic kernel basis introduced in [24] is defined as:

$$\kappa_{r\ell m}(x) = \exp\left(-\frac{\left|\|x\|_2 - \rho\frac{r}{n_R-1}\right|^2}{2\sigma^2}\right) Y_{\ell,m}\left(\frac{x}{\|x\|_2}\right), \tag{7}$$

where, ρ is a positive scale parameter, n_R is the number of radial samples and $\sigma = \frac{\rho}{n_R-1}$. Note that the kernel depends on a radial component, indexed by $r \in 0\dots n_R - 1$, and

defined by Gaussian shells of radius $r\frac{\rho}{n_R-1}$, and an angular component indexed by ℓ, m with $\ell \in 0...n_L-1$ and $m \in -\ell...\ell$, defined by values of the spherical harmonics.

This kernel inherits the behaviour of spherical harmonics under rotation, that is:

$$\kappa_{r\ell m}(Rx) = \sum_{n=-\ell}^{\ell} D^{\ell}_{mn}(R)\kappa_{r\ell n}(x) \tag{8}$$

Where $R \in SO(3)$.

4.3. Convolution layer

Below, we describe our SPHNet method. To define it we need to adapt the convolution and extension operators used in PCNN [2], while the restriction operators are kept exactly the same.

Extension. PCNN use Gaussian basis functions to extend functions to $\mathbb{R}^3$. However, convolution of the spherical harmonics kernel with Gaussians does not admit a closed-form expression. Therefore, we "extend" a signal f defined on a point cloud via a weighted combination of Dirac measures. Namely $\mathcal{E}_X : \mathbb{R}^{I\times J} \to C(\mathbb{R}^3, \mathbb{R}^J)$ is:

$$(\mathcal{E}_X[f])_j = \sum_i f_{ij}\omega_i\delta_{x_i}, \tag{9}$$

where we use the weights: $\omega_i = 1/(\sum_j \exp(-\frac{||x_j-x_i||^2}{2\sigma^2}))$. The Dirac measure has the following property:

$$\int_X f(y)\delta_x(y)\,\mathrm{d}y = f(x) \tag{10}$$

Convolution. We first introduce a non-linear rotation-invariant convolution operator: SPHConv. Using Eq. (10), the convolution between an extended signal and the spherical harmonic kernel $S[f] : C(\mathbb{R}^3, \mathbb{R}^J) \to C(\mathbb{R}^3, \mathbb{R}^J)$ is given by:

$$(S[f])_j(x) = (\mathcal{E}_X[f] * \kappa_{r\ell m})_j(x) = \sum_i f_{ij}\omega_i\kappa_{r\ell m}(x_i - x) \tag{11}$$

Using Eq. (8), we can express the convolution operator when a rotation R is applied to the point cloud X as a function of the kernel functions:

$$\begin{aligned}(R(\mathcal{E}_X[f] * \kappa_{r\ell m}))_j(x) &= \sum_i f_{ij}\omega_i\kappa_{r\ell m}(R(x_i - x)) \\ &= \sum_{n=-\ell}^{\ell} D^{\ell}_{mn}(R) \sum_i f_{ij}\omega_i\kappa_{r\ell n}(x_i - x) \\ &= \sum_{n=-\ell}^{\ell} D^{\ell}_{mn}(R)(\mathcal{E}_X[f] * \kappa_{r\ell n})_j(x)\end{aligned} \tag{12}$$

We observe that a rotation of the point cloud induces a rotation in feature space. In order to ensure rotation invariance we recall that the Wigner matrices D^l are all orthonormal. Thus, by taking the the norm of the convolution with respect to the degree of the spherical harmonic kernels, we can gain independence from R. To see this note that thanks to Eq. (12) only the m-indexed dimension of $(\mathcal{E}_X[f] * \kappa_{r\ell m})_j(x)$ is affected by the rotation. Therefore, we define our rotation-invariant convolution operator $I_{rl}[X, f] : C(\mathbb{R}^3, \mathbb{R}^J) \to C(\mathbb{R}^3, \mathbb{R}^J)$ as:

$$(I_{rl}[X, f])_j(x) = \|(\mathcal{E}_X[f] * \kappa_{r\ell m})_j(x)\|_2^m, \tag{13}$$

where for a tensor T_{rlmj} we use the notation $\|T\|_2^m$ to denote a tensor T^* obtained by taking the L_2 norm along the m dimension: $T^*_{rlj} = \sqrt{\sum_m T^2_{rlmj}}$.

Importantly, unlike the original PCNN approach [2], we cannot apply learnable weights directly on $(\mathcal{E}_X[f] * \kappa_{r\ell m})_j(x)$ as the result is not rotation-invariant. Instead, we take a linear combination of $(I_{rl}[X, f])$, obtained after taking the reduction along the m dimension above, using learnable weights $W \in \mathbb{R}^{G\times I\times n_R\times r_L}$, where G is the number of output channels. This leads to:

$$(O[f])_g(x) = \xi\left(\sum_{jr\ell} W_{gjr\ell}(I_{rl}[X, f])_j(x) + b_g\right) \tag{14}$$

Finally, we define the convolution operator $O_X : C(\mathbb{R}^3, \mathbb{R}^J) \to C(\mathbb{R}^3, \mathbb{R}^G)$ by restricting to the point cloud as in Eq. (5):

$$((O_X)_g[f])_i = \xi\left(\sum_{jr\ell} W_{gjr\ell}(I_{rl}[X, f](x_i) + b_g\right) \tag{15}$$

4.4. Pooling and upsampling

In addition to introducing a rotation-invariant convolution into the PCNN framework, we also propose several improvements aimed primarily to improve computational efficiency.

The original PCNN work [2] used Euclidean farthest point sampling and pooling in Voronoi cells. Both of these steps can be computationally inefficient, especially when handling large datasets and with data augmentation. Instead, we propose to use a space-partitioning data-structure for both steps using constructions similar to those in [12] . For this, we start by building a kd-tree for each point cloud, which we then use as follows.

Pooling. For a given point cloud P, our pooling layer of depth k reduces P from size 2^n to 2^{n-k} by applying max pooling to the features of each subtree. The coordinates of the points of the subtree are averaged. The resulting reduced

point cloud kd-tree structure and indexing can be immediately computed from the one computed for P. This gives us a family $T_{k\in 1\dots n}$ of kd-trees of varying depths.

Upsampling. The upsampling layer is computed simply by repeating the features of each point of a point cloud at layer k using the kd-tree of structure T_k. The upsampled point cloud follows the structure of T_{k+1}.

Comparison to PCNN. In PCNN pooling is performed through farthest point sampling. The maximum over the corresponding Voronoi cell is then assigned to each point of the sub-pointcloud. This method has a complexity of $O(|P|^2)$ while ours has a complexity of $O(|P|\log^2|P|)$, leading to noticeable improvement in practice.

We remark that kd-tree based pooling breaks full rotation invariance of our architecture, due to the construction of axis-aligned separating hyperplanes. However, as we show in the results below, this has a very mild effect in practice and our approach very similar performance regardless of the rotation of the data-set.

5. Architecture and implementation details

We adapted the classification and segmentation architectures from [2]. Using these as generic models we derive three general networks: our main rotation-invariant architecture **SPHnet**, which stands for Spherical Harmonic Net and uses the rotation invariant convolution we described. We also compare it to two baselines **SPHBase** which is identical to **SPHnet** except that we do not take the norm of each spherical harmonic component and apply the weights directly instead. We use this as the closest invariant baseline. We also compare to PCNN (mod), which is also non rotation-invariant and uses the Gaussian kernels from the original PCNN, but employs the same architecture and pooling as we use in **SPHnet** and **SPHBase**.

The original architectures in [2] consist of arrangements of convolution blocks, pooling and up-sampling layers that we replaced by our own. We kept the basic structure that we describe bellow. In all cases, we use construct the convolutional block using one convolutional layer followed by a batch normalization layer and a ReLU non linearity. Our convolution layer depends on the following parameters:

- Number of input and output channels
- Number n_L of spherical harmonics spaces in our kernel basis
- Number n_R of spherical shells in our kernel basis
- The kernel scale $\rho > 0$

For efficiency we also restrict the computation of convolution to a fixed number of points locally using k-nearest neighbours patches.

The number of input channels is deduced from the number of channels of the preceding layer. We used 64 points per patch in the classification case and 48 in the segmentation case. We fixed $n_L = 4$ and $n_R = 2$ throughout all of our experiments. The scale factor ρ can be defined only for the first layer and deduced for the other ones as will be explained bellow.

5.1. Classification

The classification architecture is made of 3 convolutional blocks with 64, 256, 1024 output channels respectively the first two are followed by a max pooling layer of ratio 4 and the last one if followed by a global max pooling layer. Finally we have a fully connected block over channels composed of two layers with 512 and 256 units each followed by a dropout layer of rate 0.5 and a final softmax layer for the classification as shown in Figure 2.

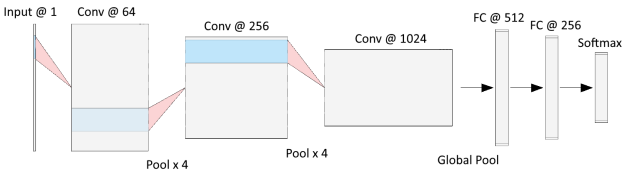

Figure 2. Classification architecture, Conv @ k indicates a conv layer with k output channels, Pool $\times k$ is a pooling of factor k and FC stands for fully connected layer

We chose $\rho = 0.1$ for the first layer and 2ρ, 4ρ for the second and third layers. The classification architecture we use expects a point cloud of 1024 points as input and uses it to compute the convolution layers according to our method. We use the constant constant function equal to 1 as the input feature to the first layer. Note that, since our goal is to ensure rotation invariance, we cannot use coordinate functions as input.

5.2. Segmentation

Our segmentation network takes as input a point cloud with 2048 points and, similarly to the classification case, we use the constant constant function equal to 1 as the input feature to the first layer of our segmentation architecture shown in Figure 3

The architecture is U-shaped consisting of an encoding and a decoding block each having 3 convolutional blocks. The encoding convolutional blocks have 64, 128, 256 output channels respectively. The first two are followed by max pooling layers of ratio 4 and the third by a pooling layer of ratio 8. These are followed by a decoding block where each conv block is preceded by an up-sampling layer to match the corresponding encoding block, which is then concatenated with it. The final conv-layer with softmax activation is then applied to predict pointwise labels. The two last

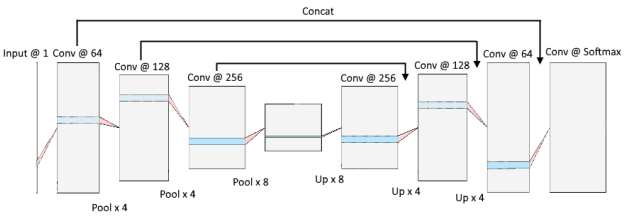

Figure 3. Segmentation architecture, Conv @ k indicates a conv layer with k output channels, Pool $\times k$ is a pooling of factor k and Up $\times k$ is an upsampling by a factor k

conv-blocks of the decode part are followed by dropout of rate 0.5. We chose a scale factor of $\rho = 0.08$ for the for the fist convolutional layer, the two next layers in the encoding block have respective scale factors 2ρ and 4ρ. The scale factors for the decode block are the same in reverse order. The scale factor for the final conv layer is ρ.

6. Results

6.1. Classification

We tested our method on the standard ModelNet40 benchmark [27]. This dataset consists of 9843 training and 2468 test shapes in 40 different classes, such as guitar, cone, laptop etc. We use the same version as in [2] with point clouds consisting of 2048 points centered and normalized so that the maximal distance of any point to the origin is 1. We randomly sub-sample the point clouds to 1024 points before sending them to the network. The dataset is aligned meaning that all point clouds are in canonical position.

We compare our approach **SPHNet** with different methods for learning on point clouds in Table 1. In addition to the original PCNN architecture and our modified versions of it, we compare to PointNet++ [20]. We also include the results on the recent rotation invariant framework PRIN [29].

We train the different models in two settings: we first train with the original (denoted by 'O') dataset and also with the dataset augmented by random rotations (denoted by 'A'). We then test with either the original testset or the test set augmented by rotations, again denoted by 'O' and 'A' respectively.

We observe that while other methods have a significant drop in accuracy when trained on the augmented training set, our method remains stable, and in particular outperforms all methods trained and tested with data augmentation (A/A) column. Implying that the rotation of the different shapes is not an important factor in the learning process. Moreover, our method achieves the best accuracy in this augmented training set setting. For PRIN evaluation, we used the architecture for classification described in [29] trained for 40 epochs as suggested in the paper. In all our experiments we train PRIN model with the default parameters except from the bandwidth which we set to 16 in order to fit within the 24GB of memory available in Titan RTX. As demonstrated in [29] this parameter choice produces slightly lower results but they are still comparable. In our experiments, we observed that PRIN method achieves poor performance when trained with rotation augmented data.

Method	O / O	A / O	O / A	A / A	time
PCNN	**92.3**	85.9	11.9	85.1	264 s
PointNet++	91.4	84.0	10.7	83.4	78 s
PCNN (mod)	91.1	83.4	9.4	84.5	22.6 s
SPHBaseNet	90.7	82.8	10.1	84.8	25.5 s
SPHNet (ours)	87.7	**87.1**	**86.6**	**87.6**	25.5 s
PRIN	71.5	0.78	43.1	0.78	811 s

Table 1. Classification accuracy on the modelnet40 dataset. A stands for data augmented by random rotations and O for original data. E.g., the model A/O was trained with augmented and tested on the original data. Timings per epoch are given when training on a NVIDIA RTX 2080 Ti card.

We also remark that even when training with data augmentation, our method converges significantly faster and to higher accuracies since it does not need to learn invariance to rotations. We show the evolution of the validation accuracy curves for different methods in Figure 4.

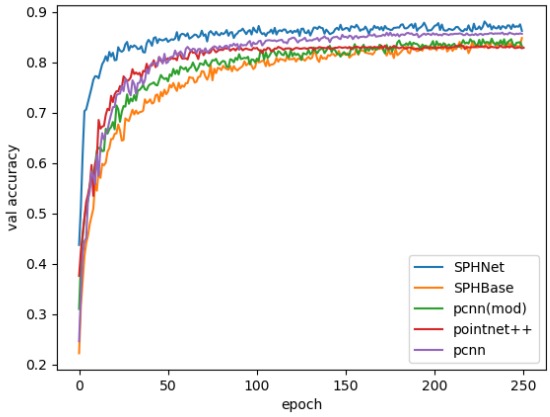

Figure 4. Validation accuracy for ModelNet40 classification with rotation augmentation at training.

6.2. Segmentation

We also applied our approach to tackle the challenging task of segmenting molecular surfaces into functionally-homologous regions within RNA molecules. We considered a family of 640 structures of 5s ribosomal RNAs (5s rRNAs), downloaded from the PDB database [3]. A surface, or molecular envelope, was computed for each RNA model

by sweeping the volume around the atomic positions with a small ball of fixed radius. This task was performed using the scripting interface of the Chimera structure-modelling environment [18]. Individual RNA sequences were then globally aligned, using an implementation of the Needleman-Wunsch algorithm [17], onto the RFAM [11] reference alignment RF00001 (5s rRNAs). Since columns in multiple sequence alignments are meant to capture functional homology, we treated each column index as a label, which we assigned to individual nucleotides, and their corresponding atoms, within each RNA. Labels were finally projected onto individual vertices of the surface by assigning to a vertex the label of its closest atom. This results each shape represented as a triangle mesh consisting of approximately 10k vertices, and a segmentation of each shape into approximately 120 regions, typically represented as connected components.

Due to the nature of this dataset the shapes are not pre-aligned and can have fairly significant geometric variability arising both from different conformations as well as from the molecule acquisition and reconstruction process (see Fig. 5 for an example).

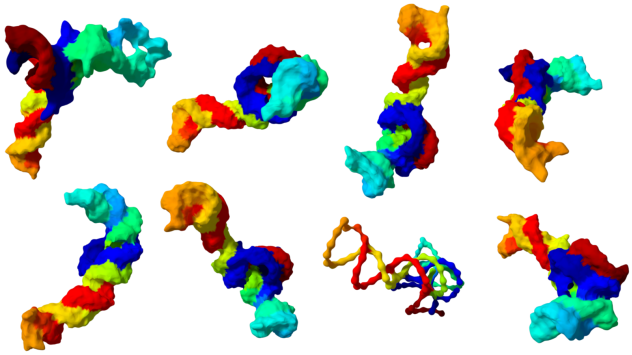

Figure 5. Labeled RNA molecules

Given a surface mesh, we first downsample it to 4096 points with farthest point sampling and then randomly sample to 2048 points before sending the data to the network.

We report the segmentation accuracy in Table 2. As in the classification case we observed that PRIN [29] degrades severely when augmenting the training set by random rotations. Overall, we observe that our method achieves the best accuracy in all settings. Furthermore, as we mentioned for the classification task, the accuracy of our method is stable to the different settings (with and without data augmentation) for this dataset.

We show a qualitative comparison to the PCNN method in Figure 6. We see that when trained on the original dataset and tested on an augmented dataset, we achieve significantly better performance than PCNN. This demonstrates that unlike PCNN, the performance of our method does not depend on the orientation of the shapes.

Method	O/O	A/O	O/A	A/A	time
PCNN	76.7	78.0	35.1	77.8	65 s
PointNet++	72.3	74.4	46.1	74.2	18 s
PCNN (mod)	74.2	74.3	30.9	73.7	9.7 s
SPHBaseNet	74.8	74.7	28.3	74.8	17 s
SPHNet (ours)	**80.8**	**80.1**	**79.5**	**80.4**	18 s
PRIN	66.9	6.84	53.7	6.57	10 s

Table 2. Segmentation accuracy on the RNA molecules dataset. Timings for one epoch are given when training on a NVIDIA RTX 2080 Ti card.

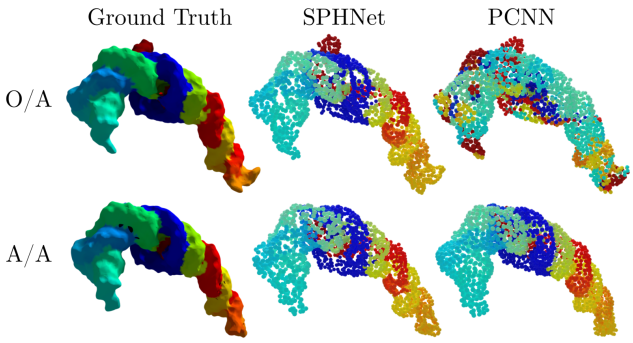

Figure 6. RNA segmentation

6.3. Matching

We also apply our method on the problem of finding correspondences between non-rigid shapes. This is an especially difficult problem since in addition to *global* rigid motion, the shapes can undergo non-rigid deformations, such as articulated motion of humans. Modeling such deformations through data augmentation would be challenging due to the very large number of degrees of freedom. On the other hand, our method, due to its full invariance *at every layer of the network* is especially amenable to this task, since it is not only globally but even locally invariant to rigid motions.

In this setting, we trained and tested different methods on point clouds sampled from the D-FAUST dataset [4]. This dataset contains scans of 10 different subjects completing various sequences of motions given as meshes with the same structure and indexing. We prepared a reduced dataset for our experiments, we selected 10 subject, motion sequences pairs for our test set and the complementary pairs defines our training set. Furthermore we sampled the motions sequences every 10 time-steps we selected 4068 shapes int total with a 3787/281 train/test split. We sampled 10k points uniformly on the first selected mesh and then subsampled 2048 points using farthest point sampling, we then transferred these points to all other meshes using their barycentric coordinates in the triangles of the first mesh to have a consistent indexation of points across all point-clouds. We produce labels by partitioning the first shape in

Method	O/O	A/O	O/A	A/A	disterr
PCNN	**99.6**	79.9	5.7	77.1	$7.7e{-3}$
PointNet++	97.1	85.0	10.4	84.5	$1.8e{-3}$
PCNN (mod)	60.4	55.2	2.5	54.7	0.01
SPHNet (ours)	98.0	**97.2**	**91.5**	**97.1**	$\mathbf{3.5e{-5}}$
PRIN	86.7	3.24	11.3	3.63	0.59

Table 3. Part label prediction accuracy on our D-FAUST dataset and average Euclidean distance error of the inferred correspondences between Voronoi cell centroids in the A/A case.

256 Voronoi cells associated to 256 farthest point samples. We then associate a label to each cell. We seek to predict this label and infer correspondences between the cells of two given shapes. This experiment is a particular instance of segmentation, we evaluate it with two metrics, first we measure the standard segmentation accuracy. In addition, to each cell a we can associate a cell $f(a)$ by taking the most represented cell among the predictions over a and measure the average Euclidean distance between the respective centroids of $f(a)$ and the ground truth image of a. Table 3 shows quantitative performance of different methods. Note that the performance of PointNet++ and PCNN decreases drastically when trained on the original dataset and tested on rotated (augmented) data sets. Our SPHNet performs well in all training and test settings. Moreover, SPHNet strongly outperforms existing methods with data augmentation applied both during training and testing, which more closely reflects a scenario of non pre-aligned training/test data.

In Figure 7, we show that the correspondences computed by SHPNet when trained on augmented data are highly accurate. For visual/qualitative evaluation we associate a color to each Voronoi cell using the x coordinate of its barycenter and transfer this color using the computed correspondence. Figure 8 shows qualitative comparison on DFAUST shape examples. We note that PCNN and PointNet++ correspondences present visually more artefacts including symmetry issues, while our SPHNet results in more smooth and accurate maps across all training and test settings.

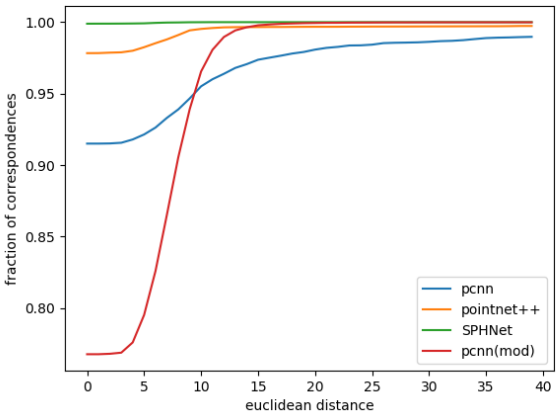

Figure 7. Fraction of correspondences on the D-FAUST dataset within a certain Euclidean error in the A/A case

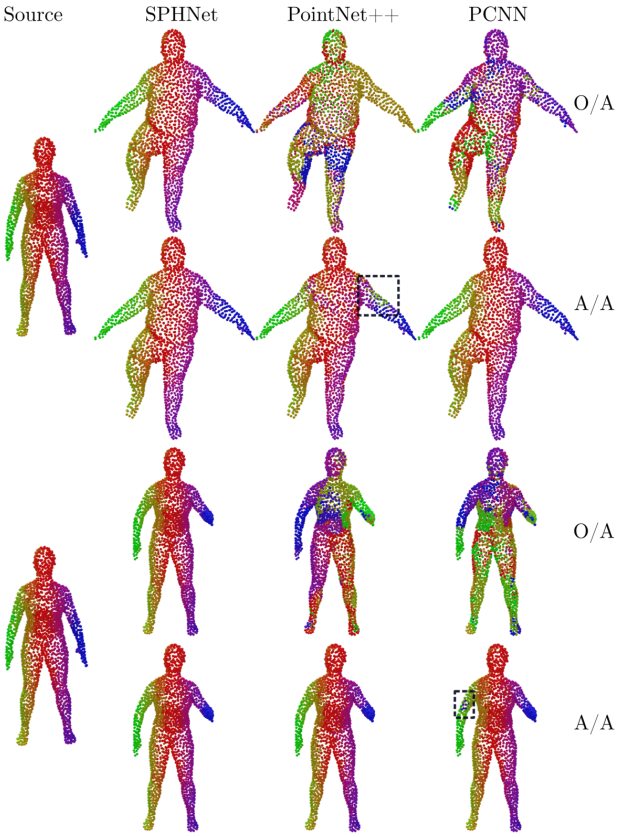

Figure 8. Qualitative comparison on D-FAUST dataset

7. Conclusion

We presented a novel approach for ensuring rotational invariance in a point-based deep neural network, based on the previously-proposed PCNN architecture. Key to our approach is the use of spherical harmonics kernels, which are both efficient, theoretically guaranteed to be rotationally invariant and can be applied at any layer of the network, providing flexibility between local and global invariance. Unlike previous methods in this domain, our resulting network outperforms existing approaches on un-aligned datasets even when data augmentation is applied. In the future, we plan to extend our method to a more general framework combining non-invariant, equivariant and fully invariant features at different levels of the network, and to devise ways for *automatically* deciding the optimal layers at which invariance must be ensured.

References

[1] V. Andrearczyk, J. Fageot, V. Oreiller, X. Montet, and A. Depeursinge. Exploring local rotation invariance in 3d cnns with steerable filters. 2018. https://openreview.net/forum?id=H1gXZLzxeE. 2

[2] M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018. 1, 2, 3, 4, 5, 6

[3] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000. 6

[4] F. Bogo, J. Romero, G. Pons-Moll, and M. J. Black. Dynamic FAUST: Registering human bodies in motion. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 7

[5] T. Cohen and M. Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999, 2016. 2

[6] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling. Spherical cnns. *arXiv preprint arXiv:1801.10130*, 2018. 2

[7] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. Pcpnet learning local shape properties from raw point clouds. In *Computer Graphics Forum*, volume 37, pages 75–85. Wiley Online Library, 2018. 2

[8] P. Hermosilla, T. Ritschel, P.-P. Vázquez, À. Vinacua, and T. Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. In *SIGGRAPH Asia 2018 Technical Papers*, page 235. ACM, 2018. 2

[9] B.-S. Hua, M.-K. Tran, and S.-K. Yeung. Pointwise convolutional neural networks. In *Proc. CVPR*, pages 984–993, 2018. 2

[10] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015. 1

[11] I. Kalvari, E. P. Nawrocki, J. Argasinska, N. Quinones-Olvera, R. D. Finn, A. Bateman, and A. I. Petrov. Non-coding RNA analysis using the RFAM database. *Current protocols in bioinformatics*, 62:e51, June 2018. 7

[12] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872, 2017. 4

[13] R. Kondor and S. Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. *arXiv preprint arXiv:1802.03690*, 2018. 2

[14] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018. 2

[15] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015. 1

[16] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017. 1, 2

[17] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48:443–453, Mar. 1970. 7

[18] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin. Ucsf chimera–a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25:1605–1612, Oct. 2004. 7

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 1, 2

[20] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 1, 2, 6

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

[22] N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018. 2

[23] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. 2

[24] M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems*, pages 10381–10392, 2018. 2, 3

[25] M. Weiler, F. A. Hamprecht, and M. Storath. Learning steerable filters for rotation equivariant cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2018. 2

[26] D. Worrall and G. Brostow. Cubenet: Equivariance to 3d rotation and translation. In *Proc. ECCV*, pages 567–584, 2018. 2

[27] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 1, 6

[28] K. Xu, V. G. Kim, Q. Huang, N. Mitra, and E. Kalogerakis. Data-driven shape analysis and processing. In *SIGGRAPH ASIA 2016 Courses*, page 4. ACM, 2016. 2

[29] Y. You, Y. Lou, Q. Liu, L. Ma, W. Wang, Y. Tai, and C. Lu. Prin: Pointwise rotation-invariant network. *arXiv preprint arXiv:1811.09361*, 2018. 2, 6, 7

[30] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Pu-net: Point cloud upsampling network. In *Proc. CVPR*, 2018. 2

[31] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. Pcn: Point completion network. In *2018 International Conference on 3D Vision (3DV)*, pages 728–737. IEEE, 2018. 2