Dynamic Graph CNN for Learning on Point Clouds

Yue Wang* MIT yuewang@csail.mit.edu
Yongbin Sun* MIT yb_sun@mit.edu
Ziwei Liu UC Berkeley zwliu@icsi.berkeley.edu
Sanjay E. Sarma MIT sesarma@mit.edu
Michael M. Bronstein USI / TAU / Intel michael.bronstein@usi.ch
Justin M. Solomon MIT jsolomon@mit.edu

Abstract

Point clouds provide a flexible and scalable geometric representation suitable for countless applications in computer graphics; they also comprise the raw output of most 3D data acquisition devices. Hence, the design of intelligent computational models that act directly on point clouds is critical, especially when efficiency considerations or noise preclude the possibility of expensive denoising and meshing procedures. 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. 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 is differentiable and can be plugged into existing architectures. Compared to existing modules operating largely in extrinsic space or treating each point independently, EdgeConv has several appealing properties: It incorporates local neighborhood information; it can be stacked or recurrently 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. Beyond proposing this module, we provide extensive evaluation and analysis revealing that EdgeConv captures and exploits fine-grained geometric properties of point clouds. The proposed approach achieves state-of-the-art performance on standard benchmarks including ModelNet40 and S3DIS.*

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 [57], self-driving vehicles [33], robotics [40], and shape synthesis and modeling [14].

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 paper, we primarily consider point cloud classification and segmentation, two model tasks in the point cloud processing world. Traditional methods for solving these problems employ handcrafted features to capture geometric properties of point clouds [26, 38, 39]. 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 [10].

Adaptation of deep learning to point cloud data, however, is far from straightforward. Most critically, standard deep neural network models take as 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

*Equal Contribution

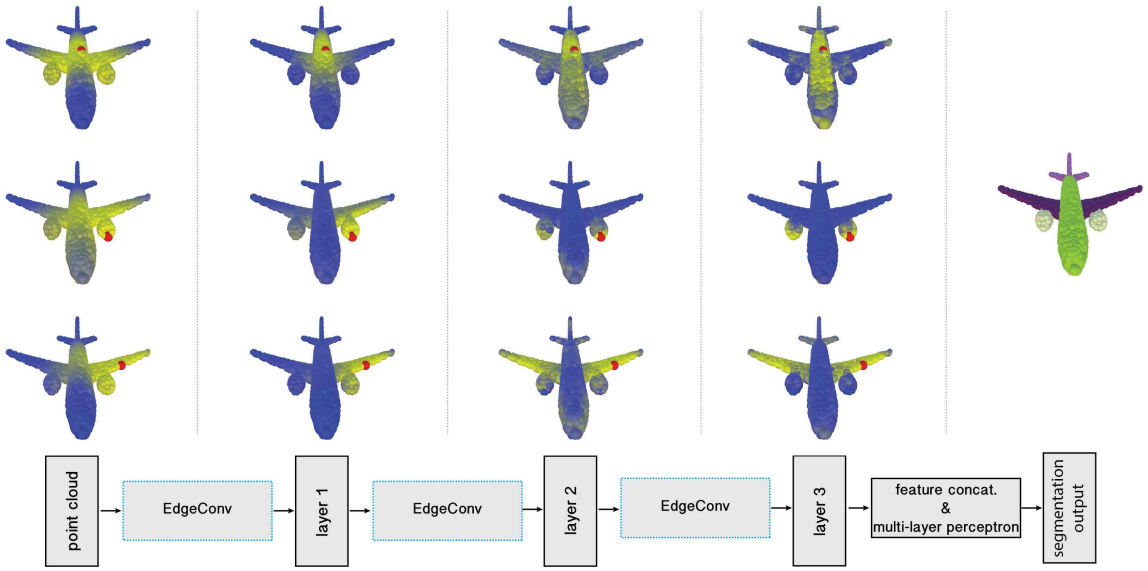


Figure 1. **Point cloud segmentation using the proposed neural network.** Bottom: schematic neural network architecture. Top: Structure of the feature spaces produced at different layers of the network, visualized as the distance from the red point to all the rest of the points (shown left-to-right are 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.

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 [30, 54]. 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* [34], 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 [36, 43]; 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 leads to local features missing.

To address these drawbacks, we propose a novel simple operation, called EdgeConv, which captures local geometric structure while maintaining permutation invariance. Instead of generating points' 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 permutation invariant.

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 performance improvement by a large margin; 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 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 the graph.
- 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.

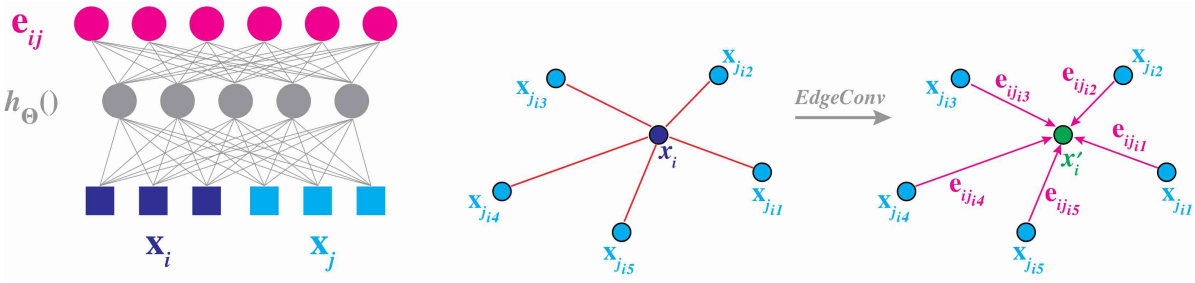


Figure 2. **Left**: An example of computing an edge feature, e_{ij}, from a point pair, x_i and x_j. In this example, $h_\Theta()$ is instantiated using a fully connected layer, and the learnable parameters are its associated weights and bias. **Right**: Visualize the EdgeConv operation. The output of EdgeConv is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

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 paper, but we refer the reader to [51, 15, 4] for comprehensive 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 [3], spin images [17], integral features [27], distance-based descriptors [24], point feature histograms [39, 38], and normal histograms [50], to name a few. Intrinsic descriptors treat the 3D shape as a manifold whose metric structure is discretized as a mesh or graph; quantities expressed in terms of the metric are by definition intrinsic and invariant to isometric deformation. Representatives of this class include spectral descriptors such as global point signatures [37], the heat and wave kernel signatures [48, 2], and variants [8]. Most recently, several approaches wrap machine learning schemes around standard descriptors [15, 42].

Learned Features. In computer vision, approaches relying on 'hand-crafted' features have reached a plateau in performance on challenging image analysis problems like image recognition. A breakthrough came with the use of convolutional neural networks (CNNs) [22, 21], leading to an overwhelming trend to abandon hand-crafted features in favor of models that learn task-specific features from data.

A basic CNN architecture is the *deep neural network*, which interleaves convolutional and pooling layers to aggregate local information in images. This success of deep learning for images suggests the value of adapting related insight to geometric data like point clouds. Unlike images, however, geometric data usually are not on an underlying grid, requiring new definitions for building blocks like convolution and pooling.

Existing 3D deep learning methods can be split into two classes. View-based and volumetric representations exemplify techniques that try to "place" geometric data onto a grid and apply existing deep learning algorithms to the adapted structure. Other methods replace the standard building blocks of deep neural architectures with special operations suitable for unstructured geometric data [29, 6, 31, 34, 36]. We provide details about the closest techniques to ours below.

View-based Methods View-based techniques represent a 3D object as a collection of 2D views, to which standard CNNs used in image analysis can be applied. Typically, a CNN is applied to each view and then the resulting features are aggregated by a view pooling procedure [47]. View-based approaches are also good match for applications where the input comes from a 3D sensor and represented as a range image [53], in which case a single view can be used.

Volumetric Methods Voxelization is a straightforward way to convert unstructured geometric data to a regular 3D grid over which standard CNN operations can be applied [30, 54]. These volumetric representations are often wasteful, since voxelization produces a sparsely-occupied 3D grid. Time and space complexity considerations limit the resolution of the volumetric grids, yielding quantization artifacts. Recent space partition methods like k-d trees [20] or octrees [49] remedy some resolution issues but still rely on subdivision of a bounding volume rather than local geometric structure. Finally, [35] studied a combination of view-based and volumetric approaches for 3D shape classification.

PointNets PointNets [34] comprise a special class of architectures for point sets like 3D point clouds. The key ingredient is a symmetric function applied to 3D coordinates in a manner invariant to permutation. While they achieve impressive performance on point cloud analysis tasks, *PointNets* treat each point individually, essentially learning a mapping from 3D to the latent features without leveraging local geometric structure. Furthermore, the learned mapping is sensitive to the global transformation of the point cloud; to cope with this issue, PointNet employs a complex and computationally expensive spatial transformer network [16] to learn 3D alignment.

Local information is important for feature learning in two ways. First, as for handcrafted descriptors, local features usually account for geometric relationships among neighboring points to be robust to various transformations. Second, local information is critical to the succcess of image-based deep convolutional architectures. Follow-up work proposed an improved *PointNet++* architecture exploiting geometric features in local point sets and hierarchically aggregating them for inference [36]. A similar approach is proposed in [43], where initial point features are obtained from a point kernel correlation layer and then aggregated among nearby points. Benefiting from local structure, *PointNet++* achieves state-of-the-art results on several point cloud analysis benchmarks. *PointNet++*, however, still treats individual points in local point sets independently and does not consider relationships between point pairs.

Geometric Deep Learning PointNets exemplify a broad class of deep learning architectures on non-Euclidean structured data termed *geometric deep learning* [7]. These methods date back to early methods to construct neural networks on graphs [41]. More recently, [9] proposed a generalization of convolution for graphs via the Laplacian operator [44]. This foundational approach had a number of drawbacks including the computational complexity of Laplacian eigendecomposition, the large number of parameters to express the convolutional filters, and a lack of spatial localization. These issues are alleviated in follow-up work using polynomial [11, 19] or rational [23] spectral filters that avoid the Laplacian eigendecomposition and also guarantee localization.

Spectral graph CNN models are notable for isometry invariance and hence have applied to non-rigid shape analysis [5]. A key difficulty, however, is that the Laplacian eigenbasis is domain-dependent; thus, a filter learned on one shape may not generalize to others. Spectral transformer networks address this problem to some extent [56].

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 [29]. Its key advantage over spectral approaches is better generalization. Follow-up work proposed different local charting techniques using anisotropic diffusion [6] or Gaussian mixture models [52, 31]. [25] incorporate a differentiable functional map [32] layer 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 attempt to pull back a convolution operation by embedding the shape into a domain with shift-invariant structure such as the sphere [46], torus [28], or plane [13].

a

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 graph neural networks. We show in the following that such an operation, dubbed *edge convolution* (EdgeConv), has the properties of lying between translation-invariant and non-locality.

Differently from graph CNNs, the graph is not fixed but rather is dynamically updated after each layer of the network. That is, the 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.

3.1. Edge Convolution

Consider a F-dimensional point cloud with n points, denoted by $X = \{x_1, \dots, x_n\} \subseteq R^F$. In the simplest setting of $F = 3$, each point contains 3D coordinates $x_i = (x_i, y_i, z_i)$; 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 further assume to be given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the local structure of the point cloud, where $\mathcal{V} = \{1, \dots, n\}$ 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 in $\mathbb{R}^F$, containing directed edges of the form $(i, j_{i1}), \dots, (i, j_{ik})$ such that points $x_{j_{i1}}, \dots, x_{j_{ik}}$ are the closest to x_i. We define *edge features* as $e_{ij} = h_{\mathbf{\Theta}}(x_i, x_j)$, where $h_{\mathbf{\Theta}} : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^{F'}$ is some parametric non-linear function parameterized by the set of learnable parameters $\mathbf{\Theta}$.

Finally, we define the EdgeConv operation by applying a channel-wise symmetric aggregation operation $\square$ (e.g., $\sum$

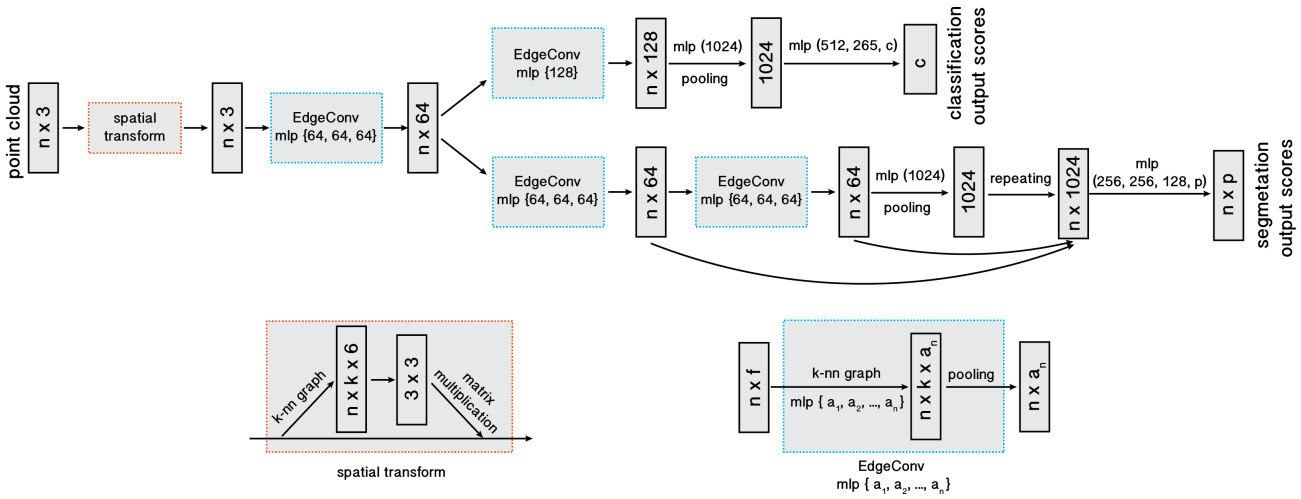


Figure 3. **Model architectures:** The model architectures used for classification (top branch) and segmentation (bottom branch). The classification model takes as input n points, calculates an edge feature set of size k for each point at an EdgeConv layer, and aggregates features within each set to 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. For illustration purposes, two arrowed arcs are plotted to represent feature 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 $\{a_1, a_2, ..., a_n\}$, and generates a tensor of shape $n \times a_n$ after pooling among neighboring edge features.

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

$$x'_i = \square_{j:(i,j)\in\mathcal{E}} h_{\mathbf{\Theta}}(x_i, x_j). \tag{1}$$

Making analogy to the classical convolution operation in images, we can regard x_i as the central pixel and $\{x_j : (i,j) \in \mathcal{E}\}$ 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 the resulting EdgeConv operation.

First, note that in the setting when $x_1, \ldots, x_n$ represent image pixels layed out on a regular grid and the graph has a local connectivity representing patches of fixed size around each pixel, the choice $h_{\mathbf{\Theta}}(x_i, x_j) = \theta_j x_j$ as the edge function and sum as the aggregation operation yields the classical Euclidean convolution,

$$x'_i = \sum_{j:(i,j)\in\mathcal{E}} \theta_j x_j,$$

where the parameters $\mathbf{\Theta} = (\theta_i, \ldots, \theta_k)$ act as the weights of the filter.

The second possible choice of h is $h_{\mathbf{\Theta}}(x_i, x_j) = h_{\mathbf{\Theta}}(x_i)$, 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 particular choice of our EdgeConv.

A third option is $h_{\mathbf{\Theta}}(x_i, x_j) = h_{\mathbf{\Theta}}(x_j - x_i)$. Note that such a choice encodes only local information, essentially treating the shape as a collection of small patches and losing the global shape structure.

Finally, the fourth option, which we adopt in this paper, is an asymmetric edge function of the form $h_{\mathbf{\Theta}}(x_i, x_j) = h_{\mathbf{\Theta}}(x_i, x_j - x_i)$. Such a function combines both the global shape structure (captured by the coordinates of the patch centers x_i) and local neighborhood information (captured by $x_j - x_i$).

3.2. Dynamic Graph CNNs

Similarly to classical CNNs used in computer vision, the EdgeConv operation can be applied multiple times, possibly interleaved with pooling depending on the task at hand. When applied without pooling, multiple applications of EdgeConv produce effectively larger support ('receptive field') of the filter. We denote by $X^{(l)} = \{x_1^{(l)}, \ldots, x_n^{(l)}\} \subseteq$

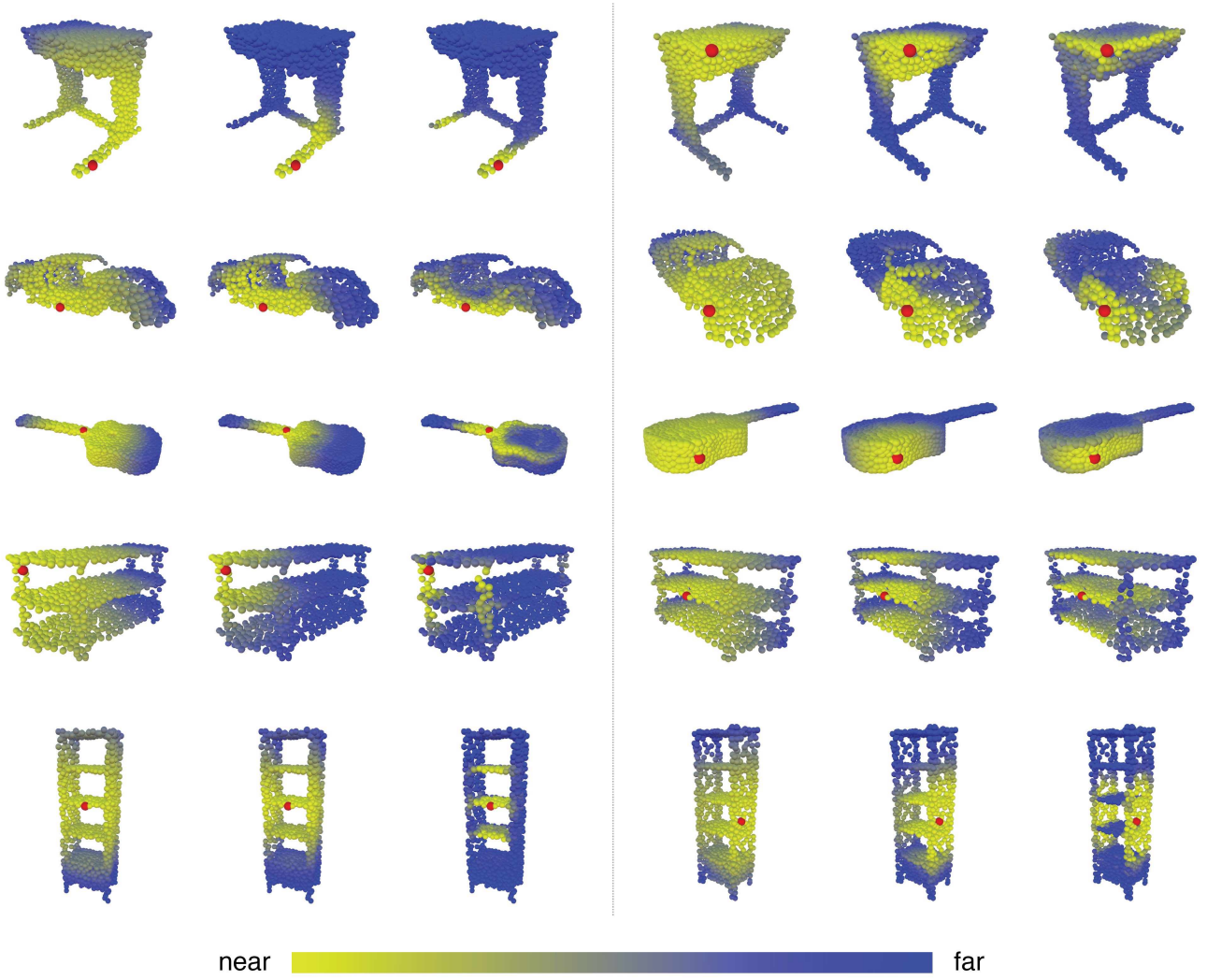


Figure 4. **Structure of the feature spaces** produced at different stages of our shape classification neural network architecture, visualized as the distance between the red point to the rest of the points. For each set, **Left:** Euclidean distance in the input $\mathbb{R}^3$ space; **Middle:** Distance 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.

$\mathbb{R}^{F_l}$ the output of the l-th layer; $X^{(0)} = X$ is the input point cloud.

Dynamic graph update Our experiments suggests that it is possible and actually beneficial to *recompute the graph* using nearest neighbors in the features space produces 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)*. At each layer we thus have a different graph $\mathcal{G}^{(l)} = (\mathcal{V}^{(l)}, \mathcal{E}^{(l)})$, where the l-th layer edges are of the form $(i, j_{i1}), \ldots, (i, j_{ik_l})$ such that $x^{(l)}_{j_{i1}}, \ldots, x^{(l)}_{j_{ik_l}}$ are the k_l points closest to $x^{(l)}_i$. The F_{l+1}-dimensional output of the $(l+1)$-st layer is produced by applying EdgeConv to the F_l-dimensional output of the l-th layer,

$$x^{(l+1)}_i = \square_{j:(i,j)\in\mathcal{E}^{(l)}} h^{(l)}_{\boldsymbol{\Theta}}(x^{(l)}_i, x^{(l)}_j), \tag{2}$$

where $h^{(l)} : \mathbb{R}^{F_l} \times \mathbb{R}^{F_l} \to \mathbb{R}^{F_{l+1}}$.

3.3. Implementation Details

We consider particular instances of Dynamic Graph CNNs for two prototypical tasks in point cloud analysis: classification and segmentation. The respective architectures, depicted in Figure 3 (top and bottom branches), have

a similar structure to PointNet. Both architectures share a spatial transformer component, computing a global shape transformation. The classification network includes two EdgeConv layers, followed by a pooling operation and three fully-connected layers producing classification output scores. The segmentation network uses a sequence of three EdgeConv layers, followed by three fully-connected layers producing, for each point, segmentation output scores. For each EdgeConv block, we use a shared edge function $h^{(l)}(x_i^{(l)}, x_j^{(l)}) = h(x_i^{(l)}, x_j^{(l)} - x_i^{(l)})$ across all layers; the function is implemented as a multi-layer perceptron (MLP) and $\square = \max$ aggregation operation.

In our classification architecture, the graph is constructed using $k = 20$ nearest neighbors, while in our segmentation architecture, $k = 30$.

4. Comparison to existing methods

Our DGCNN is related to two classes of approaches, PointNets and graph CNNs, which we show to be particular settings of our method.

PointNet is a special case of our method with $k = 1$, which results in a graph with an empty edge set $\mathcal{E} = \varnothing$. The edge function used in PointNet is $h(x_i, x_j) = h(x_i)$, which considers only the global geometry but discards the local one. The aggregation operation used in PointNet is $\square = \max$ (or $\sum$, because the aggregation function only works on a single node).

PointNet++ tries to account for local point cloud structure by applying PointNet in a local manner. In terms of our notation, 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, a certain number of points are selected by using farthest point sampling (FPS) algorithm. Only the selected points are preserved while others are directly discarded after this layer and in this way, the graph becomes smaller after the operation applied on each layer. Different from ours, PointNet++ computes pairwise distances using point input coordinates. The edge function used by PointNet++ is also $h(x_i, x_j) = h(x_i)$, and the aggregation operation is also a $\max$.

Among graph CNNs, MoNet [31], ECC [45], and Graph Attention Networks [52] are the most related approaches. The common denominator of these methods is the notion of a local patch on a graph, in which a convolution-type operation can be defined.[1] Specifically, [31] 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 in these coordinates:

$$x_i' = \sum_{m=1}^{M} w_m \sum_{j:(i,j)\in\mathcal{E}} g_{\boldsymbol{\Theta}_m}(\mathbf{u}(x_i, x_j))x_j, \qquad (3)$$

where g denotes the Gaussian kernel, $\{\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_M\}$ encode the learnable parameters of the Gaussians (mean and covariance), and $\{w_1, \ldots, w_M\}$ are the learnable filter coefficients. We can easily observe that (3) is an instance of our more general EdgeConv operation (1), with a particular choice of the edge function

$$h_{w_1,\boldsymbol{\Theta}_1,\ldots,w_M,\boldsymbol{\Theta}_M}(x_i, x_j) = \sum_{m=1}^{M} w_m g_{\boldsymbol{\Theta}_m}(\mathbf{u}(x_i, x_j))x_j$$

and summation as the aggregation operation.

A crucial difference between EdgeConv and MoNet and other graph CNN methods is that the latter assume a given fixed graph on which the convolution-like operations are applied, while we *dynamically update the graph* for each layer output. 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.

5. 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.

5.1. Classification

Data We evaluate our model on the ModelNet40 [54] classification task, consisting in predicting the category of a previously unseen shape. The dataset contains 12,311 meshed CAD models from 40 categories. 9,843 models are used for training and 2,468 models are for testing. We follow verbatim the experimental settings of [34]. For each model, 1,024 points are uniformly sampled from the mesh faces and normalized to 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 as in [36] by randomly rotating and scaling objects and perturbing the object and point locations.

Architecture The network architecture used for the classification task is shown in Figure 3 (top branch). We use a local-aware spatial transformer network to align the point cloud. It has two shared fully-connected layers $(64, 128)$[2] to

[1] The methods of [45] and [52] can be considered as instances of [31], with the difference that the weights are constructed employing features from the adjacent nodes instead of the graph structure.

[2] Here, we use $(64, 128)$ to denote that the two fully-connected layers have 64 filters and 128 filters, respectively; we use the same notation for the remainder of our discussion.

construct one EdgeConv layer and after which, one shared fully-connected layer (1024) is used to transform the point-wise features to higher-dimensional space. After the global max pooling, two fully-connected layers $(512, 256)$ are used to compute the transformation matrix.

We use two EdgeConv layers to extract geometric features. The first EdgeConv layer uses three shared fully-connected layers $(64, 64, 64)$ while the second EdgeConv layer uses a shared fully-connected layer (128). Shortcut connections are included to extract multi-scale features and one shared fully-connected layer (1024) to aggregate multi-scale features. The number k of nearest neighbors is 20. Then, a global max pooling is used to get the point cloud global feature, after which two multi-layer perceptrons $(512, 256)$ 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 ReLU and batch normalization.

Training We use the same training strategy as [34]. We use Adam [18] with learning rate 0.001 that is divided by 2 every 20 epochs. The decay rate for batch normalization is initially 0.5 and 0.99 finally. The batch size is 32 and the momentum is 0.9.

Results Table 1 shows the results of the classification task. Our model achieves the best results on this dataset. Our baseline without transformer network and using fixed graph is 0.5% better than PointNet++. An advanced version including a local-aware network and dynamical graph recomputation achieves best results on this dataset.

	MEAN CLASS ACCURACY	OVERALL ACCURACY
3DSHAPENETS [54]	77.3	84.7
VOXNET [30]	83.0	85.9
SUBVOLUME [35]	86.0	89.2
ECC [45]	83.2	87.4
POINTNET [34]	86.0	89.2
POINTNET++ [36]	-	90.7
KD-NET (DEPTH 10) [20]	-	90.6
KD-NET (DEPTH 15) [20]	-	91.8
OURS (BASELINE)	88.8	91.2
OURS	**90.2**	**92.2**

Table 1. Classification results on ModelNet40.

5.2. Model Complexity

We use the ModelNet40 [54] classification experiment to compare the complexity of our model to previous state-of-the-art. Table 2 shows that our model achieve the best tradeoff between the model complexity (number of parameters), computational complexity (measured as forward pass time), and achieved classification accuracy.

Our baseline model outperforms the previous state-of-the-art PointNet++ by 0.5% accuracy, at the same time being 5 times faster compared to PointNet++. Our baseline model does not use a spatial transformer and uses the fixed k-NN graph. A more advanced version of our model including a spatial transformer block and dynamically graph computation outperforms PointNet++ by 1.5% while having comparable number of parameters and computational complexity.

	MODEL SIZE(MB)	FORWARD TIME(MS)	ACCURACY(%)
POINTNET (BASELINE)	9.4	11.6	87.1
POINTNET	40	25.3	89.2
POINTNET++	12	163.2	90.7
OURS (BASELINE)	11	29.7	91.2
OURS	21	94.6	92.2

Table 2. Complexity, forward time and accuracy of different models

5.3. More Experiments on ModelNet40

We also experiment with various settings of our model on the ModelNet40 [54] dataset. In particular, we analyze the effectiveness of local-aware transformer network, different distance metrics, and explicit usage of $x_i - x_j$.

Table 3 shows the results. "Centralization" denotes using concatenation of x_i and $x_i - x_j$ as the edge features rather than concatenating x_i and x_j. "Spatial Transformer" denotes the local-aware transformer network while "Dynamic graph recomputation" denotes we reconstruct the graph rather than using a fixed graph. By dynamically updating graph, there is about $0.2\%\sim0.3\%$ improvement, and Figure 4 also verifies our hypothesis that the model can extract semantics. In the later layers, certain patterns occur for recognition tasks. Explicitly centralizing each patch by using the concatenation of x_i and $x_i - x_j$ makes the operator more robust to translation, leading to about $0.2\%\sim0.3\%$ improvement for overall accuracy. The local-aware transformer makes the model invariant to rigid transformation and leads to approximately 0.7% improvement.

We also experiment with different numbers k of nearest neighbors as shown in Table 4. While we do not exhaustively experiment with all possible k, we find with large k that the performance degenerates. This confirms our hypothesis that with large k the Euclidean distance fails to approximate geodesic distance, destroying the geometry of each patch.

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

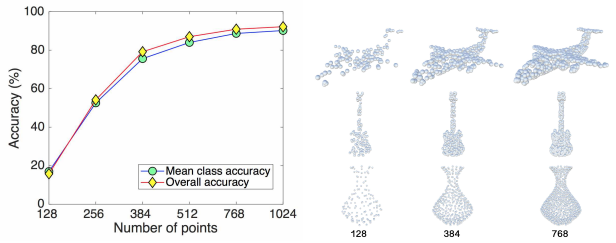


Figure 5. **Left:** Results of our model tested with random input dropout. The model is trained with number of points being 1024 and k being 20. **Right:** Point clouds with different number of points. The numbers of points are shown below the bottom row.

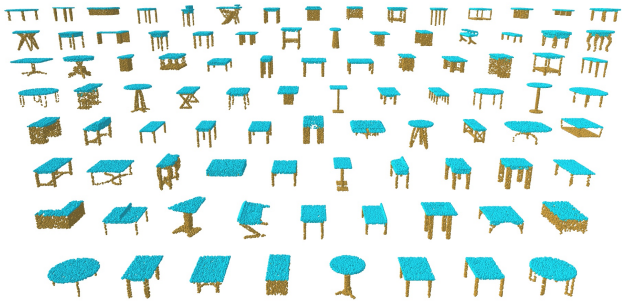


Figure 6. Our part segmentation testing results for tables.

dramatically.

CENT	DYN	XFORM	MEAN CLASS ACCURACY(%)	OVERALL ACCURACY(%)
x			88.8	91.2
x	x		88.8	91.5
x		x	89.6	91.9
	x	x	89.8	91.9
x	x	x	90.2	92.2

Table 3. Effectiveness of different components. CENT denotes centralization, DYN denotes dynamical graph recomputation, and XFORM denotes the use of a spatial transformer.

NUMBER OF NEAREST NEIGHBORS (K)	MEAN CLASS ACCURACY(%)	OVERALL ACCURACY(%)
5	88.0	90.5
10	88.8	91.4
20	90.2	92.2
40	89.2	91.7

Table 4. Results of our model with different numbers of nearest neighbors.

5.4. Part Segmentation

Data We extend our EdgeConv model architectures for part segmentation task on ShapeNet part dataset [55]. For this task, each point from a point cloud set is classified into one of a few predefined part category labels. The dataset

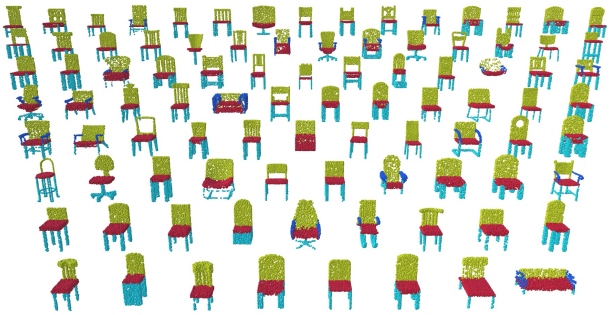


Figure 7. Our part segmentation testing results for chairs.

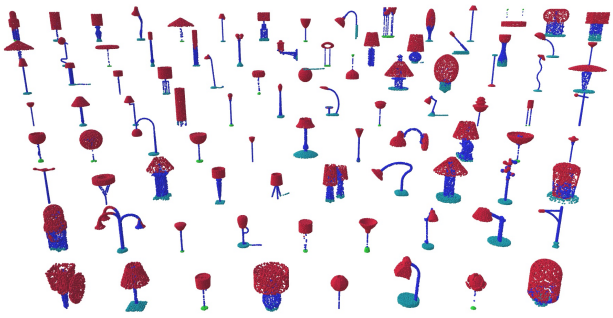
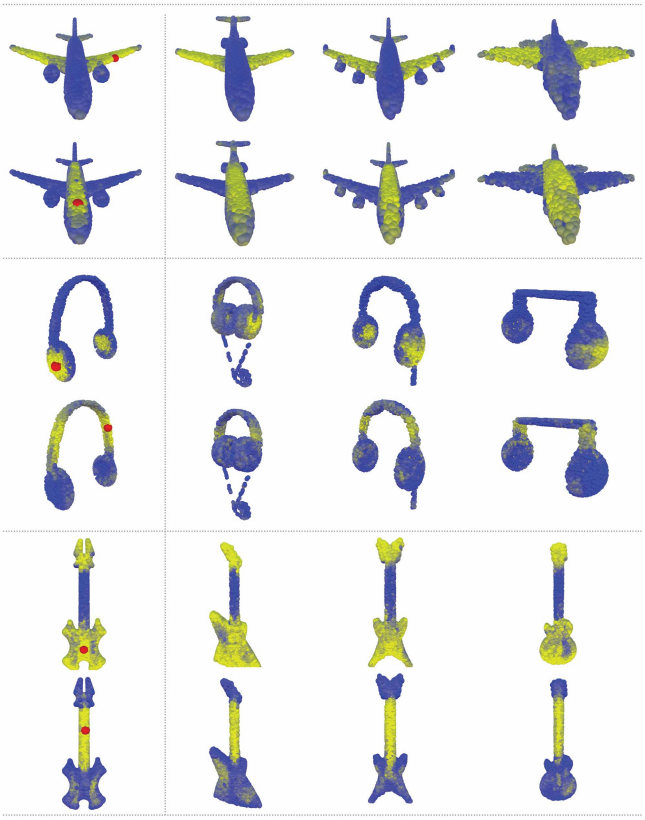


Figure 8. Our part segmentation testing results for lamps.

contains 16,881 3D shapes from 16 object categories, annotated with 50 parts in total. 2,048 points are sampled 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 in [10] in our experiment.

Architecture The network architecture is illustrated in Figure 3 (bottom branch). The same spatial transformer network is used for segmentation task. Nine shared fully-connected layers $(64, 64, 64, 64, 64, 64, 64, 64, 64)$ are used to construct three EdgeConv layers; each EdgeConv layer has three fully-connected layers $(64, 64, 64)$. A shared fully-connected layer (1024) is used to aggregate 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. Batch-norm, 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, except k is changed from 20 to 30 due to the increase of point density. A distributed training scheme is further implemented on two NVIDIA TITAN X GPUs to maintain the training batch size.

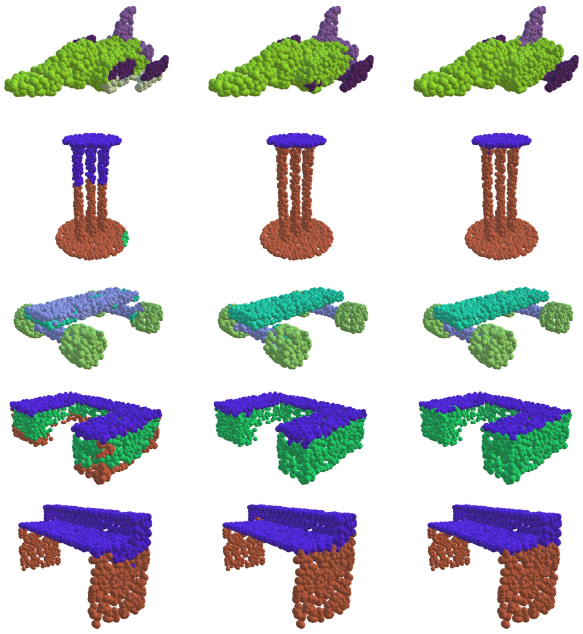


Source points Other point clouds from the same category

Figure 9. 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.

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; 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 [34], PointNet++ [36], Kd-Net [20], and LocalFeatureNet [43]. The evaluation results are shown in Table 5. We also visually compare the results of our model and PointNet in Figure 10.

Intra-cloud distances We next explore the relationships between different point clouds captured using our features. As shown in Figure 9, we take one red point from a source point cloud and compute its distance in feature space to



PointNet Ours Ground truth

Figure 10. Compare part segmentation results. For each set, from left to right: PointNet, ours and ground truth.

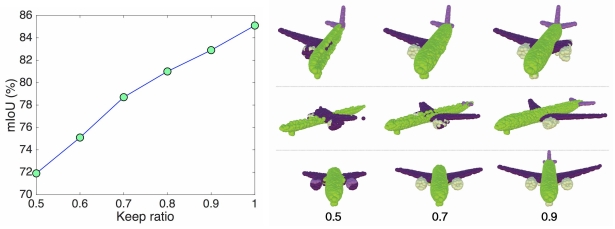


Figure 11. **Left:** The mean IoU (%) improves when the ratio of kept points increases. Points are dropped from one of six sides (top, bottom, left, right, front and back) randomly during evaluation process. **Right:** Part segmentation results on partial data. Points on each row are dropped from the 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.

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 11. On the left, the mean IoU versus "keep ratio" is shown. On the right, the results for an airplane model are visualized.

	MEAN	AREO	BAG	CAP	CAR	CHAIR	EAR PHONE	GUITAR	KNIFE	LAMP	LAPTOP	MOTOR	MUG	PISTOL	ROCKET	SKATE BOARD	TABLE	WINNING CATEGORIES
# 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	1
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	5
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	0
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**	5
OURS	**85.1**	84.2	**83.7**	84.4	77.1	**90.9**	**78.5**	**91.5**	**87.3**	82.9	96.0	67.8	93.3	82.6	**59.7**	75.5	82.0	**6**

Table 5. Part segmentation results on ShapeNet part dataset. Metric is mIoU(%) on points.

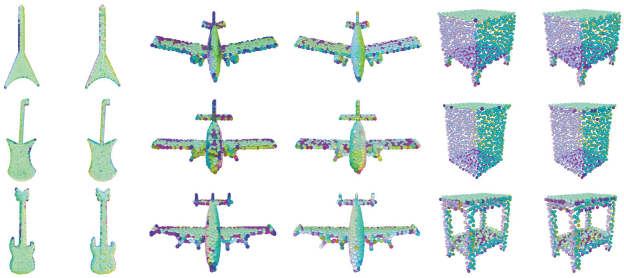


Figure 12. Surface normal estimation results. The colors shown in the figure are RGB-coded surface normals, meaning XYZ components of surface normal vectors are put into RGB color channels. For each pair: our prediction (left) and ground truth (right).

5.5. Indoor Scene Segmentation

Data We evaluate our model on Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [1] 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 in [34], 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). 4,096 points are sampled for each block during training process, and all points are used for testing. We also use the same 6-fold cross validation over the 6 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. We compare our model with both PointNet [34] 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 [12], who 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 6, and visually compare the results of PointNet and our model in Figure 13.

	MEAN IOU	OVERALL ACCURACY
POINTNET (BASELINE) [34]	20.1	53.2
POINTNET [34]	47.6	78.5
MS + CU(2) [12]	47.8	79.2
G + RCU [12]	49.7	81.1
OURS	**56.1**	**84.1**

Table 6. 3D semantic segmentation results on S3DIS. MS+CU for multi-scale block features with consolidation units; G+RCU for the grid-blocks with recurrent consolidation Units.

5.6. Surface Normal Prediction

We can also adapt our segmentation model to predict surface normals from point clouds.

Data We still use the ModelNet40 dataset. The surface normals are sampled directly from CAD models. The normal of one point is represented by (n_x, n_y, n_z). We use 9,843 models for training and 2,468 models for testing.

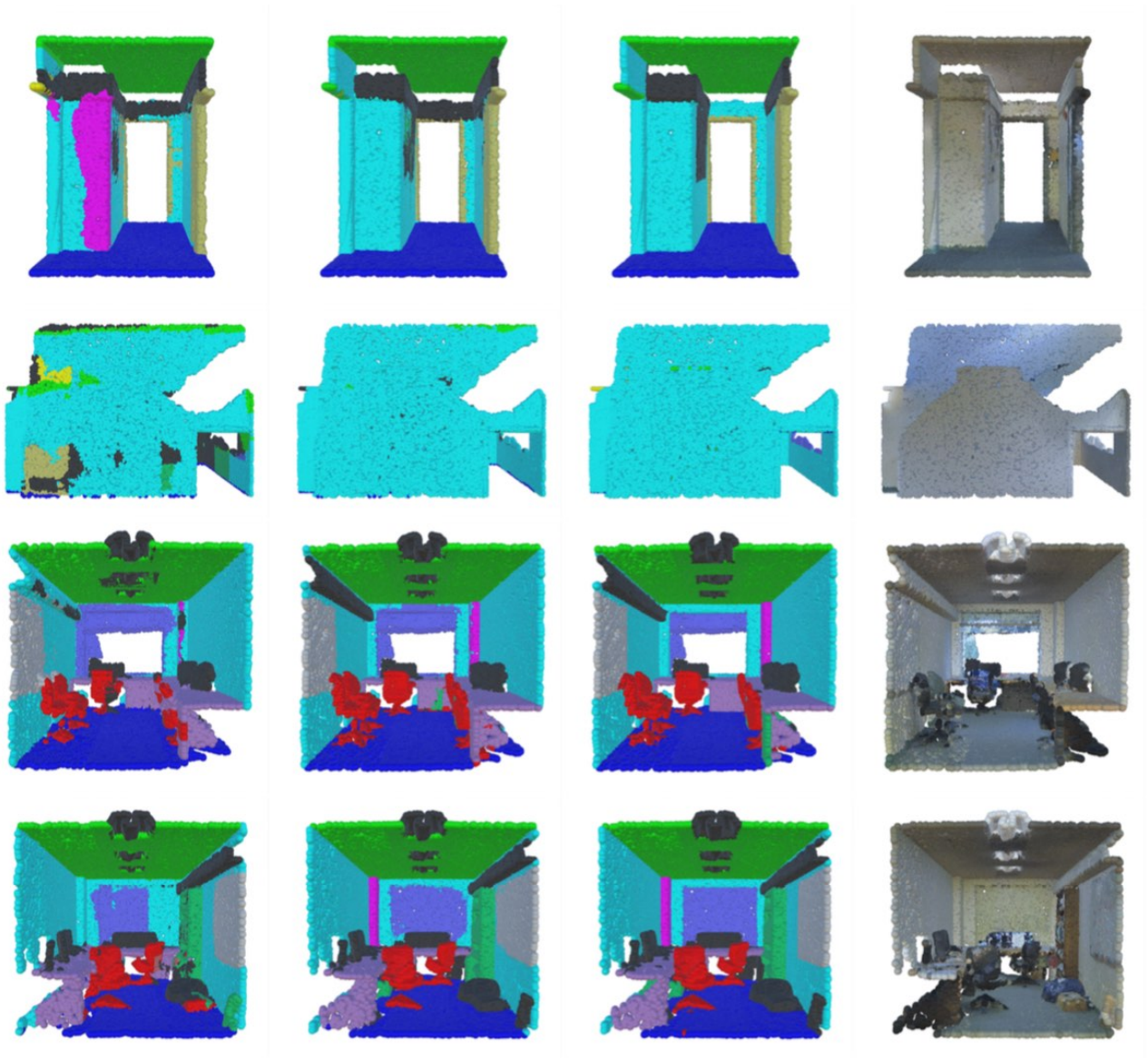
Architecture We change the last layer of our segmentation model to output 3 continuous values; mean squared error (MSE) is used as the training loss.

Results Qualitative results are shown in Figure 12, compared with ground truth. Our model faithfully captures orientation even in the presence of fairly sharp features.

6. Discussion

In this work we propose a new operator for learning on point cloud and show its performance on various tasks. The success of our technique verifies our hypothesis that local geometric features are crucial to 3D recognition tasks, even after introducing machinery from deep learning. Furthermore, we show our model can be easily modified for various tasks like normal prediction while continuing to achieve reasonable results.

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



PointNet Ours Ground truth Real color

Figure 13. 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.

avenues for future research and extension. Primarily, the success of our model suggests that intrinsic features can be equally valuable if not more than simply 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. Another possible extension is to design a non-shared transformer network that works on each local patch differently, adding flexibility to our model. Finally, we will consider applications of our techniques to more abstract point clouds coming from applications like document retrieval 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

[1] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proc. CVPR*, 2016.

[2] M. Aubry, U. Schlickewei, and D. Cremers. The wave kernel signature: A quantum mechanical approach to

shape analysis. In *Proc. ICCV Workshops*, 2011.

[3] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Proc. NIPS*, 2001.

[4] S. Biasotti, A. Cerri, A. Bronstein, and M. Bronstein. Recent trends, applications, and perspectives in 3d shape similarity assessment. *Computer Graphics Forum*, 35(6):87–119, 2016.

[5] D. Boscaini, J. Masci, S. Melzi, M. M. Bronstein, U. Castellani, and P. Vandergheynst. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. *Computer Graphics Forum*, 34(5):13–23, 2015.

[6] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Proc. NIPS*, 2016.

[7] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[8] M. M. Bronstein and I. Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Proc. CVPR*, 2010.

[9] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203*, 2013.

[10] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv:1512.03012*, 2015.

[11] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. NIPS*, 2016.

[12] F. Engelmann, T. Kontogianni, A. Hermans, and B. Leibe. Exploring spatial context for 3d semantic segmentation of point clouds. In *Proc. CVPR*, 2017.

[13] D. Ezuz, J. Solomon, V. G. Kim, and M. Ben-Chen. Gwcnn: A metric alignment layer for deep shape analysis. *Computer Graphics Forum*, 36(5):49–57, 2017.

[14] A. Golovinskiy, V. G. Kim, and T. Funkhouser. Shape-based recognition of 3d point clouds in urban environments. In *Proc. ICCV*, 2009.

[15] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, and J. Wan. 3d object recognition in cluttered scenes with local surface features: a survey. *Trans. PAMI*, 36(11):2270–2287, 2014.

[16] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Proc. NIPS*, 2015.

[17] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *Trans. PAMI*, 21(5):433–449, 1999.

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 2015.

[19] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. 2017.

[20] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. 2017.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, 2012.

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[23] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv:1705.07664*, 2017.

[24] H. Ling and D. W. Jacobs. Shape classification using the inner-distance. *Trans. PAMI*, 29(2):286–299, 2007.

[25] O. Litany, T. Remez, E. Rodolà, A. M. Bronstein, and M. M. Bronstein. Deep functional maps: Structured prediction for dense shape correspondence. In *Proc. ICCV*, 2017.

[26] M. Lu, Y. Guo, J. Zhang, Y. Ma, and Y. Lei. Recognizing objects in 3d point clouds with multi-scale local features. *Sensors*, 14(12):24156–24173, 2014.

[27] S. Manay, D. Cremers, B.-W. Hong, A. J. Yezzi, and S. Soatto. Integral invariants for shape matching. *Trans. PAMI*, 28(10):1602–1618, 2006.

[28] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman. Convolutional neural networks on surfaces via seamless toric covers. In *Proc. SIGGRAPH*, 2017.

[29] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proc. 3dRR*, 2015.

[30] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Proc. IROS*, 2015.

[31] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, 2017.

[32] M. Ovsjanikov, M. Ben-Chen, J. Solomon, A. Butscher, and L. Guibas. Functional maps: a flexible representation of maps between shapes. *TOG*, 31(4):30, 2012.

[33] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. *arXiv:1711.08488*, 2017.

[34] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proc. CVPR*, 2017.

[35] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proc. CVPR*, 2016.

[36] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. NIPS*, 2017.

[37] R. M. Rustamov. Laplace-beltrami eigenfunctions for deformation invariant shape representation. In *Proc. SGP*, 2007.

[38] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Proc. ICRA*, 2009.

[39] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *Proc. IROS*, 2008.

[40] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3D Point Cloud Based Object Maps for Household Environments. *Robotics and Autonomous Systems Journal*, 56(11):927–941, 30 November 2008.

[41] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Tran. Neural Networks*, 20(1):61–80, 2009.

[42] S. A. A. Shah, M. Bennamoun, F. Boussaid, and A. A. El-Sallam. 3d-div: A novel local surface descriptor for feature matching and pairwise range image registration. In *Proc. ICIP*, 2013.

[43] Y. Shen, C. Feng, Y. Yang, and D. Tian. Neighbors do help: Deeply exploiting local structures of point clouds. *arXiv:1712.06760*, 2017.

[44] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[45] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017.

[46] A. Sinha, J. Bai, and K. Ramani. Deep learning 3d shape surfaces using geometry images. In *Proc. ECCV*, 2016.

[47] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. CVPR*, 2015.

[48] J. Sun, M. Ovsjanikov, and L. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum*, 28(5):1383–1392, 2009.

[49] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proc. ICCV*, 2017.

[50] F. Tombari, S. Salti, and L. Di Stefano. A combined texture-shape descriptor for enhanced 3d feature matching. In *Proc. ICIP*, 2011.

[51] O. Van Kaick, H. Zhang, G. Hamarneh, and D. Cohen-Or. A survey on shape correspondence. *Computer Graphics Forum*, 30(6):1681–1707, 2011.

[52] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. *arXiv:1710.10903*, 2017.

[53] L. Wei, Q. Huang, D. Ceylan, E. Vouga, and H. Li. Dense human body correspondences using convolutional networks. In *Proc. CVPR*, 2016.

[54] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proc. CVPR*, 2015.

[55] L. Yi, V. G. Kim, D. Ceylan, I. Shen, M. Yan, H. Su, A. Lu, Q. Huang, A. Sheffer, L. Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *TOG*, 35(6):210, 2016.

[56] L. Yi, H. Su, X. Guo, and L. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proc. CVPR*, 2017.

[57] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Proc. ICRA*, 2017.