Dynamic Graph CNN for Learning on Point Clouds

YUE WANG and YONGBIN SUN, Massachusetts Institute of Technology
ZIWEI LIU, UC Berkeley/ICSI
SANJAY E. SARMA, Massachusetts Institute of Technology
MICHAEL M. BRONSTEIN, Imperial College London/USI Lugano
JUSTIN M. SOLOMON, Massachusetts Institute of Technology

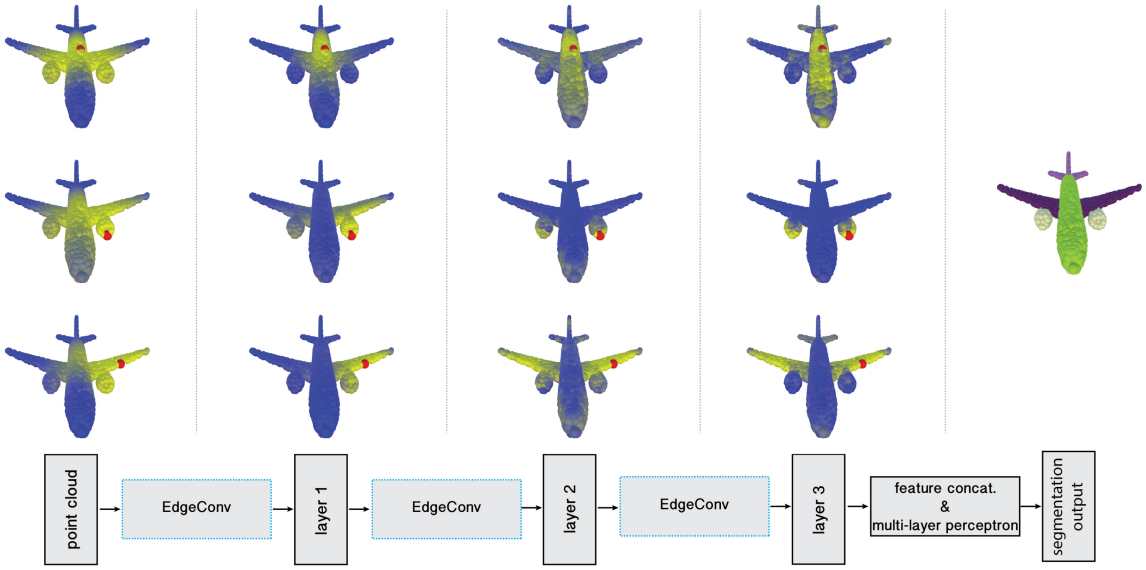


Fig. 1. Point cloud segmentation using the proposed neural network. Bottom: schematic neural network architecture. Top: Structure of the feature spaces produced at different layers of the network, visualized as the distance from the red point to all the rest of the points (shown left-to-right are the input and layers 1–3; rightmost figure shows the resulting segmentation). Observe how the feature space structure in deeper layers captures semantically similar structures such as wings, fuselage, or turbines, despite a large distance between them in the original input space.

Point clouds provide a flexible geometric representation suitable for countless applications in computer graphics; they also comprise the raw output of most 3D data acquisition devices. While hand-designed features on point clouds have long been proposed in graphics and vision, however, the recent overwhelming success of convolutional neural networks (CNNs) for image analysis suggests the value of adapting insight from CNN to the point cloud world. Point clouds inherently lack topological information, so designing a model to recover topology can enrich the representation power of point clouds. To this end, we propose a new neural network module dubbed *EdgeConv* suitable for CNN-based high-level tasks on point clouds, including classification and segmentation. EdgeConv acts on graphs dynamically computed in each layer of the network. It is differentiable and can be plugged into existing architectures. Compared to existing modules operating in extrinsic space or treating each point independently, EdgeConv has several appealing properties: It incorporates local neighborhood information; it can be stacked applied to learn global shape properties; and in multi-layer systems affinity in feature space captures semantic characteristics over potentially long distances in the original embedding. We show the performance of our model on standard benchmarks, including ModelNet40, ShapeNetPart, and S3DIS.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Point-based models**; *Shape analysis*;

Additional Key Words and Phrases: Point cloud, classification, segmentation

The authors acknowledge the generous support of Army Research Office Grant No. W911NF-12-R-0011, of Air Force Office of Scientific Research Award No. FA9550-19-1-0319, of National Science Foundation Grant No. IIS-1838071, of ERC Consolidator Grant No. 724228 (LEMAN), from an Amazon Research Award, from the MIT-IBM Watson AI Laboratory, from the Toyota-CSAIL Joint Research Center, from the Skoltech-MIT Next Generation Program, and from Google Faculty Research Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these organizations.
Authors' addresses: Y. Wang, Y. Sun, S. E. Sarma, and J. M. Solomon, Massachusetts Institute of Technology; emails: yuewang@csail.mit.edu, {yb_sun, sesarma, jsolomon}@mit.edu; Z. Liu, The Chinese University of Hong Kong; email: zwliu.hust@gmail.com; M. M. Bronstein, Imperial College London / USI Lugano; email: m.bronstein@imperial.ac.uk.

0730-0301/2019/10-ART146 $15.00
https://doi.org/10.1145/3326362

ACM Reference format:
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, Article 146 (October 2019), 12 pages.
https://doi.org/10.1145/3326362

1 INTRODUCTION

Point clouds, or scattered collections of points in 2D or 3D, are arguably the simplest shape representation; they also comprise the output of 3D sensing technology, including LiDAR scanners and stereo reconstruction. With the advent of fast 3D point cloud acquisition, recent pipelines for graphics and vision often process point clouds directly, bypassing expensive mesh reconstruction or denoising due to efficiency considerations or instability of these techniques in the presence of noise. A few of the many recent applications of point cloud processing and analysis include indoor navigation (Zhu et al. 2017), self-driving vehicles (Liang et al. 2018; Qi et al. 2017a; Wang et al. 2018b), robotics (Rusu et al. 2008b), and shape synthesis and modeling (Golovinskiy et al. 2009; Guerrero et al. 2018).

These modern applications demand *high-level* processing of point clouds. Rather than identifying salient geometric features like corners and edges, recent algorithms search for semantic cues and affordances. These features do not fit cleanly into the frameworks of computational or differential geometry and typically require learning-based approaches that derive relevant information through statistical analysis of labeled or unlabeled datasets.

In this article, we primarily consider point cloud classification and segmentation, two model tasks in point cloud processing. Traditional methods for solving these problems employ handcrafted features to capture geometric properties of point clouds (Lu et al. 2014; Rusu et al. 2009, 2008a). More recently, the success of deep neural networks for image processing has motivated a data-driven approach to learning features on point clouds. Deep point cloud processing and analysis methods are developing rapidly and outperform traditional approaches in various tasks (Chang et al. 2015).

Adaptation of deep learning to point cloud data, however, is far from straightforward. Most critically, standard deep neural network models require input data with regular structure, while point clouds are fundamentally irregular: Point positions are continuously distributed in the space, and any permutation of their ordering does not change the spatial distribution. One common approach to process point cloud data using deep learning models is to first convert raw point cloud data into a volumetric representation, namely a 3D grid (Maturana and Scherer 2015; Wu et al. 2015). This approach, however, usually introduces quantization artifacts and excessive memory usage, making it difficult to go to capture high-resolution or fine-grained features.

State-of-the-art deep neural networks are designed specifically to handle the irregularity of point clouds, directly manipulating raw point cloud data rather than passing to an intermediate regular representation. This approach was pioneered by *PointNet* (Qi et al. 2017b), which achieves permutation invariance of points by operating on each point independently and subsequently applying a symmetric function to accumulate features. Various extensions of PointNet consider neighborhoods of points rather than acting on each independently (Qi et al. 2017c; Shen et al. 2017); these allow the network to exploit local features, improving upon performance of the basic model. These techniques largely treat points independently at local scale to maintain permutation invariance. This independence, however, neglects the geometric relationships among points, presenting a fundamental limitation that cannot capture local features.

To address these drawbacks, we propose a novel simple operation, called EdgeConv, which captures local geometric structure while maintaining permutation invariance. Instead of generating point features directly from their embeddings, EdgeConv generates *edge features* that describe the relationships between a point and its neighbors. EdgeConv is designed to be invariant to the ordering of neighbors, and thus is permutation invariant. Because EdgeConv explicitly constructs a local graph and learns the embeddings for the edges, the model is capable of grouping points both in Euclidean space and in semantic space.

EdgeConv is easy to implement and integrate into existing deep learning models to improve their performance. In our experiments, we integrate EdgeConv into the basic version of *PointNet* without using any feature transformation. We show the resulting network achieves state-of-the-art performance on several datasets, most notably *ModelNet40* and *S3DIS* for classification and segmentation.

Key Contributions. We summarize the key contributions of our work as follows:

- We present a novel operation for learning from point clouds, EdgeConv, to better capture local geometric features of point clouds while still maintaining permutation invariance.
- We show the model can learn to semantically group points by dynamically updating a graph of relationships from layer to layer.
- We demonstrate that EdgeConv can be integrated into multiple existing pipelines for point cloud processing.
- We present extensive analysis and testing of EdgeConv and show that it achieves state-of-the-art performance on benchmark datasets.

2 RELATED WORK

Hand-Crafted Features. Various tasks in geometric data processing and analysis—including segmentation, classification, and matching—require some notion of local similarity between shapes. Traditionally, this similarity is established by constructing feature descriptors that capture local geometric structure. Countless papers in computer vision and graphics propose local feature descriptors for point clouds suitable for different problems and data structures. A comprehensive overview of hand-designed point features is out of the scope of this article, but we refer the reader to Biasotti et al. (2016), Guo et al. (2014), and Van Kaick et al. (2011) for discussion.

Broadly speaking, one can distinguish between *extrinsic* and *intrinsic* descriptors. Extrinsic descriptors usually are derived from the coordinates of the shape in 3D space and includes classical methods like shape context (Belongie et al. 2001), spin images (Johnson and Hebert 1999), integral features (Manay et al. 2006), distance-based descriptors (Ling and Jacobs 2007), point feature histograms (Rusu et al. 2009, 2008a), and normal histograms (Tombari et al. 2011), to name a few. Intrinsic descriptors treat

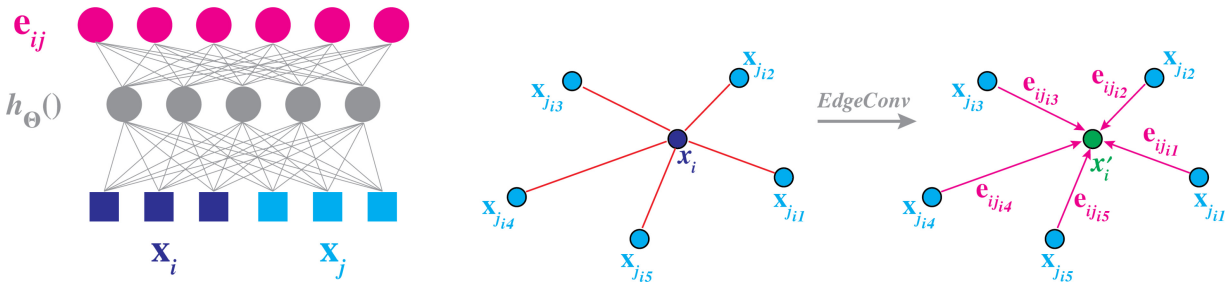


Fig. 2. Left: Computing an edge feature, $\mathbf{e}_{ij}$ (top), from a point pair, $\mathbf{x}_i$ and $\mathbf{x}_j$ (bottom). In this example, $h_\Theta()$ is instantiated using a fully connected layer, and the learnable parameters are its associated weights. Right: The EdgeConv operation. The output of EdgeConv is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

the 3D shape as a manifold whose metric structure is discretized as a mesh or graph; quantities expressed in terms of the metric are invariant to isometric deformation. Representatives of this class include spectral descriptors such as global point signatures (Rustamov 2007), the heat and wave kernel signatures (Aubry et al. 2011; Sun et al. 2009), and variants (Bronstein and Kokkinos 2010). Most recently, several approaches wrap machine learning schemes around standard descriptors (Guo et al. 2014; Shah et al. 2013).

Deep Learning on Geometry. Following the breakthrough results of convolutional neural networks (CNNs) in vision (Krizhevsky et al. 2012; LeCun et al. 1989), there has been strong interest to adapt such methods to geometric data. Unlike images, geometry usually does not have an underlying grid, requiring new building blocks replacing convolution and pooling or adaptation to a grid structure.

As a simple way to overcome this issue, view-based (Su et al. 2015; Wei et al. 2016) and volumetric representations (Klokov and Lempitsky 2017; Maturana and Scherer 2015; Tatarchenko et al. 2017; Wu et al. 2015)—or their combination (Qi et al. 2016)—"place" geometric data onto a grid. More recently, PointNet (Qi et al. 2017b, 2017c) exemplifies a broad class of deep learning architectures on non-Euclidean data (graphs and manifolds) termed *geometric deep learning* (Bronstein et al. 2017). These date back to early methods to construct neural networks on graphs (Scarselli et al. 2009), recently improved with gated recurrent units (Li et al. 2016) and neural message passing (Gilmer et al. 2017). Bruna et al. (2013) and Henaff et al. (2015) generalized convolution to graphs via the Laplacian eigenvectors (Shuman et al. 2013). Computational drawbacks of this foundational approach were alleviated in follow-up works using polynomial (Defferrard et al. 2016; Kipf and Welling 2017; Monti et al. 2017b, 2018), or rational (Levie et al. 2017) spectral filters that avoid Laplacian eigendecomposition and guarantee localization. An alternative definition of non-Euclidean convolution employs spatial rather than spectral filters. The *Geodesic CNN (GCNN)* is a deep CNN on meshes generalizing the notion of patches using local intrinsic parameterization (Masci et al. 2015). Its key advantage over spectral approaches is better generalization as well as a simple way of constructing directional filters. Follow-up work proposed different local charting techniques using anisotropic diffusion (Boscaini et al. 2016) or Gaussian mixture models (Monti et al. 2017a; Veličković et al. 2017). In Halimi et al. (2018) and Litany et al. (2017b), a differentiable functional map (Ovsjanikov et al. 2012) layer was incorporated into a geometric deep neural network, allowing to do intrinsic structured prediction of correspondence between nonrigid shapes.

The last class of geometric deep learning approaches attempts to pull back a convolution operation by embedding the shape into a domain with shift-invariant structure such as the sphere (Sinha et al. 2016), torus (Maron et al. 2017), plane (Ezuz et al. 2017), sparse network lattice (Su et al. 2018), or spline (Fey et al. 2018).

Finally, we should mention *geometric generative models*, which attempt to generalize models such as autoencoders, variational autoencoders (VAE) (Kingma and Welling 2013), and generative adversarial networks (GAN) (Goodfellow et al. 2014) to the non-Euclidean setting. One of the fundamental differences between these two settings is the lack of canonical order between the input and the output vertices, thus requiring an input-output correspondence problem to be solved. In 3D mesh generation, it is commonly assumed that the mesh is given and its vertices are canonically ordered; the generation problem thus amounts only to determining the embedding of the mesh vertices. Kostrikov et al. (2017) proposed SurfaceNets based on the extrinsic Dirac operator for this task. Litany et al. (2017a) introduced the intrinsic VAE for meshes and applied it to shape completion; a similar architecture was used by Ranjan et al. (2018) for 3D face synthesis. For point clouds, multiple generative architectures have been proposed (Fan et al. 2017; Li et al. 2018b; Yang et al. 2018).

3 OUR APPROACH

We propose an approach inspired by PointNet and convolution operations. Instead of working on individual points like PointNet, however, we exploit local geometric structures by constructing a local neighborhood graph and applying convolution-like operations on the edges connecting neighboring pairs of points, in the spirit of graph neural networks. We show in the following that such an operation, dubbed *edge convolution* (EdgeConv), has properties lying between translation-invariance and non-locality.

Unlike graph CNNs, our graph is not fixed but rather is dynamically updated after each layer of the network. That is, the set of k-nearest neighbors of a point changes from layer to layer of the network and is computed from the sequence of embeddings. Proximity in feature space differs from proximity in the input, leading to nonlocal diffusion of information throughout the point cloud. As a connection to existing work, Non-local Neural Networks (Wang et al. 2018a) explored similar ideas in the video recognition field, and follow-up work by Xie et al. (2018) proposed using

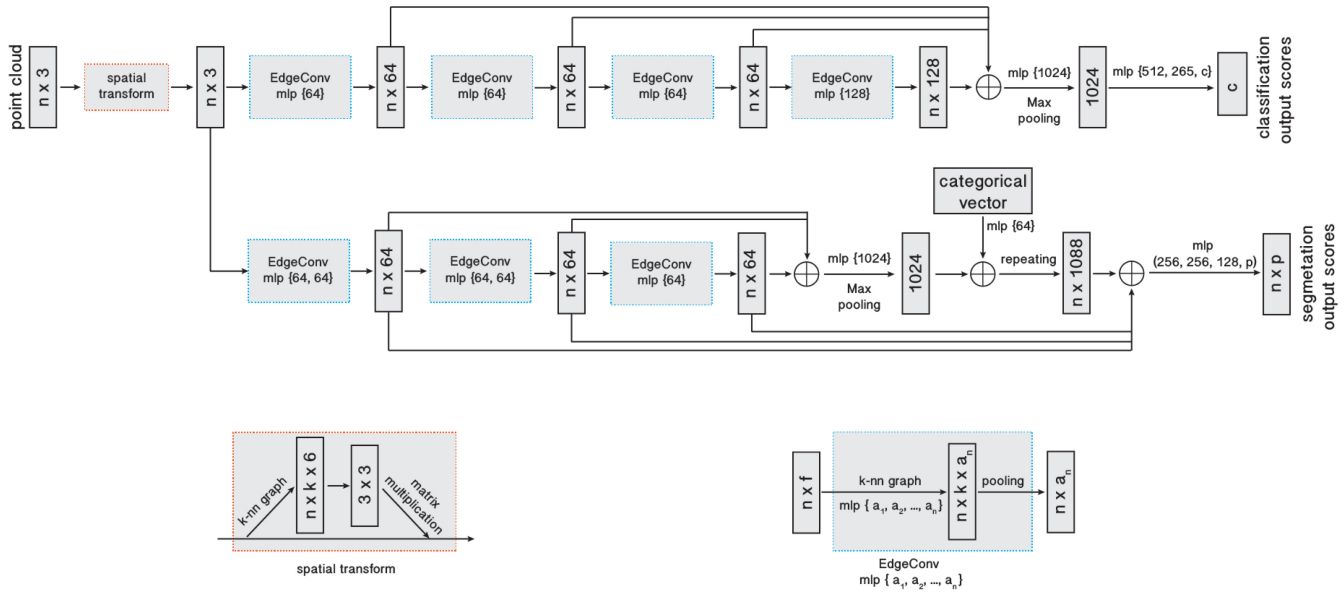


Fig. 3. Model architectures: The model architectures used for classification (top branch) and segmentation (bottom branch). The classification model takes as input n points, calculates an edge feature set of size k for each point at an EdgeConv layer, and aggregates features within each set to compute EdgeConv responses for corresponding points. The output features of the last EdgeConv layer are aggregated globally to form an 1D global descriptor, which is used to generate classification scores for c classes. The segmentation model extends the classification model by concatenating the 1D global descriptor and all the EdgeConv outputs (serving as local descriptors) for each point. It outputs per-point classification scores for p semantic labels. $\oplus$: concatenation. Point cloud transform block: The point cloud transform block is designed to align an input point set to a canonical space by applying an estimated 3×3 matrix. To estimate the 3×3 matrix, a tensor concatenating the coordinates of each point and the coordinate differences between its k neighboring points is used. EdgeConv block: The EdgeConv block takes as input a tensor of shape $n \times f$, computes edge features for each point by applying a multi-layer perceptron (mlp) with the number of layer neurons defined as $\{a_1, a_2, \ldots, a_n\}$, and generates a tensor of shape $n \times a_n$ after pooling among neighboring edge features.

non-local blocks to denoise feature maps to defend against adversarial attacks.

3.1 Edge Convolution

Consider an F-dimensional point cloud with n points, denoted by $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^F$. In the simplest setting of $F = 3$, each point contains 3D coordinates $\mathbf{x}_i = (x_i, y_i, z_i)$; it is also possible to include additional coordinates representing color, surface normal, and so on. In a deep neural network architecture, each subsequent layer operates on the output of the previous layer, so more generally the dimension F represents the feature dimensionality of a given layer.

We compute a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing local point cloud structure, where $\mathcal{V} = \{1, \ldots, n\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ are the *vertices* and *edges*, respectively. In the simplest case, we construct $\mathcal{G}$ as the k-nearest neighbor (k-NN) graph of $\mathbf{X}$ in $\mathbb{R}^F$. The graph includes self-loop, meaning each node also points to itself. We define *edge features* as $\boldsymbol{e}_{ij} = h_{\boldsymbol{\Theta}}(\mathbf{x}_i, \mathbf{x}_j)$, where $h_{\boldsymbol{\Theta}} : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^{F'}$ is a nonlinear function with a set of learnable parameters $\boldsymbol{\Theta}$.

Finally, we define the EdgeConv operation by applying a channel-wise symmetric aggregation operation $\square$ (e.g., $\sum$ or max) on the edge features associated with all the edges emanating from each vertex. The output of EdgeConv at the i-th vertex is thus given by

$$\mathbf{x}'_i = \underset{j:(i,j)\in\mathcal{E}}{\square} h_{\boldsymbol{\Theta}}(\mathbf{x}_i, \mathbf{x}_j). \tag{1}$$

Making analogy to convolution along images, we regard $\mathbf{x}_i$ as the central pixel and $\{\mathbf{x}_j : (i,j) \in \mathcal{E}\}$ as a patch around it (see Figure 2). Overall, given an F-dimensional point cloud with n points, EdgeConv produces an F'-dimensional point cloud with the same number of points.

Choice of h and $\square$. The choice of the edge function and the aggregation operation has a crucial influence on the properties of EdgeConv. For example, when $\mathbf{x}_1, \ldots, \mathbf{x}_n$ represent image pixels on a regular grid and the graph $\mathcal{G}$ has connectivity representing patches of fixed size around each pixel, the choice $\boldsymbol{\theta}_{\boldsymbol{m}} \cdot \mathbf{x}_j$ as the edge function and sum as the aggregation operation yields standard convolution:

$$x'_{im} = \sum_{j:(i,j)\in\mathcal{E}} \boldsymbol{\theta}_m \cdot \mathbf{x}_j. \tag{2}$$

Here, $\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M)$ encodes the weights of M different filters. Each $\boldsymbol{\theta}_m$ has the same dimensionality as $\mathbf{x}$, and $\cdot$ denotes the Euclidean inner product.

A second choice of h is

$$h_{\boldsymbol{\Theta}}(\mathbf{x}_i, \mathbf{x}_j) = h_{\boldsymbol{\Theta}}(\mathbf{x}_i), \tag{3}$$

encoding only global shape information oblivious of the local neighborhood structure. This type of operation is used in PointNet, which can thus be regarded as a special case of EdgeConv.

A third choice of h adopted by Atzmon et al. (2018) is

$$h_{\boldsymbol{\Theta}}(\mathbf{x}_i, \mathbf{x}_j) = h_{\boldsymbol{\Theta}}(\mathbf{x}_j) \tag{4}$$

Table 1. Comparison to Existing Methods

	Aggregation	Edge Function	Learnable parameters
PointNet (Qi et al. 2017b)	—	$h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) = h_{\Theta}(\mathbf{x}_i)$	Θ
PointNet++ (Qi et al. 2017c)	max	$h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) = h_{\Theta}(\mathbf{x}_j)$	Θ
MoNet (Monti et al. 2017a)	$\sum$	$h_{\boldsymbol{\theta}_m, \boldsymbol{w_n}}(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\theta}_m \cdot (\mathbf{x}_j \odot g_{\boldsymbol{w}_n}(u(\mathbf{x}_i, \mathbf{x}_j)))$	$\boldsymbol{w_n}, \boldsymbol{\theta}_m$
PCNN (Atzmon et al. 2018)	$\sum$	$h_{\boldsymbol{\theta}_m}(\mathbf{x}_i, \mathbf{x}_j) = (\boldsymbol{\theta}_m \cdot \mathbf{x}_j) g(u(\mathbf{x}_i, \mathbf{x}_j))$	$\boldsymbol{\theta_m}$

The per-point weight w_i in Atzmon et al. (2018) effectively is computed in the first layer and could be carried onward as an extra feature; we omit this for simplicity.

and

$$x'_{im} = \sum_{j \in \mathcal{V}} (h_{\theta(\mathbf{x}_j)}) g(u(\mathbf{x}_i, \mathbf{x}_j)), \tag{5}$$

where g is a Gaussian kernel and u computes pairwise distance in Euclidean space.

A fourth option is

$$h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) = h_{\Theta}(\mathbf{x}_j - \mathbf{x}_i). \tag{6}$$

This encodes only local information, treating the shape as a collection of small patches and losing global structure.

Finally, a fifth option that we adopt in this article is an asymmetric edge function

$$h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) = \bar{h}_{\Theta}(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i). \tag{7}$$

This explicitly combines global shape structure, captured by the coordinates of the patch centers $\mathbf{x}_i$, with local neighborhood information, captured by $\mathbf{x}_j - \mathbf{x}_i$. In particular, we can define our operator by notating

$$e'_{ijm} = \text{ReLU}(\boldsymbol{\theta}_m \cdot (\mathbf{x}_j - \mathbf{x}_i) + \boldsymbol{\phi}_m \cdot \mathbf{x}_i), \tag{8}$$

which can be implemented as a shared MLP, and taking

$$x'_{im} = \max_{j:(i,j) \in \mathcal{E}} e'_{ijm}, \tag{9}$$

where $\Theta = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M, \boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_M)$.

3.2 Dynamic Graph Update

Our experiments suggest that it is beneficial to *recompute the graph* using nearest neighbors in the feature space produced by each layer. This is a crucial distinction of our method from graph CNNs working on a fixed input graph. Such a dynamic graph update is the reason for the name of our architecture, the *Dynamic Graph CNN (DGCNN)*. With dynamic graph updates, the receptive field is as large as the diameter of the point cloud, while being sparse.

At each layer, we have a different graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$, where the lth layer edges are of the form $(i, j_{i1}), \ldots, (i, j_{ik_l})$ such that $\mathbf{x}^{(l)}_{j_{i1}}, \ldots, x^{(l)}_{j_{ik_l}}$ are the k_l points closest to $\mathbf{x}^{(l)}_i$. Put differently, our architecture learns *how* to construct the graph $\mathcal{G}$ used in each layer rather than taking it as a fixed constant constructed before the network is evaluated. In our implementation, we compute a pairwise distance matrix in feature space and then take the closest k points for each single point.

3.3 Properties

Permutation Invariance. Consider the output of a layer,

$$\mathbf{x}'_i = \max_{j:(i,j) \in \mathcal{E}} h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j), \tag{10}$$

and a permutation operator π. The output of the layer $\mathbf{x}'_i$ is invariant to permutation of the input $\mathbf{x}_j$ because max is a symmetric function (other symmetric functions also apply). The global max pooling operator to aggregate point features is also permutation-invariant.

Translation Invariance. Our operator has a "partial" translation invariance property, in that our choice of edge functions Equation (7) explicitly exposes the part of the function that can be translation-dependent and optionally can be disabled. Consider a translation applied to $\mathbf{x}_j$ and $\mathbf{x}_i$; we can show that part of the edge feature is preserved when shifting by $\boldsymbol{T}$. In particular, for the translated point cloud, we have

$$\begin{aligned} e'_{ijm} &= \boldsymbol{\theta}_m \cdot (\mathbf{x}_j + T - (\mathbf{x}_i + T)) + \boldsymbol{\phi}_m \cdot (\mathbf{x}_i + T) \\ &= \boldsymbol{\theta}_m \cdot (\mathbf{x}_j - \mathbf{x}_i) + \boldsymbol{\phi}_m \cdot (\mathbf{x}_i + T). \end{aligned}$$

If we only consider $\mathbf{x}_j - \mathbf{x}_i$ by taking $\boldsymbol{\phi}_m = \mathbf{0}$, then the operator is fully invariant to translation. In this case, however, the model reduces to recognizing an object based on an unordered set of patches, ignoring the positions and orientations of patches. With both $\mathbf{x}_j - \mathbf{x}_i$ and $\mathbf{x}_i$ as input, the model takes account into the local geometry of patches while keeping global shape information.

3.4 Comparison to Existing Methods

DGCNN is related to two classes of approaches, PointNet and graph CNNs, which we show to be particular settings of our method. We summarize different methods in Table 1.

PointNet is a special case of our method with $k = 1$, yielding a graph with an empty edge set $\mathcal{E} = \varnothing$. The edge function used in PointNet is $h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) = h_{\Theta}(\mathbf{x}_i)$, which considers global but not local geometry. PointNet++ tries to account for local structure by applying PointNet in a local manner. In our parlance, PointNet++ first constructs the graph according to the Euclidean distances between the points, and in each layer applies a graph coarsening operation. For each layer, some points are selected using farthest point sampling (FPS); only the selected points are preserved while others are directly discarded after this layer. In this way, the graph becomes smaller after the operation applied on each layer. In contrast to DGCNN, PointNet++ computes pairwise distances using point input coordinates, and hence their graphs are fixed during training. The edge function used by PointNet++ is $h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j) = h_{\Theta}(\mathbf{x}_j)$, and the aggregation operation is also a max.

Among graph CNNs, MoNet (Monti et al. 2017a), ECC (Simonovsky and Komodakis 2017), Graph Attention Networks (Veličković et al. 2017), and the concurrent work (Atzmon et al. 2018) are the most related approaches. Their common denominator

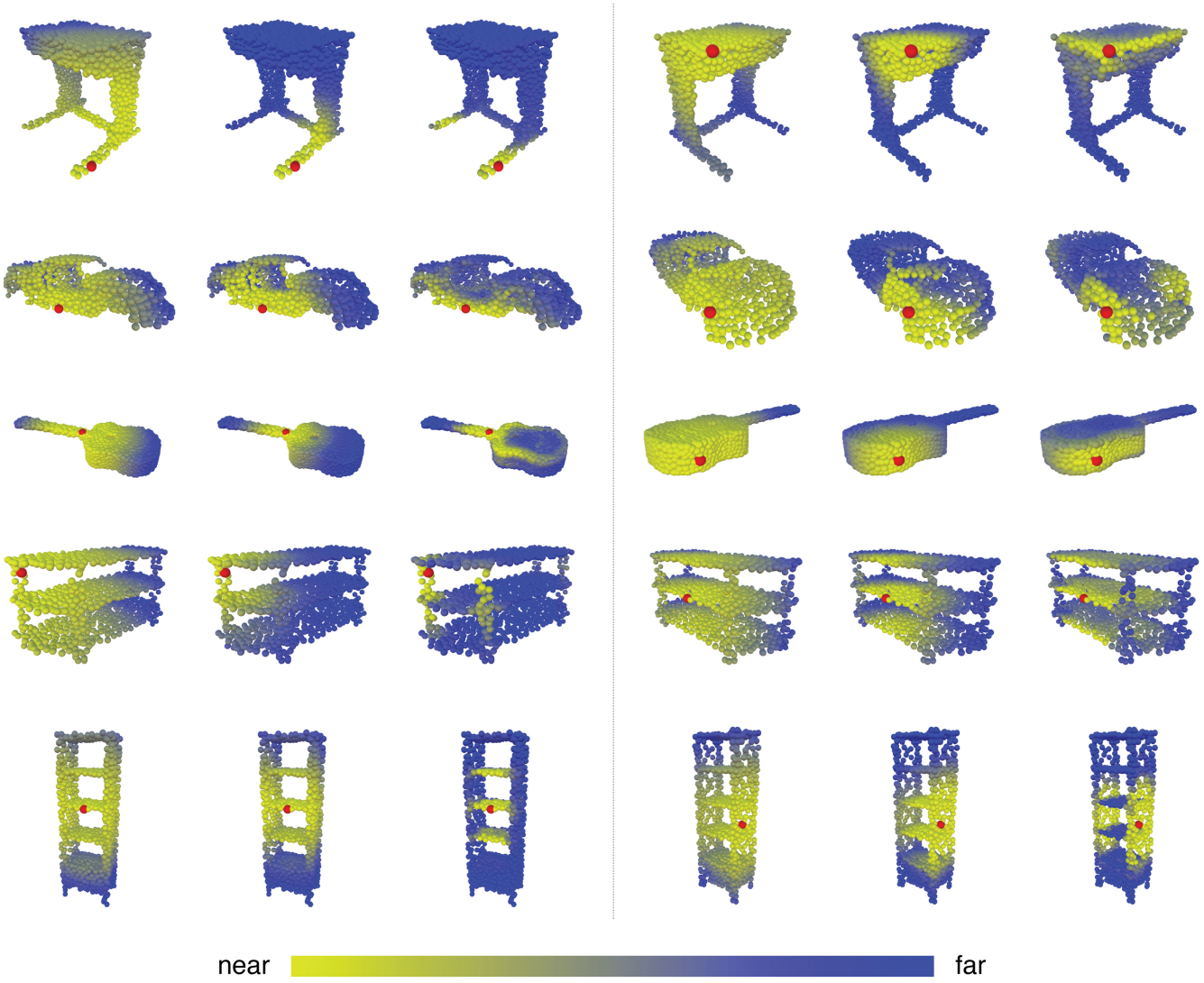


Fig. 4. Structure of the feature spaces produced at different stages of our shape classification neural network architecture, visualized as the distance between the red point to the rest of the points. For each set, Left: Euclidean distance in the input $\mathbb{R}^3$ space; Middle: Distance after the point cloud transform stage, amounting to a global transformation of the shape; Right: Distance in the feature space of the last layer. Observe how in the feature space of deeper layers semantically similar structures such as shelves of a bookshelf or legs of a table are brought close together, although they are distant in the original space.

is a notion of a local patch on a graph, in which a convolution-type operation can be defined.[1]

Specifically, Monti et al. (2017a) use the graph structure to compute a local "pseudo-coordinate system" $\mathbf{u}$ in which the neighborhood vertices are represented; the convolution is then defined as an M-component Gaussian mixture,

$$x'_{im} = \sum_{j:(i,j)\in\mathcal{E}} \boldsymbol{\theta}_m \cdot (\mathbf{x}_j \odot g_{\boldsymbol{w}_n}(u(\mathbf{x}_i, \mathbf{x}_j))), \tag{11}$$

where g is a Gaussian kernel, $\odot$ is the elementwise (Hadamard) product, $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\}$ encode the learnable parameters of the Gaussians (mean and covariance), and $\{\theta_1, \ldots, \theta_M\}$ are the learnable filter coefficients. Equation (11) is an instance of our general operation Equation (1), with a particular edge function

$$h_{\boldsymbol{\theta}_m, \boldsymbol{w}_n}(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\theta}_m \cdot (\mathbf{x}_j \odot g_{\boldsymbol{w}_n}(u(\mathbf{x}_i, \mathbf{x}_j)))$$

and $\square = \sum$. Again, their graph structure is fixed, and u is constructed based on the degrees of nodes.

Atzmon et al. (2018) can be seen as a special case of Monti et al. (2017a) with g as predefined Gaussian functions. Removing learnable parameters $(w_1, \ldots, w_N)$ and constructing a dense

[1] Simonovsky and Komodakis (2017) and Veličković et al. (2017) can be considered instances of Monti et al. (2017a), with the difference that the weights are constructed employing features from adjacent nodes instead of graph structure; Atzmon et al. (2018) is also similar except that the weighting function is hand-designed.

Table 2. Classification Results on ModelNet40

	Mean Class Accuracy	Overall Accuracy
3DShapeNets (Wu et al. 2015)	77.3	84.7
VoxNet (Maturana and Scherer 2015)	83.0	85.9
Subvolume (Qi et al. 2016)	86.0	89.2
VRN (single view) (Brock et al. 2016)	88.98	—
VRN (multiple views) (Brock et al. 2016)	91.33	—
ECC (Simonovsky and Komodakis 2017)	83.2	87.4
PointNet (Qi et al. 2017b)	86.0	89.2
PointNet++ (Qi et al. 2017c)	—	90.7
Kd-net (Klokov and Lempitsky 2017)	—	90.6
PointCNN (Li et al. 2018a)	88.1	92.2
PCNN (Atzmon et al. 2018)	—	92.3
Ours (baseline)	88.9	91.7
Ours	**90.2**	**92.9**
Ours (2048 points)	**90.7**	**93.5**

Table 3. Complexity, Forward Time, and Accuracy of Different Models

	Model size(MB)	Time(ms)	Accuracy(%)
PointNet (Baseline) (Qi et al. 2017b)	9.4	6.8	87.1
PointNet (Qi et al. 2017b)	40	16.6	89.2
PointNet++ (Qi et al. 2017c)	12	163.2	90.7
PCNN (Atzmon et al. 2018)	94	117.0	92.3
Ours (Baseline)	11	19.7	91.7
Ours	21	27.2	92.9

graph from point clouds, we have

$$x'_{im} = \sum_{j:j\in\mathcal{V}} (\theta_m \cdot \mathbf{x}_j) g(u(\mathbf{x}_i, \mathbf{x}_j)), \tag{12}$$

where u is the pairwise distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ in Euclidean space.

While MoNet and other graph CNNs assume a given fixed graph on which convolution-like operations are applied, to our knowledge our method is the first for which the graph changes from layer to layer and even on the same input during training when learnable parameters are updated. This way, our model not only learns how to extract local geometric features but also how to group points in a point cloud. Figure 4 shows the distance in different feature spaces, exemplifying that the distances in deeper layers carry semantic information over long distances in the original embedding.

4 EVALUATION

In this section, we evaluate the models constructed using EdgeConv for different tasks: classification, part segmentation, and semantic segmentation. We also visualize experimental results to illustrate key differences from previous work.

4.1 Classification

Data. We evaluate our model on the ModelNet40 (Wu et al. 2015) classification task, consisting in predicting the category of a previously unseen shape. The dataset contains 12,311 meshed CAD models from 40 categories. We used 9,843 models for training and 2,468 models for testing. We follow verbatim the experimental settings of Qi et al. (2017b). For each model, 1,024 points are uniformly sampled from the mesh faces; the point cloud is rescaled to fit into the unit sphere. Only the (x, y, z) coordinates of the sampled points are used, and the original meshes are discarded. During the training procedure, we augment the data by randomly scaling objects and perturbing the object and point locations.

Table 4. Effectiveness of Different Components

CENT	DYN	MPOINTS	Mean Class Accuracy(%)	Overall Accuracy(%)
			88.9	91.7
x			89.3	92.2
x	x		90.2	92.9
x	x	x	90.7	93.5

CENT denotes centralization, DYN denotes dynamical graph recomputation and MPOINTS denotes experiments with 2,048 points.

Table 5. Results of Our Model with Different Numbers of Nearest Neighbors

Number of nearest neighbors (k)	Mean Class Accuracy(%)	Overall Accuracy(%)
5	88.0	90.5
10	88.9	91.4
20	90.2	92.9
40	89.4	92.4

Architecture. The network architecture used for the classification task is shown in Figure 3 (top branch without spatial transformer network). We use four EdgeConv layers to extract geometric features. The four EdgeConv layers use three shared fully connected layers (64, 64, 128, 256). We recompute the graph based on the features of each EdgeConv layer and use the new graph for next layer. The number k of nearest neighbors is 20 for all EdgeConv layers (for the last row in Table 2, k is 40). Shortcut connections are included to extract multi-scale features and one shared fully connected layer (1,024) to aggregate multi-scale features, where we concatenate features from previous layers to get a $64 + 64 + 128 + 256 = 512$-dimensional point cloud. Then, a global max/sum pooling is used to get the point cloud global feature, after which two fully connected layers $(512, 256)$ are used to transform the global feature. Dropout with keep probability of 0.5 is used in the last two fully connected layers. All layers include LeakyReLU and batch normalization. The number k was chosen using a validation set. We split the training data to 80% for training and 20% for validation to search the best k. After k is chosen, we retrain the model on the whole training data and evaluate the model on the testing data. Other hyperparameters were chosen in a similar ways.

Training. We use SGD with learning rate 0.1, and we reduce the learning rate until 0.001 using cosine annealing (Loshchilov and Hutter 2017). The momentum for batch normalization is 0.9, and we do not use batch normalization decay. The batch size is 32 and the momentum is 0.9.

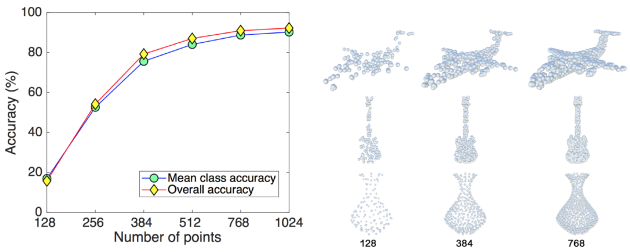


Fig. 5. Left: Results of our model tested with random input dropout. The model is trained with number of points being 1024 and k being 20. Right: Point clouds with different number of points. The numbers of points are shown below the bottom row.

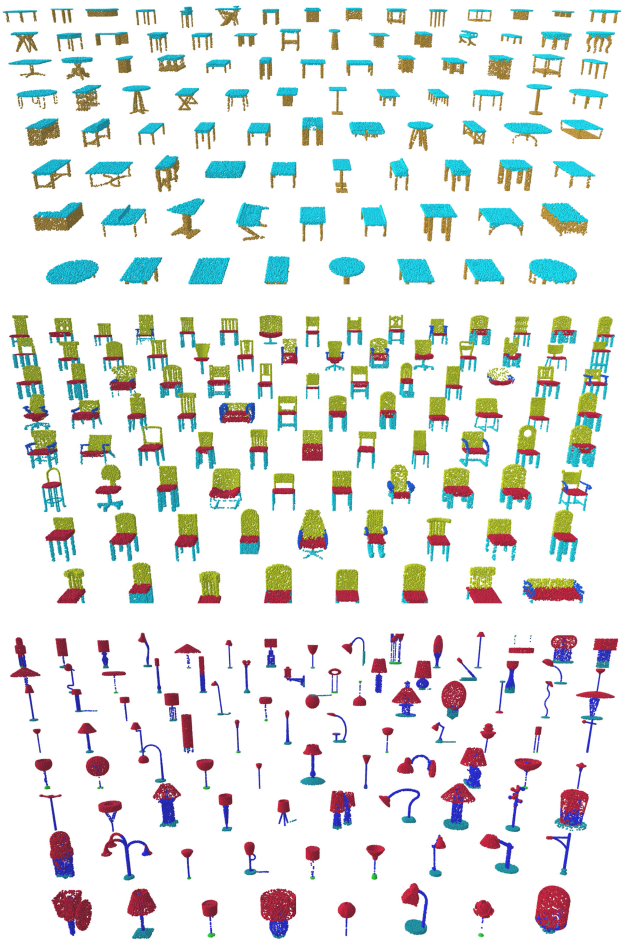


Fig. 6. Our part segmentation testing results for tables, chairs, and lamps.

Results. Table 2 shows the results for the classification task. Our model achieves the best results on this dataset. Our baseline using a fixed graph determined by proximity in the input point cloud is 1.0% better than PointNet++. An advanced version including dynamical graph recomputation achieves the best results on this dataset. All the experiments are performed with point clouds that contain 1024 points except last row. We further test out model with 2,048 points. The k used for 2,048 points is 40 to maintain the same density. Note that PCNN (Atzmon et al. 2018) uses additional augmentation techniques like randomly sampling 1,024 points out of 1,200 points during both training and testing.

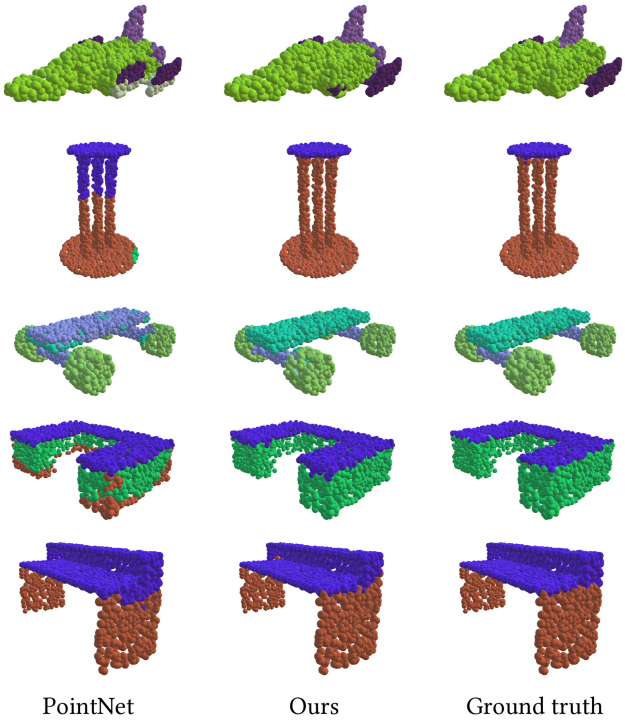


Fig. 7. Compare part segmentation results. For each set, from left to right: PointNet, ours, and ground truth.

4.2 Model Complexity

We use the ModelNet40 (Wu et al. 2015) classification experiment to compare the complexity of our model to previous state-of-the-art. Table 3 shows that our model achieves the best tradeoff between the model complexity (number of parameters), computational complexity (measured as forward pass time), and the resulting classification accuracy.

Our baseline model using the fixed k-NN graph outperforms the previous state-of-the-art PointNet++ by 1.0% accuracy, at the same time being seven times faster. A more advanced version of our model including a dynamically updated graph computation outperforms PointNet++, PCNN by 2.2% and 0.6%, respectively, while being much more efficient. The number of points in each experiment is also 1,024 in this section.

4.3 More Experiments on ModelNet40

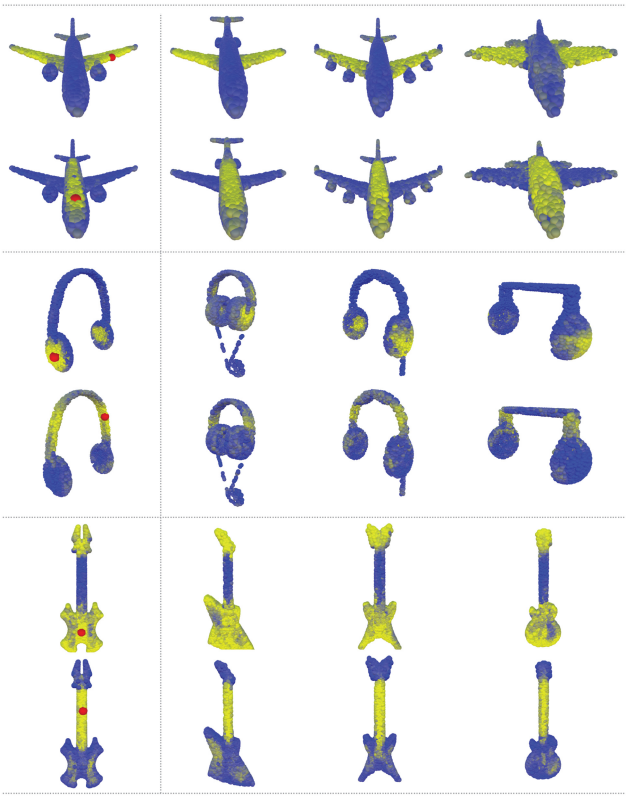
We also experiment with various settings of our model on the ModelNet40 (Wu et al. 2015) dataset. In particular, we analyze the effectiveness of the different distance metrics, explicit usage of $\mathbf{x}_i - \mathbf{x}_j$, and more points.

Table 4 shows the results. "Centralization" denotes using concatenation of $\mathbf{x}_i$ and $\mathbf{x}_i - \mathbf{x}_j$ as the edge features rather than concatenating $\mathbf{x}_i$ and $\mathbf{x}_j$. "Dynamic graph recomputation" denotes we reconstruct the graph rather than using a fixed graph. Explicitly

Table 6. Part Segmentation Results on ShapeNet Part Dataset

	mean	AREO	BAG.	CAP	CAR	CHAIR	EAR PHONE	GUITAR	KNIFE	LAMP	LAPTOP	MOTOR	MUG	PISTOL	ROCKET	SKATE BOARD	TABLE
# SHAPES		2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271
POINTNET	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
POINTNET++	85.1	82.4	79.0	**87.7**	77.3	**90.8**	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
KD-NET	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
LOCALFEATURENET	84.3	**86.1**	73.0	54.9	77.4	88.8	55.0	90.6	86.5	75.2	**96.1**	57.3	91.7	83.1	53.9	72.5	**83.8**
PCNN	85.1	82.4	80.1	85.5	79.5	**90.8**	73.2	91.3	86.0	85.0	95.7	73.2	94.8	83.3	51.0	75.0	81.8
POINTCNN	**86.1**	84.1	**86.45**	86.0	**80.8**	90.6	**79.7**	**92.3**	**88.4**	**85.3**	**96.1**	**77.2**	**95.3**	**84.2**	**64.2**	**80.0**	83.0
OURS	85.2	84.0	83.4	86.7	77.8	90.6	74.7	91.2	87.5	82.8	95.7	66.3	94.9	81.1	63.5	74.5	82.6

Metric is mIoU(%) on points.



Source points Other point clouds from the same category

Fig. 8. Visualize the Euclidean distance (yellow: near; blue: far) between source points (red points in the left column) and multiple point clouds from the same category in the feature space after the third EdgeConv layer. Notice source points not only capture semantically similar structures in the point clouds that they belong to but also capture semantically similar structures in other point clouds from the same category.

centralizing each patch by using the concatenation of $\mathbf{x}_i$ and $\mathbf{x}_i - \mathbf{x}_j$ leads to about 0.5% improvement for overall accuracy. By dynamically updating graph, there is about 0.7% improvement, and Figure 4 also suggests that the model can extract semantically meaningful features. Using more points further improves the overall accuracy by 0.6%.

We also experiment with different numbers k of nearest neighbors as shown in Table 5. For all experiments, the number of points is still 1,024. While we do not exhaustively experiment with all possible k, we find with large k that the performance degenerates. This confirms our hypothesis that for certain density, with large k the Euclidean distance fails to approximate geodesic distance, destroying the geometry of each patch.

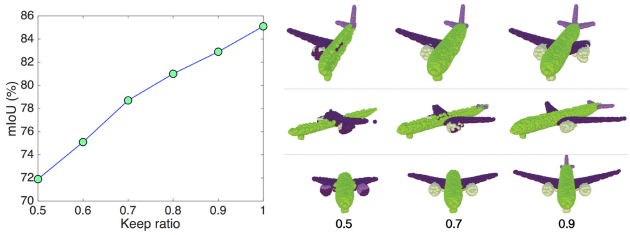


Fig. 9. Left: The mean IoU (%) improves when the ratio of kept points increases. Points are dropped from one of six sides (top, bottom, left, right, front, and back) randomly during evaluation process. Right: Part segmentation results on partial data. Points on each row are dropped from the same side. The keep ratio is shown below the bottom row. Note that the segmentation results of turbines are improved when more points are included.

Table 7. 3D Semantic Segmentation Results on S3DIS

	MEAN IoU	OVERALL ACCURACY
POINTNET (BASELINE) (QI ET AL. 2017B)	20.1	53.2
POINTNET (QI ET AL. 2017B)	47.6	78.5
MS + CU(2) (ENGELMANN ET AL. 2017)	47.8	79.2
G + RCU (ENGELMANN ET AL. 2017)	49.7	81.1
POINTCNN (LI ET AL. 2018A)	**65.39**	—
OURS	56.1	**84.1**

MS+CU for multi-scale block features with consolidation units; G+RCU for the grid-blocks with recurrent consolidation units.

We further evaluate the robustness of our model (trained on 1,024 points with $k = 20$) to point cloud density. We simulate the environment that random input points drops out during testing. Figure 5 shows that even half of points is dropped, the model still achieves reasonable results. With fewer than 512 points, however, performance degenerates dramatically.

4.4 Part Segmentation

Data. We extend our EdgeConv model architectures for part segmentation task on ShapeNet part dataset (Yi et al. 2016). For this task, each point from a point cloud set is classified into one of a few predefined part category labels. The dataset contains 16,881

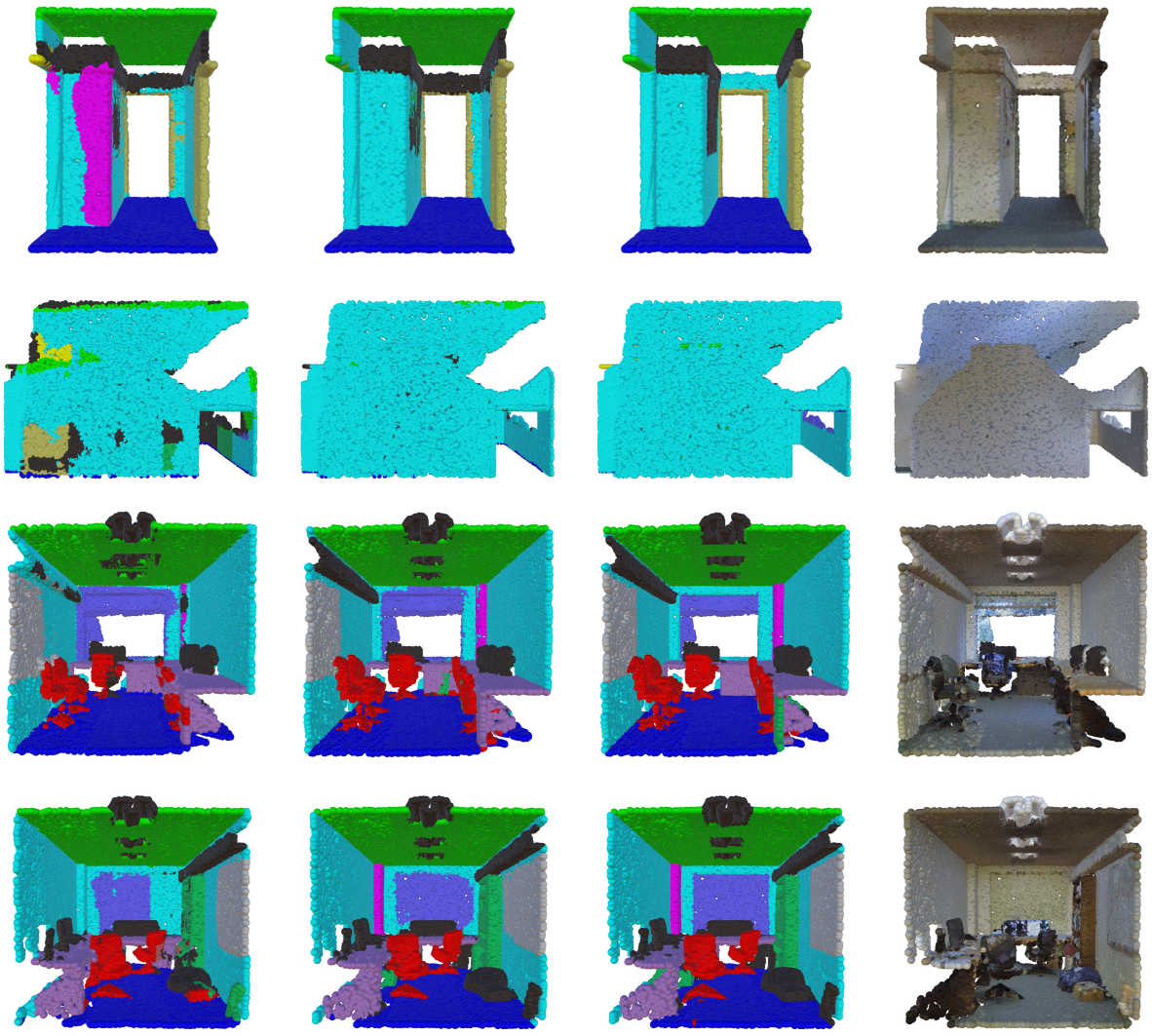


Fig. 10. Semantic segmentation results. From left to right: PointNet, ours, ground truth, and point cloud with original color. Notice our model outputs smoother segmentation results, for example, wall (cyan) in top two rows, chairs (red) and columns (magenta) in bottom two rows.

3D shapes from 16 object categories, annotated with 50 parts in total. We sampled 2,048 points from each training shape, and most sampled point sets are labeled with less than six parts. We follow the official train/validation/test split scheme as Chang et al. (2015) in our experiment.

Architecture. The network architecture is illustrated in Figure 3 (bottom branch). After a spatial transformer network, three EdgeConv layers are used. A shared fully connected layer (1,024) aggregates information from the previous layers. Shortcut connections are used to include all the EdgeConv outputs as local feature descriptors. At last, three shared fully connected layers (256, 256, 128) are used to transform the pointwise features. Batchnorm, dropout, and ReLU are included in the similar fashion to our classification network.

Training. The same training setting as in our classification task is adopted. A distributed training scheme is further implemented on two NVIDIA TITAN X GPUs to maintain the training batch size.

Results. We use Intersection-over-Union (IoU) on points to evaluate our model and compare with other benchmarks. We follow the same evaluation scheme as PointNet: The IoU of a shape is computed by averaging the IoUs of different parts occurring in that shape, and the IoU of a category is obtained by averaging the IoUs of all the shapes belonging to that category. The mean IoU (mIoU) is finally calculated by averaging the IoUs of all the testing shapes. We compare our results with PointNet (Qi et al. 2017b), PointNet++ (Qi et al. 2017c), Kd-Net (Klokov and Lempitsky 2017), LocalFeatureNet (Shen et al. 2017), PCNN (Atzmon et al. 2018), and PointCNN (Li et al. 2018a). The evaluation results are shown in

Table 6. We also visually compare the results of our model and PointNet in Figure 7. More examples are shown in Figure 6.

Intra-cloud Distances. We next explore the relationships between different point clouds captured using our features. As shown in Figure 8, we take one red point from a source point cloud and compute its distance in feature space to points in other point clouds from the same category. An interesting finding is that although points are from different sources, they are close to each other if they are from semantically similar parts. We evaluate on the features after the third layer of our segmentation model for this experiment.

Segmentation on Partial Data. Our model is robust to partial data. We simulate the environment that part of the shape is dropped from one of six sides (top, bottom, right, left, front, and back) with different percentages. The results are shown in Figure 9. On the left, the mean IoU versus "keep ratio" is shown. On the right, the results for an airplane model are visualized.

4.5 Indoor Scene Segmentation

Data. We evaluate our model on Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) (Armeni et al. 2016) for a semantic scene segmentation task. This dataset includes 3D scan point clouds for 6 indoor areas including 272 rooms in total. Each point belongs to one of 13 semantic categories—e.g., board, bookcase, chair, ceiling, and beam—plus clutter. We follow the same setting as Qi et al. (2017b), where each room is split into blocks with area $1m \times 1m$, and each point is represented as a 9D vector (XYZ, RGB, and normalized spatial coordinates). We sampled 4,096 points for each block during training process, and all points are used for testing. We also use the same sixfold cross validation over the six areas, and the average evaluation results are reported.

The model used for this task is similar to part segmentation model, except that a probability distribution over semantic object classes is generated for each input point and no categorical vector is used here. We compare our model with both PointNet (Qi et al. 2017b) and PointNet baseline, where additional point features (local point density, local curvature, and normal) are used to construct handcrafted features and then fed to an MLP classifier. We further compare our work with Engelmann et al. (2017) and PointCNN (Li et al. 2018a). Engelmann et al. (2017) present network architectures to enlarge the receptive field over the 3D scene. Two different approaches are proposed in their work: MS+CU for multi-scale block features with consolidation units; G+RCU for the grid-blocks with recurrent consolidation Units. We report evaluation results in Table 7 and visually compare the results of PointNet and our model in Figure 10.

5 DISCUSSION

In this work, we propose a new operator for learning on point cloud and show its performance on various tasks. Our model suggests that local geometric features are important to 3D recognition tasks, even after introducing machinery from deep learning.

While our architectures easily can be incorporated as-is into existing pipelines for point cloud-based graphics, learning, and vision, our experiments also indicate several avenues for future research and extension. Some details of our implementation could be revised and/or re-engineered to improve efficiency or scalability, e.g. incorporating fast data structures rather than computing pairwise distances to evaluate k-nearest neighbors queries. We also could consider higher-order relationships between larger tuples of points, rather than considering them pairwise. Another possible extension is to design a non-shared transformer network that works on each local patch differently, adding flexibility to our model.

Our experiments suggest that intrinsic features can be equally valuable if not more valuable than point coordinates; developing a practical and theoretically justified framework for balancing intrinsic and extrinsic considerations in a learning pipeline will require insight from theory and practice in geometry processing. Given this, we will consider applications of our techniques to more abstract point clouds coming from applications like document retrieval and image processing rather than 3D geometry; beyond broadening the applicability of our technique, these experiments will provide insight into the role of geometry in abstract data processing.

REFERENCES

Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 2016. 3D semantic parsing of large-scale indoor spaces. In *Proceedings of the CVPR*.

Matan Atzmon, Haggai Maron, and Yaron Lipman. 2018. Point convolutional neural networks by extension operators. *ACM Trans. Graph.* 37, 4, Article 71 (July 2018), 12 pages. DOI: https://doi.org/10.1145/3197517.3201301

Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. 2011. The wave kernel signature: A quantum mechanical approach to shape analysis. In *Proceedings of the ICCV Workshops*.

Serge Belongie, Jitendra Malik, and Jan Puzicha. 2001. Shape context: A new descriptor for shape matching and object recognition. In *Proceedings of the NIPS*.

Silvia Biasotti, Andrea Cerri, A. Bronstein, and M. Bronstein. 2016. Recent trends, applications, and perspectives in 3D shape similarity assessment. *Comput. Graph. Forum* 35, 6 (2016), 87–119.

Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. 2016. Learning shape correspondence with anisotropic convolutional neural networks. In *Proceedings of the NIPS*.

Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J. Weston. 2016. Generative and discriminative voxel modeling with convolutional neural networks. In *Proceedings of the NIPS*.

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, 4 (2017), 18–42.

Michael M. Bronstein and Iasonas Kokkinos. 2010. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Proceedings of the CVPR*.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203* (2013).

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su et al. 2015. Shapenet: An information-rich 3D model repository. *arXiv:1512.03012* (2015).

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the NIPS*.

Francis Engelmann, Theodora Kontogianni, Alexander Hermans, and Bastian Leibe. 2017. Exploring spatial context for 3D semantic segmentation of point clouds. In *Proceedings of the CVPR*.

Danielle Ezuz, Justin Solomon, Vladimir G. Kim, and Mirela Ben-Chen. 2017. GWCNN: A metric alignment layer for deep shape analysis. *Comput. Graph. Forum* 36, 5 (2017), 49–57.

Haoqiang Fan, Hao Su, and Leonidas J. Guibas. 2017. A point set generation network for 3D object reconstruction from a single image. In *Proceedings of the CVPR*.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. 2018. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *Proceedings of the CVPR*.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. *arXiv:1704.01212* (2017).

Aleksey Golovinskiy, Vladimir G. Kim, and Thomas Funkhouser. 2009. Shape-based recognition of 3D point clouds in urban environments. In *Proceedings of the ICCV*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proceedings of the NIPS*.

Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. 2018. PCPNet: Learning local shape properties from raw point clouds. *Comput. Graph. Forum* 37, 2 (2018), 75–85. DOI: https://doi.org/10.1111/cgf.13343

Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, and Jianwei Wan. 2014. 3D object recognition in cluttered scenes with local surface features: A survey. *Trans. PAMI* 36, 11 (2014), 2270–2287.

Oshri Halimi, Or Litany, Emanuele Rodolà, Alex Bronstein, and Ron Kimmel. 2018. Self-supervised learning of dense shape correspondence. *arXiv:1812.02415* (2018).

M. Henaff, J. Bruna, and Y. LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv:1506.05163* (2015).

Andrew E. Johnson and Martial Hebert. 1999. Using spin images for efficient object recognition in cluttered 3D scenes. *Trans. PAMI* 21, 5 (1999), 433–449.

Diederik P. Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv:1312.6114* (2013).

Thomas N. Kipf and Max Welling. 2017. Semi-Supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*.

Roman Klokov and Victor Lempitsky. 2017. Escape from cells: Deep Kd-networks for the recognition of 3D point cloud models. (2017).

Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. 2017. Surface networks. In *Proceedings of the CVPR*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the NIPS*.

Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. 1989. Backpropagation applied to handwritten ZIP code recognition. *Neural Comput.* 1, 4 (1989), 541–551.

Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. 2017. CayleyNets: Graph convolutional neural networks with complex rational spectral filters. *arXiv:1705.07664* (2017).

Chun-Liang Li, Manzil Zaheer, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov. 2018b. Point cloud GAN. *arXiv:1810.05795* (2018).

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. 2018a. PointCNN: Convolution On X-transformed points. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 820–830. Retrieved from http://papers.nips.cc/paper/7362-pointcnn-convolution-on-x-transformed-points.pdf.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. In *Proceedings of the ICLR*.

Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. 2018. Deep continuous fusion for multi-sensor 3D object detection. In *Proceedings of the ECCV*.

Haibin Ling and David W. Jacobs. 2007. Shape classification using the inner-distance. *Trans. PAMI* 29, 2 (2007), 286–299.

Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. 2017a. Deformable shape completion with graph convolutional autoencoders. *arXiv:1712.00268* (2017).

Or Litany, Tal Remez, Emanuele Rodolà, Alex M. Bronstein, and Michael M. Bronstein. 2017b. Deep functional maps: Structured prediction for dense shape correspondence. In *Proceedings of the ICCV*.

I. Loshchilov and F. Hutter. 2017. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the ICLR*.

Min Lu, Yulan Guo, Jun Zhang, Yanxin Ma, and Yinjie Lei. 2014. Recognizing objects in 3D point clouds with multi-scale local features. *Sensors* 14, 12 (2014), 24156–24173.

Siddharth Manay, Daniel Cremers, Byung-Woo Hong, Anthony J. Yezzi, and Stefano Soatto. 2006. Integral invariants for shape matching. *Trans. PAMI* 28, 10 (2006), 1602–1618.

Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G Kim, and Yaron Lipman. 2017. Convolutional neural networks on surfaces via seamless toric covers. In *Proceedings of the SIGGRAPH*.

Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. 2015. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the 3dRR*.

Daniel Maturana and Sebastian Scherer. 2015. Voxnet: A 3D convolutional neural network for real-time object recognition. In *Proceedings of the IROS*.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. 2017a. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proceedings of the CVPR*.

F. Monti, M. M. Bronstein, and X. Bresson. 2017b. Geometric matrix completion with recurrent multi-graph neural networks. In *Proceedings of the NIPS*.

Federico Monti, Karl Otness, and Michael M. Bronstein. 2018. MotifNet: A motif-based graph convolutional network for directed graphs. *arXiv:1802.01572* (2018).

Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. 2012. Functional maps: A flexible representation of maps between shapes. *Trans. Graph.* 31, 4 (2012), 30.

Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. 2017a. Frustum PointNets for 3D object detection from RGB-D data. *arXiv:1711.08488* (2017).

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017b. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the CVPR*.

Charles R. Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. 2016. Volumetric and multi-view CNNs for object classification on 3D data. In *Proceedings of the CVPR*.

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017c. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the NIPS*.

Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. 2018. Generating 3D faces using convolutional mesh autoencoders. *arXiv:1807.10267* (2018).

Raif M. Rustamov. 2007. Laplace-beltrami eigenfunctions for deformation invariant shape representation. In *Proceedings of the SGP*.

Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. 2009. Fast point feature histograms (FPFH) for 3D registration. In *Proceedings of the ICRA*.

Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. 2008a. Aligning point cloud views using persistent feature histograms. In *Proceedings of the IROS*.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. 2008b. Towards 3D point cloud-based object maps for household environments. *Robot. Auton. Syst. J.* 56, 11 (Nov. 2008), 927–941.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Tran. Neural Networks* 20, 1 (2009), 61–80.

Syed Afaq Ali Shah, Mohammed Bennamoun, Farid Boussaid, and Amar A. El-Sallam. 2013. 3D-Div: A novel local surface descriptor for feature matching and pairwise range image registration. In *Proceedings of the ICIP*.

Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. 2017. Neighbors do help: Deeply exploiting local structures of point clouds. *arXiv:1712.06760* (2017).

David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Process. Mag.* 30, 3 (2013), 83–98.

Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the CVPR*.

Ayan Sinha, Jing Bai, and Karthik Ramani. 2016. Deep learning 3D shape surfaces using geometry images. In *Proceedings of the ECCV*.

Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. 2018. SPLATNet: Sparse lattice networks for point cloud processing. In *Proceedings of the CVPR*. 2530–2539.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. 2015. Multi-view convolutional neural networks for 3D shape recognition. In *Proceedings of the CVPR*.

Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. 2009. A concise and provably informative multi-scale signature based on heat diffusion. *Comput. Graph. Forum* 28, 5 (2009), 1383–1392.

Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. 2017. Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs. In *Proceedings of the ICCV*.

Federico Tombari, Samuele Salti, and Luigi Di Stefano. 2011. A combined texture-shape descriptor for enhanced 3D feature matching. In *Proceedings of the ICIP*.

Oliver Van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. 2011. A survey on shape correspondence. *Comput. Graph. Forum* 30, 6 (2011), 1681–1707.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph attention networks. *arXiv:1710.10903*.

Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. 2018b. Deep parametric continuous convolutional neural networks. In *Proceedings of the CVPR*.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. 2018a. Non-local neural networks. In *Proceedings of the CVPR*.

Lingyu Wei, Qixing Huang, Duygu Ceylan, Etienne Vouga, and Hao Li. 2016. Dense human body correspondences using convolutional networks. In *Proceedings of the CVPR*.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D shapenets: A deep representation for volumetric shapes. In *Proceedings of the CVPR*.

Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan Yuille, and Kaiming He. 2018. Feature denoising for improving adversarial robustness. *arXiv:1812.03411*.

Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. 2018. FoldingNet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the CVPR*.

Li Yi, Vladimir G. Kim, Duygu Ceylan, I. Shen, Mengyan Yan, Hao Su, A. R. Cewu Lu, Qixing Huang, Alla Sheffer, Leonidas Guibas et al. 2016. A scalable active framework for region annotation in 3D shape collections. *Trans. Graph.* 35, 6 (2016), 210.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Proceedings of the ICRA*.

Received January 2019; revised May 2019; accepted June 2019