DeltaConv: Anisotropic Operators for Geometric Deep Learning on Point Clouds

RUBEN WIERSMA, Delft University of Technology, The Netherlands
AHMAD NASIKUN, Delft University of Technology, The Netherlands and Universitas Gadjah Mada, Indonesia
ELMAR EISEMANN, Delft University of Technology, The Netherlands
KLAUS HILDEBRANDT, Delft University of Technology, The Netherlands

Learning from 3D point-cloud data has rapidly gained momentum, motivated by the success of deep learning on images and the increased availability of 3D data. In this paper, we aim to construct anisotropic convolution layers that work directly on the surface derived from a point cloud. This is challenging because of the lack of a global coordinate system for tangential directions on surfaces. We introduce DeltaConv, a convolution layer that combines geometric operators from vector calculus to enable the construction of anisotropic filters on point clouds. Because these operators are defined on scalar- and vector-fields, we separate the network into a scalar- and a vector-stream, which are connected by the operators. The vector stream enables the network to explicitly represent, evaluate, and process directional information. Our convolutions are robust and simple to implement and match or improve on state-of-the-art approaches on several benchmarks, while also speeding up training and inference.

CCS Concepts: • **Computing methodologies → Neural networks**; **Shape analysis**.

Additional Key Words and Phrases: Point Clouds, Point Cloud Learning, Point Cloud Processing, Geometric Deep Learning, Graph CNN

ACM Reference Format:
Ruben Wiersma, Ahmad Nasikun, Elmar Eisemann, and Klaus Hildebrandt. 2022. DeltaConv: Anisotropic Operators for Geometric Deep Learning on Point Clouds. *ACM Trans. Graph.* 41, 4, Article 105 (July 2022), 10 pages. https://doi.org/10.1145/3528223.3530166

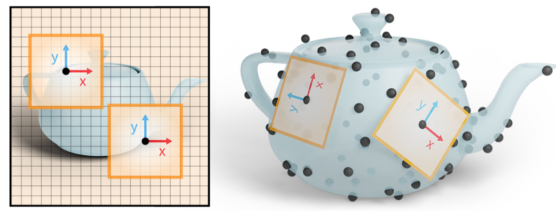

Fig. 1. Images have a global coordinate system (left). Point clouds do not (right), complicating the design of anisotropic convolutions.

1 INTRODUCTION

The success of convolutional neural networks (CNNs) on images and the increasing availability of point-cloud data motivate generalizing CNNs from images to 3D point clouds [Bronstein et al. 2017; Guo et al. 2020; Liu et al. 2019d]. One way to achieve this is to design convolutions that operate directly on the surface. Such *intrinsic* convolutions reduce the kernel space to tangent spaces, which are two-dimensional on surfaces. Compared to extrinsic convolutions, intrinsic convolutions can be more efficient and the search space for kernels is reduced, they naturally ignore empty space, and they are robust to rigid- and non-rigid deformations [Boscaini et al. 2016]. Examples of intrinsic convolutions on point clouds are GCN [Kipf and Welling 2017], PointNet++ [Qi et al. 2017b], EdgeConv [Wang et al. 2019], and DiffusionNet [Sharp et al. 2021].

Our focus is on constructing intrinsic convolutions which are anisotropic or direction-dependent. This is difficult because of the fundamental challenge that non-linear manifolds lack a global coordinate system. As an illustration of the problem, consider a CNN on images (Figure 1, left). Because an image has a globally consistent up-direction, the network can build anisotropic filters that activate the same way across the image. For example, one filter can test for vertical edges and the other for horizontal edges. No matter where the edges are in the image, the filter response is consistent. In subsequent layers, the output of these filters can be combined, e.g., to find a corner. Because we do not have a global coordinate system on surfaces (Figure 1, right), one cannot build and use anisotropic filters in the same way as on images. This limits current intrinsic convolutions on point clouds. For example, GCNs filters are isotropic. PointNet++ uses maximum aggregation and adds relative point positions, but still applies the same weight matrix to each neighboring point.

We introduce a new way to construct anisotropic convolution layers for geometric CNNs. Our convolutions are described in terms of geometric operators instead of kernels. The operator-based perspective is familiar from GCN, which uses the Laplacian on graphs. While the Laplacian is a natural fit for intrinsic learning on surfaces, it is isotropic. A classical way of creating anisotropic operators is to write the Laplacian as the divergence of the gradient and apply a linear or non-linear operation on the intermediate vector field [Weickert 1998]. We build on this idea by constructing learnable anisotropic operators from elemental geometric operators: the gradient, co-gradient, divergence, curl, Laplacian, and Hodge-Laplacian. These operators are defined on spaces of scalar fields and tangential vector fields. Hence, our networks are split into two

Authors' addresses: Ruben Wiersma, r.t.wiersma@tudelft.nl, Delft University of Technology, The Netherlands; Ahmad Nasikun, a.nasikun@tudelft.nl, Delft University of Technology, The Netherlands, and Universitas Gadjah Mada, Indonesia; Elmar Eisemann, e.eisemann@tudelft.nl, Delft University of Technology, The Netherlands; Klaus Hildebrandt, k.a.hildebrandt@tudelft.nl, Delft University of Technology, The Netherlands.

0730-0301/2022/7-ART105
https://doi.org/10.1145/3528223.3530166

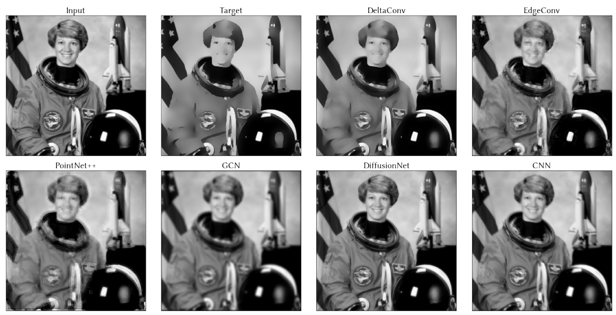

Fig. 2. A ResNet with varying convolutions is overfitted to a target image created with twenty anisotropic diffusion steps. DeltaConv can reproduce the filter well, where other convolutions struggle. (Courtesy NASA)

streams: one stream contains scalars and the other tangential vectors. The operators map along and between the two streams. The vector stream encodes feature activations and directions along the surface, allowing the network to test and relate directions in subsequent layers. Depending on the task, the network outputs scalars or vectors. A property of a network constructed from these operators is that it is coordinate-independent: though bases of the tangent spaces of a point cloud need to be chosen, the weights learned by the network will be the same no matter what bases are chosen. Hence, we can realize direction-dependent convolutions despite the lack of global coordinate systems on surfaces and without the need of specially constructed tangent space bases. We name our convolutions *DeltaConv*.

To get an idea of the benefits of DeltaConv, consider the anisotropic image filter proposed by Perona and Malik [1990]. The Perona–Malik filter integrates an anisotropic diffusion equation in which the anisotropic operator combines the gradient, a non-linearity, and the divergence. DeltaConv has access to the building blocks needed to construct such an anisotropic operator and to perform explicit integration steps of the diffusion equation. This is illustrated in Figure 2. We trained a simple ResNet [He et al. 2016] to match the result of twenty anisotropic diffusion steps on a sample image. While DeltaConv can reproduce the filter well, other intrinsic convolutions and regular image convolutions fail to capture the effect, producing overly smooth signals or artifacts instead. Additional benefits of our approach are the following: by maintaining a stream of vector features throughout the network, our convolutions can relate directional information between different points on the surface. Together with the increased expressiveness of convolutions due to anisotropy, this results in increased accuracy over isotropic convolutions, as well as state-of-the-art approaches, as we show in our experiments. Also, each operator is implemented as a sparse matrix and the combination of operators is computed per point, which is simple and efficient.

In our experiments, we demonstrate that a simple architecture with only a few DeltaConv blocks can match and, in some cases, outperform state-of-the-art results using more complex architectures. We achieve 93.8% accuracy on ModelNet40, 84.7% on the most difficult variant of ScanObjectNN, 86.9 mIoU on ShapeNet, and 99.6% on SHREC11, a dataset of non-rigidly deformed shapes. Our ablation studies show that adding the vector stream can decrease the error by up to 25% (from 90.4% to 92.8%) on ModelNet40 and up to 21% for ShapeNet (from 81.1 to 85.1 mIoU), while the use of per-point directional features speeds up inference by $1.5 - 2\times$ and the backward pass by $2.5 - 30\times$ compared to edge-based features.

Summarizing our main contributions:

- We introduce a new construction of convolution layers for geometric CNNs that supports the construction of anisotropic filters. This is achieved by letting networks learn convolutions as compositions and linear combinations of geometric differential operators and point-wise non-linearities. Moreover, the networks maintain a stream of vector features in addition to the usual stream of scalar features and use the operators to communicate in and between the streams.
- We propose a network architecture that realizes our approach and adapt the differential operators to work effectively in our networks.
- We implement and evaluate the network for point clouds[1] and propose techniques to cope with undersampled regions, noise, and missing information prevalent in point cloud learning.

2 RELATED WORK

We focus our discussion of related work on the most relevant topics. Please refer to surveys on geometric deep learning [Bronstein et al. 2021, 2017] and point cloud learning [Guo et al. 2020; Liu et al. 2019d] for a more comprehensive overview of this expanding field.

Point cloud networks and anisotropy. A common approach for learning on point-cloud data is to learn features for each point using a multi-layer perceptron (MLP), followed by local or global aggregation. Many methods also learn features on local point pairs before maximum aggregation. Well-known examples are PointNet and its successor PointNet++ [Qi et al. 2017a,b]. Several follow-up works improve speed and accuracy, for example by adding more combinations of point-pair features [Le et al. 2020; Liu et al. 2020; Lu et al. 2021; Qiu et al. 2021a; Sun et al. 2019; Xu et al. 2021b; Yang et al. 2019; Zhao et al. 2019]. Some of these point-wise MLPs explicitly encode anisotropy by splitting up the MLP for each 3D axis [Lan et al. 2019; Liu et al. 2020]. Concepts from transformers [Vaswani et al. 2017] have also made their way to point clouds [Lin et al. 2020; Zhang et al. 2021; Zhao et al. 2021]. These networks use self-attention to compute aggregation weights for (neighboring) points. Spatial information is incorporated by adding relative positions in 3D. Attention-based aggregation could be used in our approach as a replacement of maximum aggregation. The distance between points could serve as an intrinsic spatial encoding.

Pseudo-grid convolutions are a more direct translation of image convolutions to point clouds. Many of these are defined in 3D and thus support anisotropy in 3D coordinates. Several works learn a continuous kernel and apply it to local point-cloud regions [Atzmon et al. 2018; Boulch 2020; Fey et al. 2018; Hermosilla et al. 2018; Liu et al. 2019a,b; Thomas et al. 2019; Wu et al. 2019; Xu et al. 2021a]. Others learn discrete kernels and map points in local regions to a discrete grid [Choy et al. 2019; Graham et al. 2018; Hua et al. 2018;

[1]The implementation is available at https://github.com/rubenwiersma/deltaconv.

Lei et al. 2019; Li et al. 2018]. We go into an orthogonal direction by building intrinsic convolutions, which operate in fewer dimensions and naturally generalize to (non-)rigidly deformed shapes.

Finally, graph-based approaches create a k-nearest neighbor- or radius-graph from the input set and apply graph convolutions [Chen et al. 2020; Dominguez et al. 2018; Feng et al. 2019; Liu et al. 2019c; Pan et al. 2018; Shen et al. 2018; Simonovsky and Komodakis 2017; Te et al. 2018; Wang et al. 2018, 2019; Zhang et al. 2019; Zhang and Rabbat 2018]. DGCNN [Wang et al. 2019] introduces the EdgeConv operator and a dynamic graph component, which reconnects the k-nearest neighbor graph inside the network. EdgeConv computes the maximum over feature differences, which allows the network to represent directions in its channels. Channel-wise directions *can* resemble spatial directions if spatial coordinates are provided as input, which is only the case in the first layer for DGCNN. In contrast, our convolutions support anisotropy directly in the operators.

Rotation-equivariant approaches. Architectures with two streams and vector-valued features are also used in rotation-equivariant approaches for point clouds and meshes. A group of works studies rotation-equivariance in 3D space, aiming to design networks invariant to rigid point-cloud transformations [Cohen et al. 2018; Esteves et al. 2017; Poulenard et al. 2019; Thomas et al. 2018]. This concept is also incorporated in the transformer setups [Fuchs et al. 2020]. Rotation-equivariant kernels typically output vector-valued features. Vector Neurons simplify their use by linearly combining 3D vectors, followed by a vector non-linearity [Deng et al. 2021]. Our use of vector MLPs is similar. Differences are that we use tangential vectors, rather than 3D vectors, and we derive these vectors inside the network using geometric operators.

An alternative approach is to build networks using intrinsic rotation-equivariant convolutions on meshes [Cohen et al. 2019; de Haan et al. 2021; Gerken et al. 2021; Poulenard and Ovsjanikov 2018; Weiler et al. 2021; Wiersma et al. 2020]. These networks use local parametrizations and apply rotation- or gauge-equivariant kernels in the parameter domain to achieve independence from the choice of bases in the tangent spaces. Our approach is an alternative to gauge-equivariant networks. The use of differential operators also makes our networks independent of the choice of local coordinate systems. A benefit of our approach is that local parametrizations are not needed. For example, gauge-equivariant approaches typically use the exponential map for local parametrization but neglect the angular distortion induced by the parametrization. To the best of our knowledge, we are the first to implement and evaluate an intrinsic two-stream architecture on point clouds.

Geometric operators. Multiple authors use geometric operators to construct convolutions. The graph-Laplacian is used in GCN [Kipf and Welling 2017]. Spectral networks for learning on graphs are based on the eigenpairs of the graph-Laplacian [Bruna et al. 2014]. Surface networks for triangle meshes [Kostrikov et al. 2018] interleave the Laplacian with the extrinsic Dirac operator [Liu et al. 2017]. Parametrized Differential Operators (PDOs) [Jiang et al. 2019] use the gradient and divergence operators to learn from spherical signals on unstructured grids. DiffGCN [Eliasof and Treister 2020] uses finite difference schemes of the gradient and divergence operators for the construction of graph networks. DiffusionNet [Sharp et al. 2021] learns diffusion using the Laplace–Beltrami operator and directional features from gradients. DeltaConv uses a larger set of operators, combining and concatenating operators from vector calculus. In addition, it allows the processing of directional information in the stream of vector-valued features. A related approach is HodgeNet [Smirnov and Solomon 2021], which learns to build operators using the structure of differential operators. Outside of deep learning, differential operators are widely applied for the analysis of 3D shapes [Crane et al. 2013a; de Goes et al. 2016].

3 METHOD

We construct anisotropic convolutions by learning combinations of geometric differential operators. Because these operators are defined on scalar- and vector fields, we split our network into scalar and vector features. In this section, we describe these two streams, the operators and how they are discretized, and how combinations of the operators are learned. Finally, we consider the properties that result from this construction.

Streams. Consider a point cloud $\mathbf{P} \in \mathbb{R}^{N\times 3}$ with N points arranged in an $N \times 3$ matrix. All points can be associated with C additional features, which are stored in a matrix $\mathbf{X} \in \mathbb{R}^{N\times C}$. Inside the network, we refer to the features in layer l at point i as $\mathbf{x}_i^{(l)} \in \mathbb{R}^{C_l}$. All of these features constitute the *scalar stream.*

The *vector stream* runs alongside the scalar stream. Each feature in the vector stream is a tangent vector, encoded by coefficients (α_i^u, α_i^v) with respect to a basis in the corresponding tangent plane. The basis can be any pair of orthonormal vectors that are orthogonal to the normal vector. The coefficients are interleaved for each point, forming the matrix of features $\mathbf{V}^{(l)} \in \mathbb{R}^{2N\times C_l}$. One channel in $\mathbf{V}^{(l)}$ is a column of coefficients: $[\alpha_1^u, \alpha_1^v, \dots, \alpha_i^u, \alpha_i^v, \dots, \alpha_N^u, \alpha_N^v]^\intercal$. The input for the vector stream is a vector field defined at each point. In our experiments, we use the gradients of the input to the scalar stream. We will refer to the continuous counterparts of $\mathbf{X}$ and $\mathbf{V}$ as X and V, respectively.

3.1 Scalar to scalar: maximum aggregation

A simplified version of point-based MLPs is applied inside the scalar stream, building on PointNet++ [Qi et al. 2017b] and EdgeConv [Wang et al. 2019]. We apply an MLP per point and then perform maximum aggregation over a k-nn neighborhood $\mathcal{N}(i)$. The features in the scalar stream are computed as

$$\mathbf{x}_i^{(l+1)} = h_{\Theta_0}(\mathbf{x}_i^{(l)}) + \max_{j\in\mathcal{N}(i)} h_{\Theta_1}(\mathbf{x}_j^{(l)}), \tag{1}$$

where h_{Θ_0} and h_{Θ_1} denote multi-layer perceptrons (MLPs), consisting of fully connected layers, batch normalization [Ioffe and Szegedy 2015], and non-linearities. If point positions are used as input, they are centralized before maximum aggregation: $\hat{\mathbf{p}}_j = \mathbf{p}_j - \mathbf{p}_i$.

The biggest difference with EdgeConv and PointNet++ is that we use only point-based features within the network instead of edge-based features. The matrix multiplication used inside the MLP is thus not applied to kN feature vectors, but N point-wise feature vectors. This has a significant impact on the run time of the forward and backward passes. Directional information is encoded in per-point vectors instead of edges.

3.2 Scalar to vector: Gradient and co-gradient

The gradient and co-gradient operators connect the scalar stream to the vector stream. The gradients of a function represent the largest rate of change and the directions of that change as a vector at each point. The co-gradients are 90-degree rotations of the gradients. Combined, the gradients and co-gradients span the tangent planes, allowing the network to scale, skew, and rotate the gradient vectors.

We construct a discrete gradient operator using a moving least-squares approach on neighborhoods with k neighbors [Nealen 2004]. This approach is used in modeling and processing for point clouds and solving differential equations on point clouds [Crane et al. 2013b; Liang and Zhao 2013]. The procedure and accompanying theory is outlined in the supplemental material. The gradient operator is represented as a sparse matrix $\mathbf{G} \in \mathbb{R}^{2N \times N}$. It takes N values representing features on the points and outputs $2N$ values representing the gradient expressed in coefficients of the tangent basis of each point. The matrix is highly sparse as it only contains $2k$ elements in each row. The co-gradient $\mathbf{JG}$ is a composition of the gradient with a block-diagonal sparse matrix $\mathbf{J} \in \mathbb{R}^{2N \times 2N}$, where each block in $\mathbf{J}$ is a 2×2 90-degree rotation matrix.

Point clouds typically contain undersampled regions and noise. This can be problematic for the moving least-squares procedure. Consider the example in Figure 3, a chair with thin legs. Only a few points lie along the line constituting the legs of the chair. Hence, the perpendicular direction to the line is undersampled, resulting in a volatile least-squares fit: a minor perturbation of one of the points can heavily influence the outcome (left, circled area). We add a regularization term scaled by λ to the least-squares fitting procedure, which seeks to mitigate this effect (right). This is a known technique referred to as ridge regression or Tikhonov regularization.

We also argue that the gradient operator should be normalized, motivated by how information is fused in the network. If $\mathbf{G}$ exhibits diverging or converging behavior, features resulting from $\mathbf{G}$ will also diverge or converge. This is undesirable when the gradient is applied multiple times in the network. Features arising from the gradient operation would then have a different order of magnitude which needs to be accounted for by the network weights. Therefore, we normalize $\mathbf{G}$ by the ℓ_∞-operator norm, which provides an upper bound on the scaling behavior of an operator

$$\hat{\mathbf{G}} = \mathbf{G}/|\mathbf{G}|_\infty, \quad \text{where } |\mathbf{G}|_\infty = \max_i \sum_j |\mathbf{G}_{ij}|. \tag{2}$$

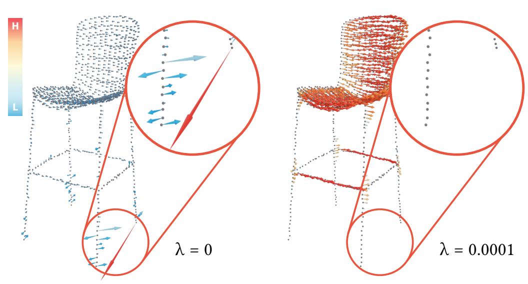

Fig. 3. Gradient of the x-coordinate on a chair without regularization (left) and with regularization (right).

3.3 Vector to scalar: Divergence, Curl, and Norm

The vector stream connects back to the scalar stream with divergence, curl, and norm. These operators are commonly used to analyze vector fields and indicate features such as sinks, sources, vortices, and the strength of the vector field. The network can use them as building blocks for anisotropic operators.

The discrete divergence is also constructed with a moving least-squares approach, which is described in the supplement. Divergence is represented as a sparse matrix $\mathbf{D} \in \mathbb{R}^{N \times 2N}$, with $2kN$ elements. Curl is derived as $-\mathbf{DJ}$.

3.4 Vector to vector: Hodge Laplacian

Vector features are diffused in the vector stream using a combination of the identity $\mathbf{I}$ and the Hodge Laplacian $\mathbf{\Delta}$ of V. Applying the Hodge Laplacian to a vector field V results in another vector field encoding the difference between the vector at each point and its neighbors. The Hodge Laplacian can be formulated as a combination of grad, div, curl and $\mathcal{J}$ [Brandt et al. 2017]

$$\mathbf{\Delta} = -(\text{grad div } + \mathcal{J} \text{ grad curl }). \tag{3}$$

In the discrete setting, we replace each operator with its discrete variant

$$\mathbf{L} = -(\mathbf{GD} - \mathbf{JGDJ}). \tag{4}$$

3.5 Why these operators?

The operators we use are related to each other in a fundamental way. They form a metric version of the *de Rham complex* of a surface [Wardetzky 2006]. The following diagram lays out the connections described in the previous sections, where each of the operators maps between functions (scalar fields) and vector fields.

$$X \underset{\text{div}}{\overset{\text{grad}}{\rightleftarrows}} V \underset{\text{co-grad}}{\overset{\text{curl}}{\rightleftarrows}} X \tag{5}$$

Note that the bottom row is a 90-degree rotated version of the top row. If we follow the diagram from left to right and apply grad and then curl to any function, the output will always be zero. The same holds for the path from right to left. The operators listed are first-order derivatives. Laplacians, which are second-order derivatives, can be formed by composing the first-order operators. For functions: to vector fields with grad and back again with div (Laplace-Beltrami). For vector fields: we go to scalars with div and curl and back again with grad and co-grad (Hodge-Laplacian). DeltaConv learns to combine these operators and supports anisotropy by adding non-linearities in-between.

3.6 DeltaConv: Learning Anisotropic Operators

Each of the operations either outputs scalar-valued or vector-valued features. We concatenate all the features belonging to each stream and then combine these features with parametrized functions

$$\begin{aligned} \mathbf{v}'_i &= \mathbf{h}_{\Theta_0}(\mathbf{v}_i,\ (\mathbf{GX})_i\,,(\mathbf{LV})_i), \\ \mathbf{x}'_i &= h_{\Theta_1}(\mathbf{x}_i, (\mathbf{DV}')_i\,, (-\mathbf{DJV}')_i\,, \|\mathbf{v}'_i\|) + \max_{j \in \mathcal{N}_i} h_{\Theta_2}(\mathbf{x}_j). \end{aligned} \tag{6}$$

We use the prime to indicate features in layer $l+1$. All other features are from layer l. h_{Θ_1} and h_{Θ_2} denote standard MLPs. $\mathbf{h}_{\Theta_0}$ denotes an MLP used for vectors. The vector MLPs scale and sum vectors,

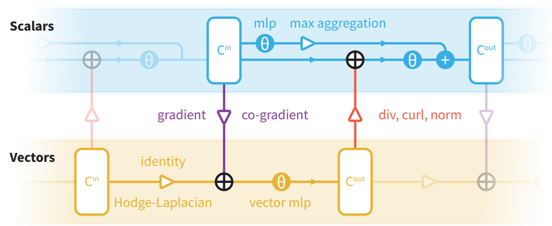

Fig. 4. Schematic of DeltaConv.

which means they do not work on individual vector coefficients and are coordinate-independent. Recall that $\mathbf{V} \in \mathbb{R}^{2N \times C^{(l)}}$ interleaves the vector coefficients for each point in the columns. One layer in the vector MLP is applied to $\mathbf{V}$ as follows

$$\mathbf{V}' = \sigma(\mathbf{VW}), \tag{7}$$

where $\mathbf{W} \in \mathbb{R}^{C^{(l)} \times C^{(l+1)}}$ is a weight matrix and σ is a non-linearity applied to vector norms. Matrix multiplication with $\mathbf{W}$ linearly combines the vector features but the individual coefficients of a vector are not mixed. Before the vector MLP is applied, we concatenate the 90-degree rotated vectors to the input features. This allows the MLP to also rotate vector features and enriches the set of operators. For example, the 90-degree rotated gradient is the co-gradient. The vector MLP can learn to combine information from local neighborhoods (through the gradient and Hodge–Laplacian), as well as information from different channels (through the identity). A schematic overview of Equation 6 can be found in Figure 4.

While Equation 6 formulates DeltaConv in terms of MLPs and feature concatenation, an alternative perspective is to consider the operations in Equation 6 as linearly combining the elementary operators and composing them with non-linearities in-between to form anisotropic geometric operators.

3.7 Properties of DeltaConv

The building blocks of DeltaConv, such as the gradient, divergence, curl, and the combination with non-linearities allow DeltaConv to build nonlinear anisotropic convolution filters. This is illustrated by the example of the Perona–Malik filter in Figure 2. The vector stream also allows DeltaConv to process vector features and their relative directions directly with the appropriate operators.

DeltaConv is formulated in terms of smooth differential operators and is not restricted to a specific surface representation. In this work, we implement DeltaConv for point clouds and images. However, the concepts generalize to other representations. For example, an implementation for meshes could be done using finite element discretizations [Brandt et al. 2017] or discrete exterior calculus [Crane et al. 2013a].

DeltaConv is coordinate-independent, meaning that the weights used in DeltaConv do not depend on the choices of tangent bases. For example, a forward pass on a shape with one choice of bases leads to the same output and weight updates when run with different bases. The coordinate-independence follows from the fact that all elementary operations in DeltaConv, such as applying geometric operators and vector MLPs, are coordinate-independent. It is known from differential geometry that one obtains the same results with geometric operators, no matter which basis is chosen [O'Neill 1983]. This property is preserved by the discretization of the operators and thus inherited by DeltaConv.

Finally, each of the building blocks of DeltaConv is isometry invariant. That means DeltaConv does not change if a shape is isometrically deformed. This property can be beneficial for tasks where shapes are rigidly or non-rigidly deformed. If the surface orientation is flipped, rotations in the tangent plane are flipped as well. DeltaConv is robust to this if only the gradient and divergence are used.

4 EXPERIMENTS

We validate our approach with comparisons to state-of-the-art approaches on classification and segmentation. In addition, we perform ablation studies to provide more insight into the effect of the vector stream on anisotropy, accuracy, and efficiency.

4.1 Implementation details

In our experiments we use network architectures based on DGCNN [Wang et al. 2019]. We replace each EdgeConv block with a DeltaConv block (Figure 4) and do not use the dynamic graph component. Thus, the networks operate at a single scale on local neighborhoods. Despite this simple architecture, DeltaConv achieves state-of-the-art results. To show what architectural optimizations mean for DeltaConv, we also test the U-ResNet architecture used in KPFCNN [Thomas et al. 2019] but with the convolution blocks in the encoder replaced by DeltaConv blocks. In the downsampling blocks used by these networks, we pool vector features by averaging them with parallel transport [Wiersma et al. 2020]. More details are provided in the supplemental material. Code is available at https://github.com/rubenwiersma/deltaconv.

Data transforms. A k-nn graph is computed for every shape. This graph is used for maximum aggregation in the scalar stream. It is reused to estimate normals when necessary and to construct the gradient. For each experiment, we use xyz-coordinates as input to the network and augment them with a random scale and translation, similar to previous works. Some datasets require specific augmentations, which are detailed in their respective sections.

Training. The parameters in the networks are optimized with stochastic gradient descent (SGD) with an initial learning rate of 0.1, momentum of 0.9 and weight decay of 0.0001. The learning rate is updated using a cosine annealing scheduler [Loshchilov and Hutter 2017], which decreases the learning rate to 0.001.

4.2 Classification

For classification, we study ModelNet40 [Wu et al. 2015], ScanObjectNN [Uy et al. 2019], and SHREC11 [Lian 2011]. With these experiments, we aim to demonstrate that our networks can achieve state-of-the-art performance on a wide range of challenges: point clouds sampled from CAD models, real-world scans, and non-rigid, deformable objects.

ModelNet40. The ModelNet40 dataset [Wu et al. 2015] consists of 12,311 CAD models from 40 categories. 9,843 models are used for

Table 1. Classification results on ModelNet40.

Method	Mean Class Accuracy	Overall Accuracy
PointNet++ [Qi et al. 2017b]	-	90.7
PointCNN [Li et al. 2018]	88.1	92.2
DGCNN [Wang et al. 2019]	90.2	92.9
KPConv deform [Thomas et al. 2019]	-	92.7
KPConv rigid [Thomas et al. 2019]	-	92.9
DensePoint [Liu et al. 2019a]	-	93.2
RS-CNN [Liu et al. 2019b]	-	93.6
GBNet [Qiu et al. 2021b]	91.0	**93.8**
PointTransformer [Zhao et al. 2021]	90.6	93.7
PAConv [Xu et al. 2021a]	-	93.6
Simpleview [Goyal et al. 2021]	-	93.6
Point Voxel Transformer [Zhang et al. 2021]	-	93.6
CurveNet [Xiang et al. 2021]	-	**93.8**
DeltaNet (ours)	**91.2**	**93.8**

training and 2,468 models for testing. Each point cloud consists of 1,024 points sampled from the surface using a uniform sampling of 8,192 points from mesh faces and subsequent furthest point sampling (FPS). We use 20 neighbors for maximum aggregation and to construct the gradient and divergence. Ground-truth normals are used to define tangent spaces for these operators and the regularizer is set to $\lambda = 0.01$. As input to the network, we use the xyz-coordinates. The classification architecture is optimized for 250 epochs. We do not use any voting procedure and list results without voting.

The results for this experiment can be found in Table 1. DeltaConv improves significantly on the most related maximum aggregation operators and is on par with or better than state-of-the-art approaches.

ScanObjectNN. ScanObjectNN [Uy et al. 2019] contains 2,902 unique object instances with 15 object categories sampled from SceneNN [Hua et al. 2016] and ScanNet [Dai et al. 2017]. The dataset is enriched to $\sim 15,000$ objects by preserving or removing background points and by perturbing bounding boxes. The variant without background points is tested without any perturbations (NO BG). The variant with background points is both tested without (BG) and with perturbations: Bounding boxes are translated (T), rotated (R), and scaled (S) before each shape is extracted. This means that some shapes are cut off, rotated, or scaled. T25 and T50 denote a translation by 25% and 50% of the bounding box size, respectively.

We use a modified version of the classification architecture with four convolution blocks with the following output dimensions: 64, 64, 64, 128. This setup matches the architecture used for DGCNN in [Uy et al. 2019]. Normals are estimated with 10 neighbors per point and the operators are constructed with 20 neighbors and $\lambda = 0.001$. As input, we provide the xyz-positions, augmented with a random rotation around the up-axis and a random scale $S \in \mathcal{U}(4/5, 5/4)$. The network is trained for 250 epochs.

Our results are compared to those reported by the authors of ScanObjectNN (row 1-8) [Uy et al. 2019] and other recent approaches in Table 2. We find that our approach outperforms all networks for every type of perturbation, including networks that explicitly account for background points.

Table 2. Classification results on ScanObjectNN.

Method	NO BG	BG	T25	T25R	T50R	T50RS
3DmFV [Ben-Shabat et al. 2018]	73.8	68.2	67.1	67.4	63.5	63.0
PointNet [Qi et al. 2017a]	79.2	73.3	73.5	72.7	68.2	68.2
SpiderCNN [Xu et al. 2018]	79.5	77.1	78.1	77.7	73.8	73.7
PointNet++ [Qi et al. 2017b]	84.3	82.3	82.7	81.4	79.1	77.9
DGCNN [Wang et al. 2019]	86.2	82.8	83.3	81.5	80.0	78.1
PointCNN [Li et al. 2018]	85.5	86.1	83.6	82.5	78.5	78.5
BGA-PN++ [Uy et al. 2019]	-	-	-	-	-	80.2
BGA-DGCNN [Uy et al. 2019]	-	-	-	-	-	79.9
GBNet [Qiu et al. 2021b]	-	-	-	-	-	80.5
GDANet [Xu et al. 2021b]	88.5	87.0	-	-	-	-
DRNet [Qiu et al. 2021a]	-	-	-	-	-	80.3
DeltaNet (ours)	**89.5**	**89.3**	**89.4**	**87.0**	**85.1**	**84.7**

SHREC11. The SHREC11 dataset [Lian 2011] consists of 900 non-rigidly deformed shapes, 30 each from 30 shape classes. This experiment aims to validate the claim that our approach is well suited for non-rigid deformations. Like previous works [Hanocka et al. 2019; Sharp et al. 2021; Wiersma et al. 2020], we train on 10 randomly selected shapes from each class and report the average over 10 runs. We sample 2048 points from the simplified meshes used in MeshCNNs experiments [Hanocka et al. 2019] and use 20 neighbors and mesh normals to construct the operators ($\lambda = 0.001$). As input, we provide xyz-coordinates, which are randomly rotated along each axis. We decrease the number of parameters in each convolution of the classification architecture to 32, since the dataset is much smaller than other datasets. The network is trained for 100 epochs. We find that our architecture is able to improve on state-of-the-art results (Table 3), validating the effectiveness of our intrinsic approach on deformable shapes.

4.3 Segmentation

For segmentation, we evaluate our architecture on ShapeNet (part segmentation) [Yi et al. 2016]. ShapeNet consists of 16,881 shapes from 16 categories. Each shape is annotated with up to six parts, totaling 50 parts. We use the point sampling of 2,048 points provided by the authors of PointNet [Qi et al. 2017a] and the train/validation/test split follows [Chang et al. 2015]. The operators are constructed with 30 neighbors and ground-truth normals to define tangent spaces

Table 3. Classification results on SHREC11.

Method	Accuracy
MeshCNN [Hanocka et al. 2019]	91.0
HSN [Wiersma et al. 2020]	96.1
MeshWalker [Lahav and Tal 2020]	97.1
PD-MeshNet [Milano et al. 2020]	99.1
HodgeNet [Smirnov and Solomon 2021]	94.7
FC [Mitchel et al. 2021]	99.2
DiffusionNet (xyz) [Sharp et al. 2021]	99.4
DiffusionNet (hks) [Sharp et al. 2021]	99.5
DeltaNet (ours)	**99.6**

Table 4. Part segmentation results on ShapeNet.

Method	Mean inst. mIoU
PointNet++ [Qi et al. 2017b]	85.1
PointCNN [Li et al. 2018]	86.1
DGCNN [Wang et al. 2019]	85.2
KPConv deform [Thomas et al. 2019]	86.4
KPConv rigid [Thomas et al. 2019]	86.2
GDANet [Xu et al. 2021b]	86.5
PointTransformer [Zhao et al. 2021]	86.6
PointVoxelTransformer [Zhang et al. 2021]	86.5
CurveNet [Xiang et al. 2021]	86.8
DeltaNet (ours)	86.6
Delta-U-ResNet (ours)	**86.9**

($\lambda = 0.001$). The xyz-coordinates are provided as input to the network, which is trained for 200 epochs. During testing, we evaluate each shape with ten random augmentations and aggregate the results with a voting procedure. Such a voting approach is used in the most recent works that we compare with.

The results are shown in Table 4, where our approach, especially the U-ResNet variant, improves upon the state-of-the-art approaches on the mean instance IoU metric and in many of the shape categories (full breakdown in the supplemental material). For each category, DeltaConv is either comparable to or better than other architectures and significantly better than the most related intrinsic approaches (PointNet++ and DGCNN). In Figure 5, we provide feature visualizations to give an idea of the features derived by the network.

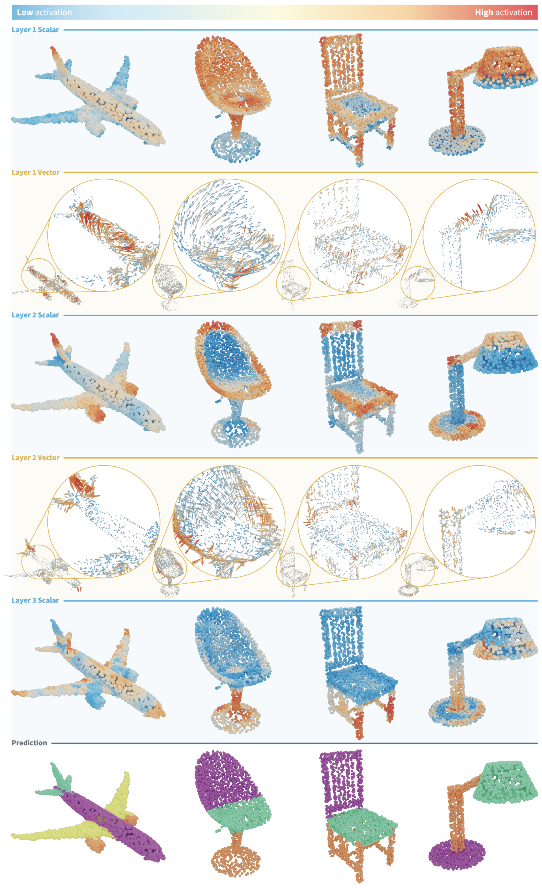

Fig. 5. For each layer of the network, we show how a single scalar- or vector-feature varies over shapes in ShapeNet. The last row shows the output of the network. The features tend to activate on similar regions.

4.4 Ablation Studies

We aim to validate the claim of anisotropy, isolate the effect of the vector stream, validate the choices to regularize and normalize the gradient and divergence operators, and investigate the impact of our approach on the timing and parameter counts of these networks.

Anisotropy. To validate that DeltaConv supports anisotropy, we train a network to mimic anisotropic diffusion [Perona and Malik 1990]. A ResNet [He et al. 2016] with 16 layers and 16 channels in the hidden layers is trained for 100 iterations with Adam [Kingma and Ba 2015] to match a target image generated with 20 anisotropic diffusion steps. In each diffusion step, the gradients are scaled with $\exp(-(|v|/0.05)^2)$. We vary the convolution blocks in the network with the ones from DiffusionNet [Sharp et al. 2021], EdgeConv [Wang et al. 2019], PointNet++ [Qi et al. 2017b], GCN [Kipf and Welling 2017], and regular image CNNs. For DiffusionNet, we set the diffusion time to a fixed value, as we are interested in the ability of the convolution to derive anisotropic filters through its gradient features. For all other convolutions, the neighborhoods are 3x3 pixel blocks. The results are shown in Figure 2 and in the supplement. DeltaConv achieves a good match. The other operators tend to blur the image or produce artifacts. For PointNet and EdgeConv, this is likely due to the variable nature and sharpness of the maximum aggregation. DiffusionNet lacks the divergence and curl operators and does not maintain a vector stream, which is necessary to analyze the relative directions of vector features in local neighborhoods.

Effectiveness of vector stream. To study the benefit of the vector stream and its effect on different types of intrinsic scalar convolutions, we set up three different scalar streams: (1) a Laplace–Beltrami operator, $\Delta = -\text{div}\,\text{grad}$, (2) GCN [Kipf and Welling 2017], and (3) maximum aggregation (Equation 1). We test three variants of each network: (1) only scalar stream, (2) scalar stream with the number of parameters adjusted to match a two-stream architecture, and (3) both the scalar and vector stream.

We test each configuration on ModelNet40 and ShapeNet. For both of these tasks, we use the DGCNN base architecture. The model for ShapeNet is trained for 50 epochs to save on training time and no voting is used, which results in slightly lower results than listed in Table 4. The results are listed in Table 5. We find that the vector stream improves the network for each scalar stream for both tasks, reducing the error between $19-25\%$ for classification and $3-21\%$ for segmentation. For maximum aggregation on ShapeNet, the improvements are lower, but still considerable, given the rate of progress on

Table 5. Ablations of DeltaConv on ShapeNet (Seg) and ModelNet40 (M40) with varying scalar streams.

Scalar Convolution	Vector Stream	Match # params	Seg mIoU	M40 mcA	M40 OA
Laplace–Beltrami	-	-	82.5	86.1	90.4
	-	✓	82.5	87.1	90.6
	✓	-	**84.9**	**89.4**	**92.2**
GCN	-	-	81.1	87.3	90.4
	-	✓	81.2	87.3	90.8
	✓	-	**85.1**	**90.6**	**92.8**
Max aggregation	-	-	85.7	89.2	92.2
	-	✓	85.7	89.5	92.6
	✓	-	**86.1**	**91.2**	**93.8**

this dataset over the last few years. Simply increasing the number of parameters in the scalar stream does not yield the same improvement as adding the vector stream, showing that the vector-valued features are of meaningful benefit. Maximum aggregation in the scalar stream yields the highest accuracy.

Timing and parameters. In our method section we argue that computing the gradient matrix is lightweight and that the simplified maximum aggregation operator is significantly faster than edge-based operators in PointNet++ and DGCNN. The main bottleneck in these convolutions is maximum aggregation over each edge. In this experiment, we demonstrate this by reporting the time it takes to train and test the classification network on one batch of 32 shapes with 1,024 points each. This includes all precomputation steps, such as computing the k-nearest neighbor graph ($\sim$ 15ms) and constructing the gradient and divergence operators ($\sim$ 30ms). The EdgeConv network is tested without a dynamic graph component, so that only the effect of precomputation and convolutions remains. All timings are obtained on the same machine with an NVIDIA RTX 2080Ti after a warm-up of 10 iterations. We implemented each method in PyTorch [Paszke et al. 2019] and PyTorch Geometric [Fey and Lenssen 2019]. The results are listed in Table 6. We find that our network only increases the number of parameters by 10%. Our network is significantly faster than the edge-based convolution: 1.5× faster in training and inference and 2.5× faster in the backward pass. DeltaConv with a Laplacian in the scalar stream is even faster: > 2× faster in training and inference and 30× faster in the backward pass.

Gradient regularization and normalization. In our method section, we argue that the least-squares fit for constructing the gradient and divergence should be regularized and the operators should be

Table 6. Timing and parameter counts for classification on ModelNet40. The timing for training and inference includes all necessary precomputations.

Convolution	Data Transform	Training	Backward	Inference	# Params
DeltaConv (Lapl.)	k-nn + ops	80ms	5ms	80ms	2,036,938
DeltaConv	k-nn + ops	130ms	60ms	125ms	2,037,962
EdgeConv	k-nn	196ms	147ms	186ms	1,801,610

Table 7. Classification accuracy on ModelNet40 with and without regularization and normalization.

λ	Normalization	Mean Class Accuracy	Overall Accuracy
10^{-32}	✓	85.2	90.3
10^{-2}	-	86.6	90.5
10^{-2}	✓	**89.4**	**92.2**

normalized. In this experiment, we intend to validate these choices. We train a model that is entirely based on our gradient operator, with a Laplace–Beltrami operator in the scalar stream. This means that every spatial operator in the network is influenced by regularization and scaling. The model is trained on the ModelNet40 for 50 epochs. The results are listed in Table 7. We notice a considerable difference between our approach with- and without regularization. There is a 2.8 percentage point decrease in mean class accuracy and 1.7 percentage point decrease in overall accuracy when the operator is not normalized.

5 CONCLUSION

In this work, we propose DeltaConv, a new convolutional layer for point cloud CNNs that is capable of extracting and processing directional features. DeltaConv separates features into a scalar- and vector stream and uses linear combinations and compositions of a selected set of geometric operators from vector calculus to map between and along the streams. This construction allows DeltaConv networks to learn anisotropic convolutions fitting to the data and task at hand. We demonstrate improved performance on a wide range of tasks, showing the potential of using DeltaConv in a learning setting on point clouds. We hope that this work will provide insight into the functionality and operation of neural networks for point clouds and spark more work that combines learning approaches with powerful tools from geometry processing.

Challenges and future work. We limit our study to analysis tasks. While we do not think it is impossible to adapt our operators for generative tasks, it is unclear if and when the operators should be recomputed when a surface is generated. Our work opens up interesting possibilities for future work. Besides exploring more applications of the vector stream, we want to test our approach on other surface discretizations and other manifolds (e.g., hyperbolic spaces and higher dimensional spaces) for which these operators are available, and also intend to study how other variants of the scalar stream impact the network.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback. This work has been partially supported by the NWO VIDI grant NextView, a gift from the TU Delft Universiteitsfonds, and a gift from Google Cloud. Ahmad Nasikun was supported through a doctoral scholarship from the Indonesia Endowment Fund for Education (LPDP).

REFERENCES

Matan Atzmon, Haggai Maron, Yaron Lipman, Atzmon Matan, Maron Haggai, and Lipman Yaron. 2018. Point convolutional neural networks by extension operators. *ACM Trans. Graph.* 37, 4 (2018), 71:1–71:12.

Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 2018. 3DmFV: Three-Dimensional Point Cloud Classification in Real-Time Using Convolutional Neural Networks. *IEEE Robotics and Automation Letters* 3 (2018), 3145–3152.

Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. 2016. Learning shape correspondence with anisotropic convolutional neural networks. *NeurIPS* (2016).

Alexandre Boulch. 2020. ConvPoint: Continuous convolutions for point cloud processing. *Comput. Graph. Forum* 88 (2020), 24–34.

Christopher Brandt, Leonardo Scandolo, Elmar Eisemann, and Klaus Hildebrandt. 2017. Spectral Processing of Tangential Vector Fields. *Computer Graphics Forum* 36, 6 (2017), 338–353.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. 2021. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478* (2021).

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Process. Mag.* 34 (2017), 18–42.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. *ICLR* (2014).

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. *ShapeNet: An Information-Rich 3D Model Repository.* Technical Report arXiv:1512.03012.

Jintai Chen, Biwen Lei, Qingyu Song, Haochao Ying, Danny Z Chen, and Jian Wu. 2020. A Hierarchical Graph Network for 3D Object Detection on Point Clouds. *CVPR* (2020).

Christopher Choy, JunYoung Gwak, and Silvio Savarese. 2019. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. *CVPR* (2019).

Taco Cohen, Mario Geiger, Jonas Koehler, and Max Welling. 2018. Spherical CNNs. *ICLR* (2018).

Taco Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. 2019. Gauge Equivariant Convolutional Networks and the Icosahedral CNN. *ICML* (2019).

Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schroder. 2013a. Digital Geometry Processing with Discrete Exterior Calculus. *SIGGRAPH Asia 2013 Courses* (2013), 7:1–7:126.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2013b. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph.* 32, 5 (2013), 152:1–152:11.

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. *CVPR* (2017).

Fernando de Goes, Mathieu Desbrun, and Yiying Tong. 2016. Vector Field Processing on Triangle Meshes. *SIGGRAPH Asia 2016 Courses* (2016), 27:1–27:49.

Pim de Haan, Maurice Weiler, Taco Cohen, and Max Welling. 2021. Gauge Equivariant Mesh CNNs: Anisotropic convolutions on geometric graphs. *ICLR* (2021).

C. Deng, O. Litany, Y. Duan, A. Poulenard, A. Tagliasacchi, and L. Guibas. 2021. Vector Neurons: A General Framework for SO(3)-Equivariant Networks. In *ICCV*.

Miguel Dominguez, Rohan Dhamdhere, Atir Petkar, Saloni Jain, Shagan Sah, and Raymond Ptucha. 2018. General-Purpose Deep Point Cloud Feature Extractor. *WACV* (2018).

Moshe Eliasof and Eran Treister. 2020. DiffGCN: Graph Convolutional Networks via Differential Operators and Algebraic Multigrid Pooling. *NeurIPS* (2020).

Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. 2017. Learning SO(3) Equivariant Representations with Spherical CNNs. *ECCV* (2017).

Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. 2019. Hypergraph neural networks. *AAAI* (2019).

Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. 2018. SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels. *CVPR* (2018).

Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. 2020. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. *NeurIPS* (2020).

Jan E Gerken, Jimmy Aronsson, Oscar Carlsson, Hampus Linander, Fredrik Ohlsson, Christoffer Petersson, and Daniel Persson. 2021. Geometric deep learning and equivariant neural networks. *arXiv preprint arXiv:2105.13926* (2021).

Ankit Goyal, Hei Law, Bowei Liu, Alejandro Newell, and Jia Deng. 2021. Revisiting Point Cloud Shape Classification with a Simple and Effective Baseline. *ICML* (2021).

Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 2018. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. *CVPR* (2018).

Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. 2020. Deep Learning for 3D Point Clouds: A Survey. *IEEE TPAMI* 43 (2020), 4338–4364.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: A Network with an Edge. *ACM Trans. Graph.* 38, 4 (2019), 90:1–90:12.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *CVPR* (2016).

Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. 2018. Monte Carlo Convolution for Learning on Non-Uniformly Sampled Point Clouds. *ACM Trans. Graph.* 37, 6 (2018), 235:1–235:12.

Binh-Son Hua, Quang-Hieu Pham, Duc Thanh Nguyen, Minh-Khoi Tran, Lap-Fai Yu, and Sai-Kit Yeung. 2016. SceneNN: A Scene Meshes Dataset with aNNotations. *3DV* (2016).

Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. 2018. Pointwise Convolutional Neural Networks. *CVPR* (2018).

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML* (2015).

Chiyu Max Jiang, Jingwei Huang, Karthik Kashinath, Prabhat, Philip Marcus, and Matthias Nießner. 2019. Spherical CNNs on Unstructured Grids. *ICLR* (2019).

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *ICLR* (2017).

Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. 2018. Surface Networks. *CVPR* (2018).

Alon Lahav and A. Tal. 2020. MeshWalker: deep mesh understanding by random walks. *ACM Trans. Graph.* 39, 6 (2020), 263:1–263:13.

Shiyi Lan, Ruichi Yu, Gang Yu, and L Davis. 2019. Modeling Local Geometric Structure of 3D Point Clouds Using Geo-CNN. *CVPR* (2019).

Eric-Tuan Le, Iasonas Kokkinos, and Niloy J Mitra. 2020. Going Deeper With Lean Point Networks. *CVPR* (2020).

Huan Lei, Naveed Akhtar, and Ajmal Mian. 2019. Octree guided CNN with Spherical Kernels for 3D Point Clouds. *CVPR* (2019).

Yangyan Li, Bu Rui, Mingchao Sun, Wei Wu, Xinhan Di, Baoquan Chen, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. 2018. PointCNN: Convolution on x-transformed points. *NeurIPS* (2018).

Zhouhui et al. Lian. 2011. SHREC '11 Track: Shape Retrieval on Non-rigid 3D Watertight Meshes. *Eurographics Workshop on 3D Object Retrieval* (01 2011), 79–88.

Jian Liang and Hongkai Zhao. 2013. Solving Partial Differential Equations on Point Clouds. *SIAM J. Sci. Comput.* 35 (2013), A1461–A1486.

Liqiang Lin, Pengdi Huang, Chi-Wing Fu, Kai Xu, Hao Zhang, and Hui Huang. 2020. One Point is All You Need: Directional Attention Point for Feature Learning. *arXiv preprint arXiv:2012.06257* (2020).

Hsueh-Ti Derek Liu, Alec Jacobson, and Keenan Crane. 2017. A Dirac Operator for Extrinsic Shape Analysis. *Comput. Graph. Forum* 36, 5 (2017), 139–149.

Jinxian Liu, Bingbing Ni, Caiyuan Li, Jiancheng Yang, and Qi Tian. 2019c. Dynamic Points Agglomeration for Hierarchical Point Sets Learning. *ICCV* (2019).

Weiping Liu, Jia Sun, Wanyi Li, Ting Hu, and Peng Wang. 2019d. Deep Learning on Point Clouds and Its Application: A Survey. *Sens* 19 (2019), 4188.

Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. 2019a. DensePoint: Learning densely contextual representation for efficient point cloud processing. *ICCV* (2019).

Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. 2019b. Relation-Shape Convolutional Neural Network for Point Cloud Analysis. *CVPR* (2019).

Ze Liu, Han Hu, Yue Cao, Zheng Zhang, and Xin Tong. 2020. A Closer Look at Local Aggregation Operators in Point Cloud Analysis. *ECCV* (2020).

Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. *ICLR* (2017).

Tao Lu, Limin Wang, and Gangshan Wu. 2021. CGA-Net: Category Guided Aggregation for Point Cloud Semantic Segmentation. *CVPR* (2021).

Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. 2020. Primal-Dual Mesh Convolutional Neural Networks. *NeurIPS* (2020).

Thomas W. Mitchel, Vladimir G. Kim, and Michael Kazhdan. 2021. Field Convolutions for Surface CNNs. *ICCV* (2021).

Andrew Nealen. 2004. An As-Short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. *URL: http://www.nealen.com/projects/mls/asapmls.pdf* 1 (2004).

B. O'Neill. 1983. *Semi-Riemannian Geometry With Applications to Relativity.* Elsevier Science.

Guanghua Pan, Jun Wang, Rendong Ying, and Peilin Liu. 2018. 3DTI-Net: Learn Inner Transform Invariant 3D Geometry Features using Dynamic GCN. *arXiv preprint arXiv:1812.06254* (2018).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS* (2019).

P Perona and J Malik. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE TPAMI* 12, 7 (1990), 629–639.

Adrien Poulenard and Maks Ovsjanikov. 2018. Multi-directional geodesic neural networks via equivariant convolution. *ACM Trans. Graph.* 37, 6 (2018), 236:1–236:14.

Adrien Poulenard, Marie-Julie Rakotosaona, Yann Ponty, and Maks Ovsjanikov. 2019. Effective Rotation-invariant Point CNN with Spherical Harmonics kernels. *3DV* (2019).

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017a. PointNet: Deep learning on point sets for 3D classification and segmentation. *CVPR* (2017).

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *NeurIPS* (2017).

Shi Qiu, Saeed Anwar, and Nick Barnes. 2021a. Dense-Resolution Network for Point Cloud Classification and Segmentation. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (2021).

Shi Qiu, Saeed Anwar, and Nick Barnes. 2021b. Geometric Back-projection Network for Point Cloud Classification. *arXiv preprint arXiv:1911.12885* (2021).

Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. 2021. DiffusionNet: Discretization Agnostic Learning on Surfaces. *ACM Trans. Graph.* 41, 3 (2021), 27:1–27:16.

Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. 2018. Mining point cloud local structures by kernel correlation and graph pooling. *CVPR* (2018).

Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. *CVPR* (2017).

Dmitriy Smirnov and Justin Solomon. 2021. HodgeNet: Learning Spectral Geometry on Triangle Meshes. *ACM Trans. Graph.* 40, 4 (2021), 166:1–166:11.

Xiao Sun, Zhouhui Lian, and Jianguo Xiao. 2019. SRINet: Learning Strictly Rotation-Invariant Representations for Point Cloud Classification and Segmentation. *ACM International Conference on Multimedia* (2019), 980–988.

Gusi Te, Wei Hu, Amin Zheng, and Zongming Guo. 2018. RGCNN: Regularized graph CNN for point cloud segmentation. *ACM International Conference on Multimedia* (2018), 746–754.

Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. 2019. KPConv: Flexible and Deformable Convolution for Point Clouds. *ICCV* (2019).

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. 2018. Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds. *arXiv preprint arXiv:1802.08219* (2018).

Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. 2019. Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data. *ICCV* (2019).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. *NeurIPS* (2017).

Chu Wang, Babak Samari, and Kaleem Siddiqi. 2018. Local spectral graph convolution for point set feature learning. *ECCV* (2018).

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graph.* 38, 5 (2019), 146:1–146:12.

Max Wardetzky. 2006. *Discrete Differential Operators on Polyhedral Surfaces - Convergence and Approximation.* Ph.D. Dissertation. Freie Universiät Berlin.

Joachim Weickert. 1998. *Anisotropic diffusion in image processing*. Vol. 1.

Maurice Weiler, Patrick Forré, Erik Verlinde, and Max Welling. 2021. Coordinate Independent Convolutional Networks – Isometry and Gauge Equivariant Convolutions on Riemannian Manifolds. *arXiv preprint arXiv:2106.06020* (2021).

Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. 2020. CNNs on surfaces using rotation-equivariant features. *ACM Trans. Graph.* 4 (2020), 92:1–92:12.

Wenxuan Wu, Zhongang Qi, and Li Fuxin. 2019. PointConv: Deep Convolutional Networks on 3D Point Clouds. *CVPR* (2019).

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets: A deep representation for volumetric shapes. *CVPR* (2015).

Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. 2021. Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis. *ICCV* (2021).

Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. 2021a. PAConv: Position Adaptive Convolution With Dynamic Kernel Assembling on Point Clouds. *CVPR* (2021).

Mutian Xu, Junhao Zhang, Zhipeng Zhou, Mingye Xu, Xiaojuan Qi, and Yu Qiao. 2021b. Learning Geometry-Disentangled Representation for Complementary Understanding of 3D Object Point Cloud. *AAAI* (2021).

Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. 2018. SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. *ECCV* (2018).

Jiancheng Yang, Qiang Zhang, B Ni, L Li, J Liu, Mengdie Zhou, and Q Tian. 2019. Modeling Point Clouds With Self-Attention and Gumbel Subset Sampling. *CVPR* (2019).

Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. 2016. A Scalable Active Framework for Region Annotation in 3D Shape Collections. *ACM Trans. Graph.* 35, 6 (2016), 210:1–210:12.

Cheng Zhang, Haocheng Wan, Shengqiang Liu, Xinyi Shen, and Zizhao Wu. 2021. PVT: Point-Voxel Transformer for 3D Deep Learning. *arXiv preprint arXiv:2108.06076* (2021).

Kuangen Zhang, Ming Hao, Jing Wang, Clarence W de Silva, and Chenglong Fu. 2019. Linked Dynamic Graph CNN: Learning on Point Cloud via Linking Hierarchical Features. *arXiv preprint arXiv:1904.10014* (2019).

Yingxue Zhang and Michael Rabbat. 2018. A Graph-CNN for 3D point cloud classification. *ICASSP* (2018), 6279–6283.

Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. 2019. PointWeb: Enhancing Local Neighborhood Features for Point Cloud Processing. *CVPR* (2019).

Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H.S. Torr, and Vladlen Koltun. 2021. Point Transformer. *ICCV* (2021).