Spatial Transformer for 3D Point Clouds

Jiayun Wang Rudrasis Chakraborty Stella X. Yu

Abstract—Deep neural networks are widely used for understanding 3D point clouds. At each *point convolution layer*, features are computed from local neighbourhoods of 3D points and combined for subsequent processing in order to extract semantic information. Existing methods adopt the same individual point neighborhoods throughout the network layers, defined by the same metric on the fixed input point coordinates. This common practice is easy to implement but not necessarily optimal. Ideally, local neighborhoods should be different at different layers, as more latent information is extracted at deeper layers. We propose a novel end-to-end approach to learn different non-rigid transformations of the input point cloud so that optimal local neighborhoods can be adopted at each layer. We propose both linear (affine) and non-linear (projective and deformable) spatial transformers for 3D point clouds. With spatial transformers on the ShapeNet part segmentation dataset, the network achieves higher accuracy for all categories, with 8% gain on earphones and rockets in particular. Our method also outperforms the state-of-the-art on other point cloud tasks such as classification, detection, and semantic segmentation. Visualizations show that spatial transformers can learn features more efficiently by dynamically altering local neighborhoods according to the geometry and semantics of 3D shapes in spite of their within-category variations.

Index Terms—point cloud, transformation, deformable, segmentation, 3D detection

1 Introduction

3D computer vision has been on the rise with more advanced 3D sensors and computational algorithms. Depth cameras and LiDAR sensors output 3D point clouds, which become key components in several 3D computer vision tasks including but not limited to virtual / augmented reality [1], [2], 3D scene understanding [3], [4], [5], and autonomous driving [6], [7], [8].

On the algorithmic side, convolutional neural networks (CNNs) have achieved great success in many computer vision tasks [9], [10]. However, the concept of convolution cannot be directly applied to a point cloud, as 3D points are not pixels on a regular grid with regular neighbourhoods. One line of approaches is to convert the 3D point cloud into a representation where CNNs are readily applicable, e.g., a regular voxel representation [11], [12], [13] or 2D view projections [14], [15], [16], [17].

Another line of approaches is to develop network architectures that can directly process point clouds [18], [19], [20], [21]. Analogous to convolution on 2D pixels, convolution on 3D points needs to first identify *local neighborhoods* around individual input points. This step is achieved by computing the so-called *point affinity matrix*, i.e., the adjacency matrix of a dense graph constructed from the point cloud. These neighborhoods are then used for extracting features with point-wise convolutions. By stacking basic point convolution layers, a neural network can extract information from the point cloud with an increasing level of abstraction.

However, unlike images where 2D pixels are laid out on a regular grid with simple and well-defined local neighborhoods, local neighborhoods of 3D points are ill-defined and subject to various geometric transformations of 3D shapes. Most methods [18], [19], [22], [23] define local neighborhoods as nearest neighbors in the Euclidean space of the input 3D point coordinates.

This common practice of defining a nearest neighbor graph according to the Euclidean distances on the fixed input 3D point coordinates may be simple but not optimal. First, such distances may not be able to efficiently encode semantics of 3D shapes, e.g., semantically or topologically far points might be spatially close in terms of the Euclidean distances. Secondly, fixed neighborhoods throughout the network may reduce the model's learning capacity as different layers capture information at different levels of abstraction, e.g., objects have a natural hierarchy and in order to segment out their parts, it would be more efficient to provide different layers the ability to parse them at different spatial scales.

We propose to address these fixed point neighbourhood restrictions by dynamically learning local neighborhoods and transforming the input point cloud at different layers. We use a parametric model that takes both point coordinates and learned features as inputs to learn the point affinity matrix. At different layers of the network, we learn several different transformations (dubbed as *spatial transformers* or *transformers* hereafter, Fig.1) and corresponding point local neighborhoods (Fig.2). Spatial transformers allow the network to adaptively learn point features covering different spatial extensions at each depth layer.

To spatially transform a point cloud, we learn a function, Φ, that generates transformed point coordinates from the input point coordinates and feature maps. However, it is nontrivial to learn Φ without smoothness constraints. Since any isometric (e.g. rigid) transformation cannot change the distance metric, we consider non-rigid transformations, both linear and non-linear families. That is, our *spatial transformers* are parameterized functions conditioned on the input point coordinates P and feature map F; they are subsequently used to transform the point coordinates, resulting in a new point affinity matrix for obtaining dynamic local neighborhoods.

We consider three families of spatial transformers.

- *The authors are with UC Berkeley / ICSI, 2150 Shattuck Ave, Berkeley, CA, 94704. E-mail: {peterwg, rudra, stellayu}@berkeley.edu. Corresponding author: Stella X. Yu.*
- *Our code us publicly available at http://pwang.pw/spn.html.*

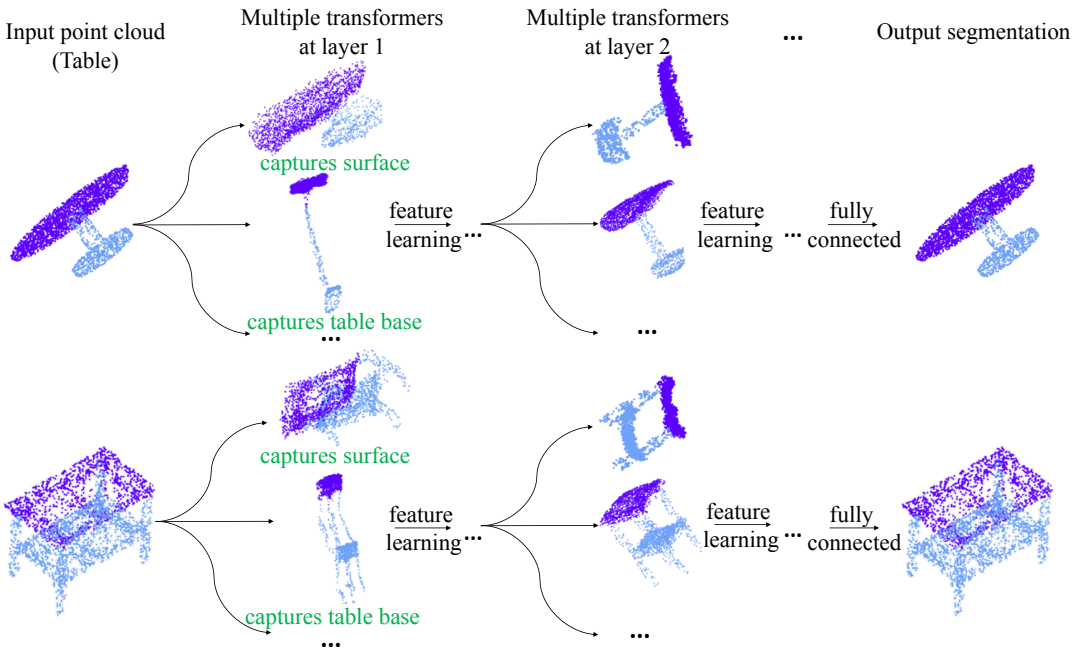

Fig. 1. Spatial transformers learn several *global transformations* at each layer to obtain different local point neighborhoods. We show transformed point clouds at different layers learned by spatial transformers for different instances of a category (e.g. tables). Compared with previous works adopting fixed local neighborhoods, dynamic point neighborhoods make the network more powerful in learning semantics from point clouds. For example, corresponding geometric transformations capture similar semantic information even high intra-class spatial variations exist. The second transformation at layer 1 deforms different tables to be more semantically similar, and makes parsing the part of table base easier. Furthermore, the proposed transformer is a stand-alone module and can be easily added to existing point cloud processing networks.

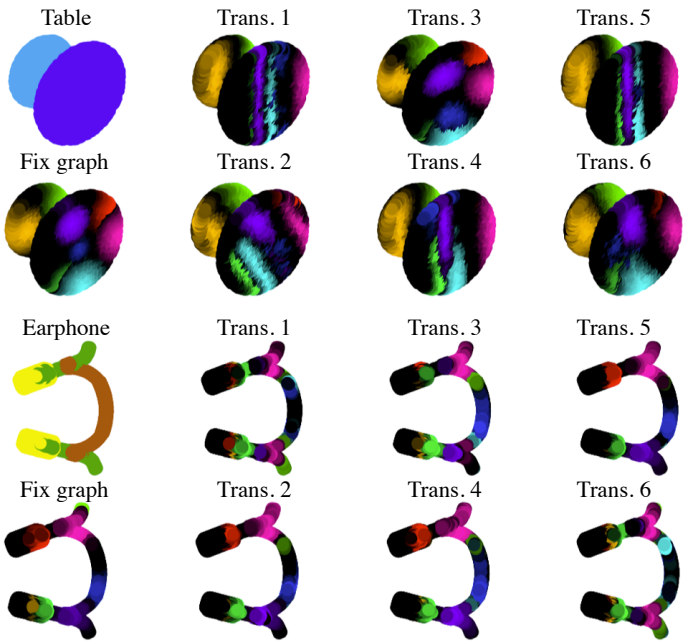

Fig. 2. At each layer, we apply multiple spatial transformers to deform the input point cloud for learning different neighborhoods. We show local neighborhoods of input point cloud examples with and without deformable transformers. Different colors indicate different neighborhoods, and intensities indicate distances to the central point. The dynamic neighborhood enhance the network capacity to learn from objects with large spatial variations. Rotating table and earphone for better visualizations.

1) Affine transformation $P \mapsto AP$, where A is an affine matrix. **2)** Projective transformer $\tilde{P} \mapsto B\tilde{P}$, where $\tilde{P}$ is 3D points expressed in homogeneous coordinates. **3)** Deformable transformer $P \mapsto CP + DF$, where C, D are respective transformation matrices of point coordinates and features F. The transformation depends on both the input point coordinates and the features the points assume.

Our work makes the following contributions.

- We propose to learn linear (affine) and non-linear (projective, deformable) spatial transformers for new point affinity matrices and thus dynamic local neighborhoods throughout the neural network.
- We demonstrate that our spatial transformers can be easily added to existing point cloud networks for a variety of tasks: classification, detection, and segmentation.
- We apply spatial transformers to various point cloud processing networks, with point-based and sampling-based metrics for point neighborhoods, and observe performance gains of dynamic graphs over fixed graphs.

2 Related Work

We discuss related works that motivate the necessity of our proposed spatial transformers.

View-based methods project 3D shapes to 2D planes and use images from multiple views as representations. Taking advantages of the power of CNNs in image processing [24], [25], [15], [14], view-based methods achieve reasonable 3D processing performance. However, certain information about 3D structures gets lost when 3D points are projected to 2D image planes; occluded surfaces and density variations are thus often troublesome for these methods.

Voxel-based methods represent 3D shapes as volumetric data on a regular 3D grid, and proceed with 3D convolution [11], [26], [27]. Their caveates are quantization artifacts,

inefficient usage of 3D voxels, and low spatial resolutions due to a large memory requirement. In addition, 3D convolutions are not biased towards surface property extraction and thus cannot capture geometrical and semantic information efficiently. Recent works that apply different partition strategies [28], [12], [27], [13] relieve these issues but depend heavily on bounding volume subdivision instead of local geometric shapes. In contract, our method works directly on the 3D point cloud, minimizing geometric information loss and maximizing processing efficiency.

Point cloud processing methods take a point cloud as the input and extract semantic information by point convolutions. PointNet [18] directly learns the embedding of every 3D point in isolation and gather that information by pooling point features later on. Although it achieves good performance at the time, PointNet does not learn any 3D local shape information since each local neighborhood contains only one point. PointNet++ [19] addresses this caveate by adopting a hierarchical application of isolated 3D point feature learning to multiple subsets of a point cloud. Many other works also explore different strategies for leveraging local structure learning from point clouds [22], [23]. Instead of finding neighbors of each point, SplatNet [29] encodes local structures from the sampling perspective: it groups points based on permutohedral lattices [30], and then applies bilateral convolution [31] for feature learning. Super-point graphs [32] partition a point cloud into super-points and learn the 3D point geometric organization. Many works focus on designing novel point convolutions given 3D point local neighborhoods [19], [29], [23], ignoring how the local neighborhoods should be formed.

Unlike pixels in a 2D image, points in a 3D point cloud are un-ordered, with irregular and heterogeneous neighborhoods; regular convolution operations thus cannot be applied. Many works [22], [23], [33], [34], [35], [36] aim to design point convolution operations that resemble regular 2D convolutions. Fixed input point coordinates are used to define local neighborhoods for the point convolution, resulting in the same local neighbourhoods at different layers that limit the model's processing power. In contrast, our work uses spatial transformers at each layer to learn dynamic local neighborhoods in a more adaptive, flexible, and efficient way.

Spatial Transformations. The idea of enabling spatial transformation in neural networks has been explored for 2D image understanding [37]. It is natural to extend the idea to 3D point clouds. PointNet [18] adopts a rigid transformation module on the input point cloud to factor out the object pose and improve classification accuracy. KPConv [38] applies *local deformation* in the neighborhood of point convolution to enhance its learning capacity. In contrast, our work learns several different global transformations to apply on the input point cloud at each layer for dynamic neighborhoods.

3 METHODS

We first briefly review different geometric transformation methods and their influence on the affinity matrix of point cloud data, then describe the design of our three *spatial transformers*, namely, **(a)** affine, **(b)** projective and **(c)** deformable. We apply the *spatial transformer block*, consisting of multiple spatial transformers, to each layer of a network for altering local neighborhoods for better point feature learning. We conclude the section by introducing how the transformers can be added to existing point cloud processing networks and the relevance to other works.

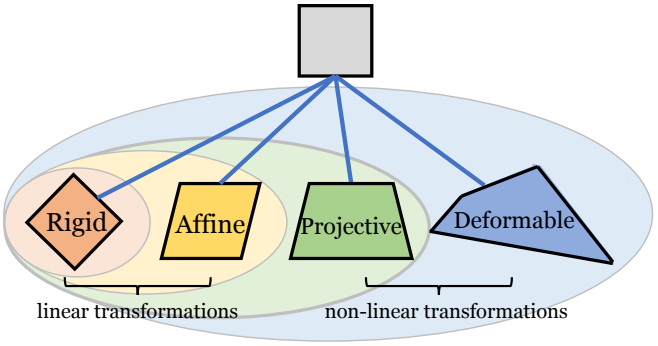

Fig. 3. Geometric transformations. We illustrate how a grey square transforms after rigid, affine, projective and deformable transformations.

3.1 Geometric Transformations

We propose to learn transformations on the input point cloud to *deform* its geometric shape, and alter local neighborhoods with new point affinity matrices. The hypothesis behind the usage of geometric transformation is as follows:

Hypothesis 1. *Let $P = \{\mathbf{p}_i\}$ be the input point cloud and let $\mathcal{N}_i$ be the local neighborhood around $\mathbf{p}_i \in \mathbf{R}^3$ from which we extract local features. Let $\mathcal{N} = \{\mathcal{N}_i\}$ be the set of local neighborhoods. Assume $\widetilde{\mathcal{N}} = \left\{\widetilde{\mathcal{N}}_i\right\}$ be the optimal neighborhood for learning local features, then $\exists$(smooth) $\Phi : \mathcal{N}_i \to \widetilde{\mathcal{N}}_i$ for all $\mathbf{p}_i$.*

Essentially we are going to use different types of geometric transformations to approximate Φ. The new learned affinity matrix will dynamically alter local neighborhoods to allow better feature learning.

Illustrated in Fig.3, transformations can be categorized into rigid and non-rigid transformations, and the latter can be further categorized into linear and non-linear transformations. We now discuss different spatial transformations.

Rigid Transformations. The group of rigid transformations consist of translations and rotations. However, rigid transformations are isometric (in ℓ_2 distance) and therefore preserves the point affinity matrix. Thus, local neighborhoods are *invariant* to rigid transformations in terms of k-NN graphs. Hence, we do not consider rigid transformations.

Affine Transformations. Affine transformations belong to non-rigid linear transformations. Consider a 3D point cloud $P = \{\mathbf{p}_i\}_{i=1}^N \subset \mathbf{R}^3$ consisting of N three-dimensional vectors $\mathbf{p}_i \in \mathbf{R}^3$. Then, an affine transformation can be parameterized by an invertible matrix $A \in \mathbf{R}^{3\times3}$ and a translation vector $\mathbf{b} \in \mathbf{R}^3$. Given $A, \mathbf{b}$, the affine transformed coordinates $\mathbf{p}_i$ can be written as $\mathbf{p}_i \mapsto A\mathbf{p}_i + \mathbf{b}$. Note that translation $\mathbf{b}$ does not change the point affinity matrix and point neighborhoods. Recall that an affine transformation preserves collinearity, parallelism, and convexity.

Projective Transformations. Projective transformations are non-rigid non-linear transformations. We first map the 3D point cloud P to the homogeneous space and get $\widetilde{P}$, by appending one-vectors to the last dimension. The projective transformation is parameterized by $A \in \mathbf{R}^{4\times4}$ and the

transformed point $\widetilde{\mathbf{p}}_i \mapsto A\widetilde{\mathbf{p}}_i$. Compared to the affine transformations, projective transformations have more degrees of freedom but cannot preserve parallelism. Projective transformations preserve collinearity and incidence, hence fail to capture all possible deformations. For example, points lying on the same line will always be mapped to a line, and this constraint may be overly restrictive. It is of interest to be able to break this constraint if these points are from different semantic categories. A more general transformation that covers various deformations may be more effective.

Deformable Transformations. When all the points have the freedom to move without much constraint, the 3D shape can deform freely. We refer to this general spatial transformation as a *deformable transformation*. It has more degrees of freedom and does not necessarily preserve the topology.

Learning new per-point offsets would be computationally hard and costly, we thus use a parametric offset model instead. Taking both point coordinates and features as inputs, the model would learn offsets dependent upon both spatial and feature representations of the input point cloud.

3.2 Spatial Transformers for 3D Point Clouds

Our so-called *spatial transformer* method applies a geometric transformation to the input point cloud to obtain different local neighborhoods for feature learning. It can be applied to existing point cloud processing networks as spatial transformers only alter local neighborhoods.

Suppose at layer t, the spatial transformer block contains $k^{(t)}$ transformers. Each transformer learns a transformation to apply to the input point coordinates. We refer to the transformed points as nodes of a *sub-graph* and their feature on it the corresponding sub-feature. We then concatenate all sub-features from these transformers to form the final output of the learning block. Suppose that the i^{th} spatial transformer at the t^{th} layer takes as input the original point cloud $P \in \mathbf{R}^{3\times N}$ and previous feature map $\mathcal{F}^{(t-1)} \in \mathbf{R}^{\mathfrak{f}^{(t-1)}\times N}$.

Affine. We form $k^{(t)}$ new transformed point from $\mathbf{p}_j$ as:

$$\mathbf{g}_{i,j}^{(t)} = A_i^{(t)}\mathbf{p}_j + \mathbf{b}_i^{(t)}, \;\; i = 1, 2, ..., k^{(t)}. \tag{1}$$

Since the point affinity matrix is invariant under uniform scaling and translation, we set $\|A_i\|_F = 1, b = 0$, for all i. Thus, with $G_i^{(t)} = \left\{\mathbf{g}_{i,j}^{(t)}\right\}_j$, we simplify Equation 1 as:

$$G_i^{(t)} = A_i^{(t)}P, \;\; i = 1, 2, \cdots, k^{(t)}. \tag{2}$$

We compute the k nearest neighbours of each transformed point $G_i^{(t)}$ and obtain the point affinity matrix $S_i^{(t)}$, based on which we define local neighborhoods and apply point convolutions on previous point cloud feature map $\mathcal{F}^{(t-1)}$. We get the point cloud feature $F_i^{(t)} \in \mathbf{R}^{f_i^{(t)}\times N}$ of the sub-graph from i-th transformation and its altered neighborhoods:

$$F_i^{(t)} = \mathsf{CONV}(\mathcal{F}^{(t-1)}, S_i^{(t)}, k), \;\; i = 1, 2, ..., k^{(t)}, \tag{3}$$

where CONV denotes the point convolution: It takes **(a)** previous point cloud features, **(b)** the affinity matrix (for defining local neighborhoods of every point) and **(c)** the number of neighbors (for defining the size of neighborhoods) as inputs.

In point convolutions such as [22], the point affinity matrix changes the input feature in a non-differentiable way. Therefore, we append the transformed point cloud $P_i^{(t)}$ to the input feature for the sake of back-propagating the transformation matrix A. In sampling-based convolutions such as bilateral convolution [29], the point affinity matrix changes the input feature in a differentiable way; no additional operation is needed.

For all the $k^{(t)}$ sub-graph in layer/ block t, we learn $k^{(t)}$ point cloud features $F_i^{(t)}$. The output of this module is the concatenation of all the sub-graph point cloud features:

$$\mathcal{F}^{(t)} = \mathsf{CONCAT}(F_1^{(t)}, F_2^{(t)}, ..., F_{k^{(t)}}^{(t)}), \tag{4}$$

where $F_i^{(t)} \in \mathbf{R}^{f_i^{(t)}\times N}$ and $\mathfrak{f}^{(t)} = \sum_i^{k^{(t)}} f_i^{(t)}$, $\mathcal{F}^{(t)} \in \mathbf{R}^{\mathfrak{f}^{(t)}\times N}$. In our implementation, we randomly initialize A from the standard normal distribution $\mathcal{N}(0, 1)$. Before computing the coordinates of the transformed point cloud, we normalize A by its norm $\|A\|_F$, as the point affinity matrix is invariant under uniform scaling.

Projective. Analogous to the affine spatial transformer, for the i^{th} graph at t^{th} layer, we apply a projective transformation to the point cloud $\widetilde{P}$ in homogeneous coordinates and get the transformed point cloud as:

$$\widetilde{G}_i^{(t)} = B_i^{(t)}\widetilde{P}, \;\; i = 1, 2, \cdots, k^{(t)}, \tag{5}$$

where $B_i^{(t)} \in \mathbf{R}^{4\times 4}$ is the transformation matrix in homogeneous coordinates. We then follow the same procedure as in Equations 3 and 4 to get the output feature $\mathcal{F}^t$.

Deformable. Affine and projective transformations can transform the input point cloud, alter the point affinity matrix, and provide learnable local neighborhoods for point convolutions at different layers. However, they are limited as affine transformations are linear and projective transformations map lines to lines only. We define a non-linear deformable spatial transformer at the t^{th} layer and i^{th} sub-graph as

$$G_i^{(t)} = A_i^{(t)}P + D_i^{(t)}, \tag{6}$$

where $A_i^{(t)}P$ is the affine transformation component and $D_i^{(t)} \in \mathbf{R}^{3\times N}$ gives every point additional freedom to move, so the point cloud has the flexibility to deform its shape. Note that the translation vector $\mathbf{b}$ in Equation 1 is a special case of the deformation matrix $D_i^{(t)}$. In general, the deformation matrix $D_i^{(t)}$ can significantly change local neighborhoods.

The spatial transformer parameters are learned in an end-to-end fashion from both point cloud coordinates and features. Since affine transformation $A_i^{(t)}P$ is dependent on spatial locations, we let the deformation matrix $D_i^{(t)}$ depend on the features: $D_i^{(t)} = \mathcal{C}_i^{(t)}\mathcal{F}^{(t-1)}$, where $\mathcal{C}_i^{(t)} \in \mathbf{R}^{3\times\mathfrak{f}}$ transforms the previous layer feature $\mathcal{F}^{(t-1)} \in \mathbf{R}^{\mathfrak{f}\times N}$ from $\mathbf{R}^{\mathfrak{f}}$ to $\mathbf{R}^3$. Hence, the deformable transformation in Equation 6 can thus be simplified as:

$$G_i^{(t)} = \begin{bmatrix} A_i^{(t)} & \mathcal{C}_i^{(t)} \end{bmatrix} \begin{bmatrix} P \\ \mathcal{F}^{(t-1)} \end{bmatrix} = C_i^{(t)} \begin{bmatrix} P \\ \mathcal{F}^{(t-1)} \end{bmatrix}, \tag{7}$$

where $C_i^{(t)} \in \mathbf{R}^{3\times(3+f^{(t-1)})}$ is the concatenation of affine and deformable transformation matrix that captures both point cloud coordinates and features.

After we compute the transformed point coordinates $G^{(t)}$, we follow Equations 3 and 4 to learn the feature of each

transformed sub-graph and concatenate them as the final output feature of layer t.

Our deformable spatial transformer has two parts: $A_i^{(t)}P$ and $\mathcal{C}_i^{(t)}\mathcal{F}^{(t-1)}$, for a linear transformation of 3D spatial coordinates and a nonlinear transformation of point features (which reflect semantics) respectively. In Section 4.5, we provide empirical analysis of these two components.

3.3 Spatial Transformer Networks

We spatially transform the input point cloud in order to obtain dynamic local neighborhoods for point convolutions. The transformer can be easily added to existing point cloud processing networks. We first describe the procedure and then provide three applications with several networks.

Point Cloud Networks with Spatial Transformers. Consider segmenting N 3D points into C classes as an example. Fig.4 depicts a general network architecture for point cloud segmentation, where several spatial transformers are used at different layers. At layer t, we learn $k^{(t)}$ transformation matrices $\{A_i^{(t)}\}_{i=1}^{k^{(t)}}$, apply each to the input point coordinates P, and then compute the point affinity matrices $\{S_i^{(t)}\}_{i=1}^{k^{(t)}}$, e.g., based on k-NN graphs for the edge convolution [22].

For each sub-transformation, we learn a feature $F_i^{(t)}$ of dimension $N \times f_i^{(t)}$. We then concatenate all $k^{(t)}$ features at this layer to form an output feature $\mathcal{F}^t$ of dimension $N \times \mathfrak{f}^{(t)}$, where $\mathfrak{f}^{(t)} = \sum_i^{k^{(t)}} f_i^{(t)}$. The output feature serves as the input to the next layer for further feature learning.

Note that affine or projective transformation matrices are applied to the original point cloud coordinates P, since each layer has not just one but multiple spatial transformers. However, the deformable transformation matrix $\mathcal{C}_i^{(t)}$ is applied to the previous feature map, the feature transformation component is thus progressively learned.

By stacking several such transformation learning blocks and finally a fully connected layer of dimension C, we can map the input point cloud to the segmentation map of dimension $C \times N$, or downsample to a vector of dimension C for classification tasks. For the spatial transformer block in a point cloud detection network (Fig.5), C is the dimension of the output feature. We train the network end-to-end.

Classification Networks. A point cloud classifier [19], [23] takes 3D points, learns features from their local neighborhoods, and outputs C classification scores, where C is the number of classes. We add spatial transformers at each layer to obtain different local neighborhoods for feature learning.

Point-based Segmentation Networks. These networks [19], [18], [23], [22] take 3D points and compute their point affinity matrices and local neighborhoods from the point coordinates. Features are learned by applying convolution operators on the points and their local neighborhoods.

We use the *edge convolution* in [22] as our baseline, which takes relative point coordinates as inputs and achieves the state-of-the-art performance. Specifically, we retain their learning settings and simply insert spatial transformers to generate new local neighborhoods for the edge convolutions.

Sampling-based Segmentation Networks. To demonstrate the general applicability of our spatial transformers, we consider point affinity matrices on transformed point clouds as defined in sampling-based networks such as SplatNet [29].

SplatNet groups 3D points onto a permutohedral lattice [30] and applies bilateral filters [31] on the grouped points to get features. The permutohedral lattice defines the local neighborhoods of every point and makes the bilateral convolution possible. We add spatial transformers to deform the point cloud and form various new lattices. The local neighborhoods can dynamically configure for learning point cloud semantics. We keep all the other settings of SplatNet.

Detection Networks. Detecting objects in a 3D point cloud generated from e.g. LiDAR sensors is important for autonomous navigation, housekeeping robots, and AR/ VR. These 3D points are often sparse and imbalanced across semantic classes. Our spatial transformers can be added to a detection network and improve feature learning efficiency and task performance with dynamic local neighborhoods.

Our baseline is VoxelNet [39], the state-of-the-art 3D object detector for autonomous driving data. We adopt all its settings, and add spatial transformers on the raw point cloud data, before point grouping (Fig.5). To demonstrate that spatial transformers enhance feature learning for point cloud processing, we let transformers only affect point features but not point coordinates for grouping. With spatial transformers, point coordinates could also be transformed at the grouping stage, which would lead to non-cuboid 3D detection boxes. Although interesting, we do not explore this variation and deem it beyond the scope of this paper.

3.4 Relevance to Other Works

We review related works on deformable convolutions [40], [38] and DGCNN [22].

Deformable Convolutions. Deformable convolutional networks [40] learn dynamic local neighborhoods for 2D images. Specifically, at each location $\mathbf{p_0}$ of the output feature map Y, deformable convolutions modify the regular grid R with offsets $\{\Delta\mathbf{p_n}\}_{n=1}^{N}$, where $N = |R|$. The output on input X by convolution with weight $\mathbf{w}$ becomes:

$$Y(\mathbf{p_0}) = \sum_{\mathbf{p_n} \in \mathbf{R}^3} w(\mathbf{p_n}) X(\mathbf{p_0} + \mathbf{p_n} + \Delta\mathbf{p_n}) \tag{8}$$

Note that KPConv [38] directly adapts this formula to point clouds as deformable point convolutions. Although also achieving dynamic local neighborhoods, our spatial transformers alter neighborhoods differently:

1) Deformable point convolutions learn to alter each neighborhood with an offset to the regular grid R. We learn global transformations on the input point cloud and the metric of defining local neighborhoods changes. Global transformations like affine transformations can retain the global geometric properties such as collinearity and parallelism, while local transformations has no such constraints as only local neighborhoods are available.
2) Offsets of deformable point convolutions are dependent upon feature values, while transformation matrices of our spatial transformers are dependent upon point coordinates for affine and projective transformations, or both point coordinates and feature values for deformable transformations. Access to point coordinates provides additional information and regularzation.

Dynamic Graph CNN. Dynamic local neighborhoods have also been explored in DGCNN [22] for point cloud processing. It has three main differences with our work.

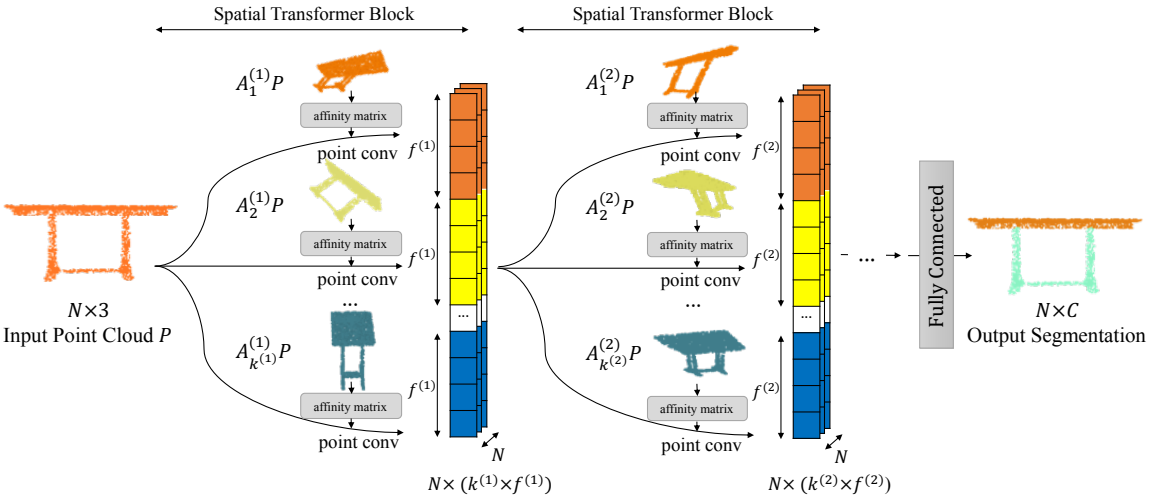

Fig. 4. The point cloud segmentation network with spatial transformers. Our network consists of several spatial transformers. At each layer, we learn k transformation matrices A to apply to the input point cloud P, and compute the corresponding point affinity matrices based on their k-NN graphs. For each sub transformation, we can learn a sub-feature, and then concatenate all features to form an output feature of dimension $f \times N$. The output feature will be used for the next spatial transformer block for feature learning. By stacking several such transformation learning blocks and finally a fully connected layer of dimension C (the number of class), we can map the input point cloud to the $C \times N$ segmentation map.

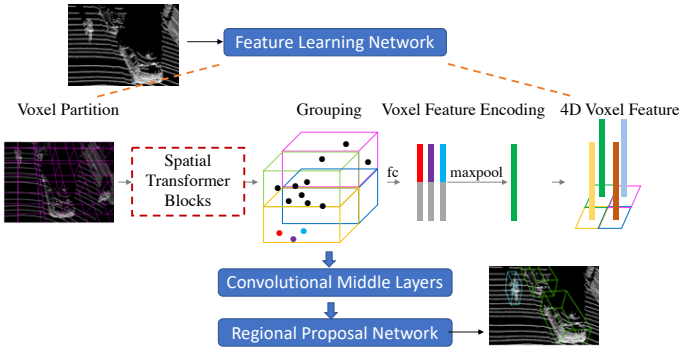

Fig. 5. The object detection network. We add spatial transformers to the the point feature learning network of [39] for obtaining dynamic local neighborhoods. Transformers only affect feature learning but not point coordinates for grouping.

1) How neighborhoods are defined is different. DGCNN uses high-dimensional feature maps to construct the point affinity matrix and generate local neighborhoods. Our local neighborhoods are from transformed point clouds. Reusing point features for defining neighborhoods may be straightforward, but reduce the distinction between spatial and semantic information and hurt generalization.
2) It is computationally costly to build dense nearest neighbor graphs in a high-dimensional feature space.
3) DGCNN [22] uses only one nearest neighbor graph at different layers, whereas we have multiple graphs at each layer for capturing different geometric transformations.

With less computational cost and more flexibility in geometric transformations, we achieve better empirical performance on semantic segmentation (Table 2 and Table 3).

4 Experiments

We conduct comprehensive experiments to verify the effectiveness of our spatial transformers. We benchmark with two types of networks, point-based and sampling-based metrics for defining point neighborhoods, on four point cloud processing tasks: classification, part segmentation, semantic segmentation and detection. We conduct ablation studies on deformable spatial transformers. We further provide visualization, analysis and insights of our method.

4.1 Classification

We benchmark on ModelNet40 3D shape classification [11]. We add transformers to two baselines [22], [29] and adopt the same network architecture, experimental setting and evaluation protocols. Table 1 and Fig.4.1 show that adding spatial transformers to point-based and sampling-based method gives 1% and 2% gain.

In addition, our performance gain over [22], which builds one per-layer dynamic neighborhood graphs with high-dimensional point features, demonstrates the advantages of our method of building multiple dynamic neighborhood graphs with transformed 3D point coordinates.

Fig.12 shows that spatial transformers align the 3D shape better according to its semantics. We augment training and testing data with random rotations, and observe that spatial transformers gain 3% over its fixed graph counterpart.

4.2 Part Segmentation

We benchmark on ShapeNet part segmentation [41], where the goal is to assign a part category label (e.g. chair leg, cup handle) to each 3D point. The dataset contains $16,881$ shapes from 16 categories, annotated with 50 parts in total, and the number of parts per category ranges from 2 to 6.

4.2.1 *Point-based Method*

Network Architectures. *Point-based* methods construct neighborhoods based on point coordinate operations such as *edge convolution* for our baseline DGCNN [22]. We follow the same network architecture and evaluation protocols of [22]. The network has 3 convolutional layers; the output feature dimension is 64. To capture information at different levels, all

TABLE 1
Spatial transformers improves ModelNet40 classification accuracy. We report ModelNet40 classification accuracy of different baselines, without and with spatial transformers, with or without random rotations. *Point-based* refers to the baseline method adopting Euclidean-distance based affinity matrices [22]. *Sampling-based* refers to the baseline method adopting permutohedral-lattice based affinity matrices [29]. We observe accuracy gains for different baseline networks with spatial transformers. Transformer gains are invariant to input rotations.

			Point-based				Point-based with rand. input rotations		Samplin-based			
	PointNet [18]	DGCNN [22]	[22] (fixed)	Affine	Proj.	Deformable	[22] (fixed)	Deformable	SplatNet[29]	Affine	Proj.	Deformable
Avg.	86.2	89.2	88.8	89.3	89.2	**89.9**	85.7	**88.3**	86.3	87.4	87.1	**88.6**

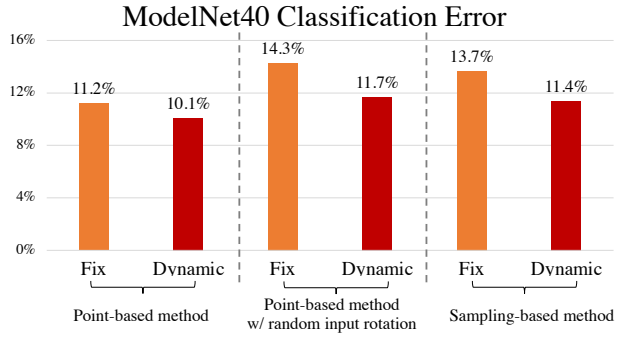

Fig. 6. Spatial transformers lead to higher accuracy and more rotation invariance on ModelNet40. We report classification errors for different baselines, without and with spatial transformers, with or without random rotations. Transformers consistently lead to the lower errors than fixed graph baselines, and the improvement is larger upon random rotations.

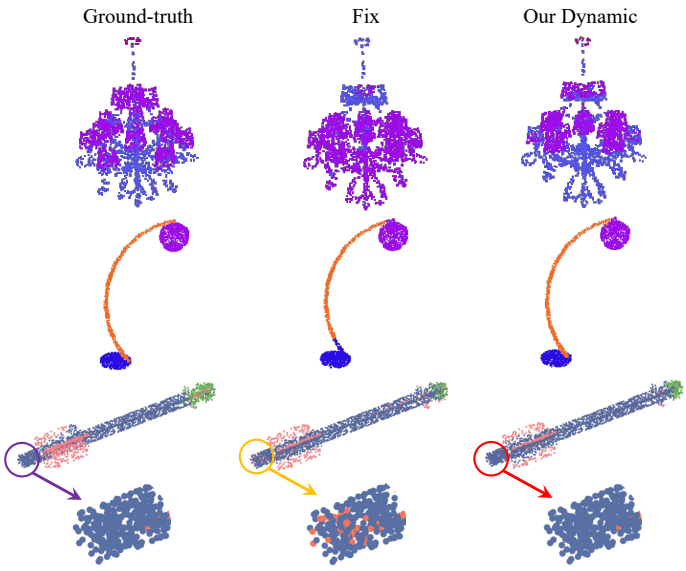

Fig. 7. Spatial transformers improve the part segmentation performance. We show part segmentation results of different baselines, where different parts are marked with different colors. With spatial transformers, part segmentation for objects with less rigid and more complicated structures improves (1st and 2nd row, lamp). The segmentation consistency within each part also improves (3rd row, rocket).

the convolutional features are concatenated and fed through several fully connected layers to output the segmentation.

As a *fixed graph* baseline, we use the same input point coordinates as the metric to define fixed local neighborhoods. We insert spatial transformers to alter the metric for defining point neighborhoods for edge convolutions. There are point-based *affine*, *projective* and *deformable* networks when inserting different spatial transformers (Section 3.2). As for classification, [22] directly uses learned features to build point affinity matrices for dynamic neighborhoods.

We follow [22] and use three edge convolution layers. At each layer, we keep the number of graphs k and sub-graph feature dimension f the same, and search for the best architecture. We report results of affine, projective and deformable networks with $k = 4$, $f =!32$. For fair comparisons, we increase the number of channels of baselines so all the methods have the same number of parameters.

Results and Analyses. In Table 2, we report the instance average mIOU (mean intersection over union), as well as the mIOU of some representative categories in ShapeNet. Compared with the fixed graph baseline, the affine, projective and deformable spatial transformers achieve 0.5%, 0.2% and 1.1% improvement respectively and beats the fixed graph baseline methods in most categories. Specifically, we observe 8.0%, 8.3% and 4.7% performance boost with spatial transformers over the fixed graph baseline. Our deformable spatial transformers gain 4.0% over [22].

We also beat other state-of-the-art methods [18], [19], [20] by a significant margin. Adding deformable spatial transformers to PointCNN [23] gains 6% (4%) on motorbike (bag) and 1% on average. We observe that categories with fewer samples are more likely to gain possibly due to regularization by transformers. Fig.7 shows that deformable spatial transformers make more smooth predictions and achieve better performance than the fixed graph baseline.

From affine to deformable transformations, the performance increases as the degree of freedom increases for the transformer. Projective transformers, however, perform slightly worse than affine transformers. The performance drop could result from geometrical distortion caused by mapping 3D points with homogeneous coordinates. Furthermore, for deformable transformers, when removing the constraint that the transformed points should be similar to the input point cloud (Fig.8, feature only $G = CF$), the performance also drops, indicating the necessity of the proposed similar-to-input constraint on spatial transformers.

4.2.2 *Sampling-based Method*

Network Architectures. *Sampling-based* methods construct neighborhoods are based on sampling operations on point coordinates. SplatNet [29] groups points on permutohedral lattices and applies learned bilateral filters [31] on naturally defined local neighbors to extract features. We follow the same architecture as SplatNet [29]. The network starts with a single 1×1 regular convolutional layer, followed by 5 bilateral convolution layers (BCL). The output of all BCL are concatenated and fed to a final 1×1 regular convolutional

TABLE 2
Spatial transformers improve part segmentation performance. We report mIoU(%) on ShapeNet PartSeg dataset. Compared with several other methods, deformable spatial transformers achieve the SOTA in average mIoU.

	Avg.	aero	bag	cap	car	chair	earphone	guitar	knife	lamp	laptop	motorbike	mug	pistol	rocket	skateboard	table
# shapes		2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271
3DCNN [18]	79.4	75.1	72.8	73.3	70.0	87.2	63.5	88.4	79.6	74.4	93.9	58.7	91.8	76.4	51.2	65.3	77.1
PointNet[18]	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++ [19]	85.0	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**
FCPN [20]	81.3	84.0	82.8	86.4	**88.3**	83.3	73.6	**93.4**	87.4	77.4	**97.7**	**81.4**	95.8	**87.7**	**68.4**	**83.6**	73.4
DGCNN [22]	81.3	84.0	82.8	86.4	78.0	**90.9**	76.8	91.1	87.4	83.0	95.7	66.2	94.7	80.3	58.7	74.2	80.1
Point-based [22] fixed graph	84.2	83.7	82.4	84.0	78.2	**90.9**	69.9	91.3	86.6	82.5	95.8	66.5	94.0	80.8	56.0	73.8	79.8
Point-based affine	84.7	84.1	83.5	86.9	79.6	**90.9**	72.5	91.6	88.2	83.3	96.1	68.9	95.3	83.3	60.9	75.2	79.7
Point-based projective	84.4	84.3	84.2	88.5	77.9	90.4	72.8	91.2	86.6	81.7	96.0	66.6	94.8	81.3	61.6	72.1	80.5
Point-based deformable	85.3	**84.6**	83.3	88.7	79.4	**90.9**	77.9	91.7	87.6	83.5	96.0	68.8	95.2	82.4	64.3	76.3	81.5
Point-based deformable random	84.7	84.3	84.4	83.2	78.9	90.8	75.6	91.4	87.1	83.0	95.9	66.8	94.8	82.1	62.3	75.7	80.4
PointCNN [23]	84.9	82.7	82.8	82.5	80.0	90.1	75.8	91.3	**87.8**	82.6	95.7	69.8	93.6	81.1	61.5	80.1	81.9
PointCNN deformable	**85.8**	83.4	**86.6**	85.5	79.1	90.3	**78.5**	91.6	**87.8**	84.2	95.8	75.3	94.6	83.3	65.0	80.7	81.7
Sampling-based baseline [29]	84.6	81.9	83.9	88.6	79.5	90.1	73.5	91.3	84.7	84.5	96.3	69.7	95.0	81.7	59.2	70.4	81.3
Sampling-based projective	84.4	82.1	84.0	**89.1**	77.9	89.6	73.7	91.1	83.3	83.0	96.3	67.2	94.5	79.8	60.0	68.8	82.1
Sampling-based deformable	85.2	82.9	83.8	87.6	79.6	90.6	73.0	92.2	86.1	**85.7**	96.3	72.7	**95.8**	83.1	65.1	76.5	81.3

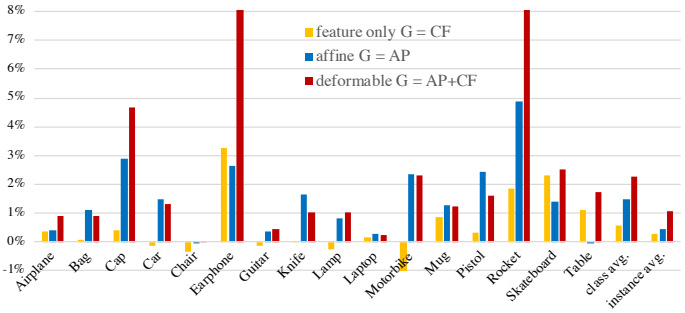

Fig. 8. Transforming both point cloud coordinates and features for dynamic local neighborhoods leads to the largest gain. We report different parts of deformable transformers' performance gain over fixed local neighborhood baseline on ShapeNet part segmentation. 0% means achieving the same accuracy as fixed local neighborhood baseline and negative value means achieving worse accuracy than fixed neighborhood baseline. Compared with preserving either affine part AP or feature part CF, deformable spatial transformers ($AP + CF$) achieves largest gains on every category, specifically 8% gains on earphone and rocket.

layer to get the segmentation output. Since each BCL directly takes raw point locations, we consider it as a fixed graph baseline. We add deformable spatial transformers to the network and feed transformed point graphs to BCL to construct permutohedral lattices. With gradients on the permutohedral lattice grid, we can make the transformation matrix learned end-to-end. Note that we increase the channel of convolution layers for fair comparisons.

Results and Analyses. Table 2 shows that our deformable spatial transformers (with $k = 1$ at all BCLs) gains over the sampling-based fixed graph baseline [29] in most categories with 0.6% on average and 5.9% for the rocket category. It also beats other state-of-the-art baselines.

4.3 Semantic Segmentation

We benchmark on the Stanford 3D semantic parsing dataset [42]. It contains 3D scans by Matterport covering 6 areas and 271 rooms. Each point is annotated into one of 13 categories such as *chair, table, floor, clutter*. We follow the data processing procedure of [18]: We first split points by room, and then sample rooms into several 1m $\times$1m blocks. When training, 4096 points are sampled from the block on the fly. We train our network to predict the point class in each block, where each point is represented by 9 values: XYZ, RGB and its [0,1]-normalized location with respect to the room.

TABLE 3
Spatial transformers improve semantic segmentation performance. We report mIoU(%) on S3DIS semantic segmentation dataset. Adding spatial transformers to [22] and [29] improves the performance.

	PointNet[18]	DGCNN[22]	[22](FIXED)	[22]+AFF	[22]+DEF	SplatNet [29]	[29]+DEF
	47.7	56.1	56.0	56.9	**57.2**	54.1	55.5
	ceiling	floor	wall	beam	column	window	clutter
[22](FIXED)	92.5	93.1	76.1	51.0	**41.7**	**49.6**	46.8
[22]+AFF	92.7	**93.6**	76.7	52.6	41.2	48.7	47.8
[22]+PROJ	92.5	93.5	76.7	52.7	40.7	48.5	**48.0**
[22]+DEF	**92.8**	**93.6**	**76.8**	**52.9**	41.1	49.0	**48.0**
	door	table	chair	sofa	bookcase	board	
[22](FIXED)	63.4	61.8	43.1	23.3	**42.0**	43.5	
[22]+AFF	**63.7**	63.4	45.1	27.0	41.3	44.8	
[22]+PROJ	**63.5**	62.3	44.8	27.0	41.5	44.9	
[22]+DEF	63.5	**64.2**	**45.2**	**28.1**	41.7	**46.1**	

TABLE 4
Spatial transformers improve object detection performance. We report car detection AP(%) on KITTI validation set. Adding spatial transformers leads to 2% performance gain.

	birds' eye			3D		
	Easy	Medium	Hard	Easy	Medium	Hard
VoxelNet[39]	77.3	59.6	51.6	43.8	32.6	27.9
VoxelNet + fixed graph	84.3	67.2	59.0	45.7	34.5	32.4
VoxelNet + deformable	**85.3**	**69.1**	**60.9**	**46.1**	**35.9**	**34.0**

Network Architectures. We adopt DGCNN [22] as Section 4.2, with $C = 13$, the number of semantic categories.

Results and Analyses. In terms of average mIoU, Table 3 shows that affine and deformable spatial transformers gain 0.9% and 1.2% respectively over the fixed graph baseline. Deformable transformers also gain 1.1% over [22] and beat all other state-of-the-art methods. Likewise for sampling-based methods [29], we observe 1.4% gain.

As for part segmentation, semantic segmentation performance improves when point clouds are given more freedom to deform (from affine to deformable spatial transformers) based on transformation of original locations and feature projections. Projective transformers give least performance gain, suggesting that mapping 3D points via homogeneous coordinates may not be most efficient.

Fig.9 shows that semantic segmentation results are smoother and more robust to missing points and occlusions with our deformable transformers.

4.4 3D Object Detection

We benchmark on KITTI 3D object detection [43]. It contains 7,481 training images / point clouds and 7,518 test images / point clouds, covering three categories: Car, Pedestrian, and Cyclist. For each class, detection outcomes are evaluated

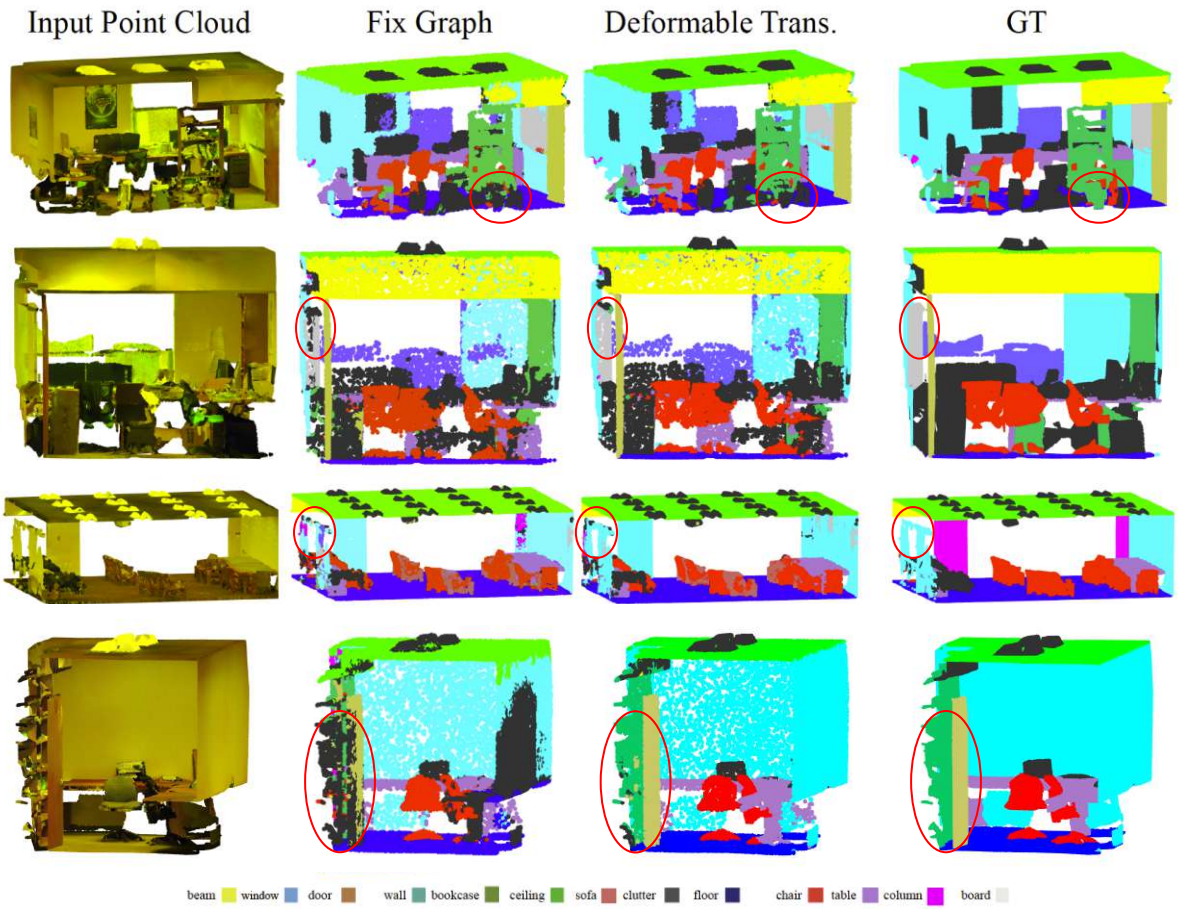

Fig. 9. Spatial transformers improve semantic segmentation results. We show qualitative visualizations for semantic segmentation of deformable spatial transformers and the fixed local neighborhood baseline. The first column is the input point cloud, the second and the third column shows the fixed graph and our spatial transformer results, and the last column is the ground truth. Points belonging to different semantic regions are colored differently. We observe better and more consistent segmentation result with our spatial transformer, specifically for the areas circled in red.

based on three difficulty levels: easy, moderate, and hard, according to the object size, occlusion state and truncation level. We follow the evaluation protocol in VoxelNet [39] and report car detection results on the validation set.

Network Architectures.. Shown in Fig.5, the network first partitions raw 3D points into voxels. We add deformable spatial transformers; points in each voxel are represented with point features. There are two deformable feature learning layers, each layer having 2 sub-graphs with 16-dimensional outputs. Note that the voxel partition is based on the input point coordinates. As in VoxelNet, the point features in each voxel are fed to 2 voxel feature encoding layers with channel 32 and 128 to get sparse 4D tensors representing the space. The middle convolutional layers process 4D tensors to further aggregate spatial contexts. Finally a Region Proposal Network (RPN) generates the 3D detection.

We report the performance of 3 networks: **(1)** VoxelNet baseline [39]; **(2)** the fixed graph baseline, where we used the original point cloud location to learn the point feature at the place of spatial transformer blocks; **(3)** deformable spatial transformer networks as discussed above.

Results and Analyses. Table 4 reports car detection results on KITTI validation set.[1] Compared with baseline, having a point feature learning module improves the performance by 7.3% and 2.8% for birds' eye view and 3D detection performance on average, respectively. The deformable module further improves 8.9% and 3.9% respectively over VoxelNet.

1. The authors did not provide code. We use the implementation by [44] and obtain lower performance than the original paper.

TABLE 5
Performance of different number of deformable transformation modules. Metric is average mIOU (%).

	fixed graph	1 graph	2 graphs	4 graphs
$f_i^{(t)} = 32$	84.2	84.9	85.2	85.3
$f_i^{(t)} k_i^{(t)} = 64$	84.2	85.3	85.2	83.5

In the first row, the output feature of each sub-graph is of dim. 32, while the number of subgraphs changes; the second row limits the multiplication of number of sub-graphs and sub-feature dim. to be 64.

4.5 Ablation Studies

We conduct ablation studies to understand how many spatial transformers may be sufficient to achieve satisfactory performance. We also study transformations of point coordinates and features of deformable spatial transformers. The influences of updating transformation matrices and transformers at different layers are investigated.

The Number of Transformers. Table 5 shows that for the fixed sub-feature dimension, the more graphs in each layer, the higher the performance. With the fixed complexity, (i.e.,

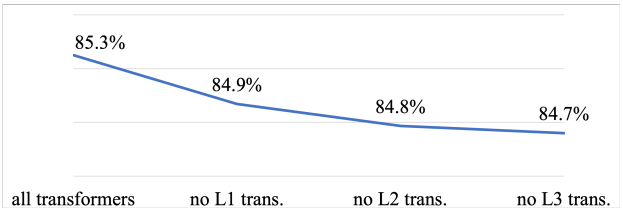

Fig. 10. Part segmentation performance (average mIOU) of deformable transformers at different layers. When applying all transformers at three layers, the performance is highest. Removing transformers at different layer lead to performance drop. Removing transformers at layer 3 gives the most performance drop.

the product of the number of sub-graphs and the sub-feature dimension fixed at 64), the best performance is achieved at $k = 1, f = 64$ and $k = 2, f = 32$.

Two Components in Deformable transformers. A deformable spatial transformer has two components (Equation 7): affine transformation on point coordinates, AP, and three-dimensional projection of high-dimensional feature, CF. Fig.8 shows that both affine and feature only spatial transformers also improve performance, but the combination of both leads to the largest gain.

Updating Transformation Matrices. The transformation matrices are updated in an end-to-end fashion with the ultimate goal of increasing the task performance. It is of interest to understand if updating transformation matrix boosts the performance. Specifically, we randomly initialize transformation matrices of deformable spatial transformers and keep them not updated during training. The performance is 0.5% better than fixed graphs, indicating that adding more transformation graphs at different layers helps; however, it is 0.6% worse than updating transformation matrices, indicating learning to update transformation matrices in an end-to-end fashion is helpful.

Transformers at Different Layers. We start with all deformable transformers effective at three layers, and remove transformers at one layer a time. Fig.10 shows that for part segmentation, the performance is best with all transformers, whereas removing transformers at layer 3 gives the largest performance drop, suggesting that transformers at every layer help and those at the last layer are most important.

4.6 Time and Space Complexity

With spatial transformers, the model size changes little and the inference takes slightly more time (Table 6). Note that for fair comparisons, we increase the number of channels in the fixed graph baseline model for all the experiments. Even without increasing the number of parameters of baselines (not shown in Table 6), adding spatial transformers only increases the number of parameters by 0.1%, as the number of parameters of spatial transformers (only transformation matrices) is very small.

TABLE 6
Model size and test time on ShapeNet part segmentation. Spatial transformers slightly increase the inference time.

	Sampling-based		Point-based	
	[29]	[29] + transformer	[22] (fixed)	[22] + transformer
# Params.	2,738K	2,738K	2,174K	2,174K
Inference time (s/shape)	0.352	0.379	0.291	0.315

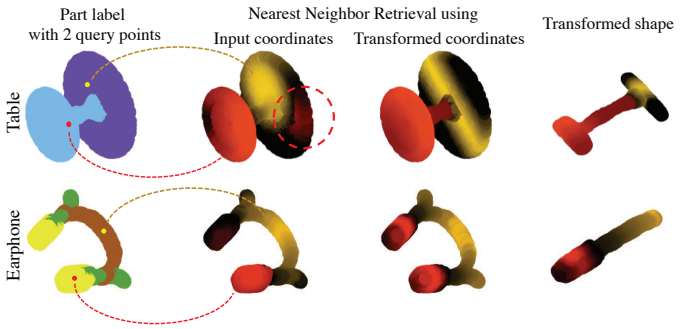

Fig. 11. Local neighborhoods of two query points (red and yellow) using (transformed) 3D coordinates with nearest neighbor retrieval. Neighborhoods of transformed point clouds makes semantic information extraction more efficient: the neighborhood inside the dashed circle adapts to table base part. View rotating version for better visualization.

4.7 Visualization and Analysis

We visualize the change in local neighborhoods when applying spatial transformers. We also visualize the transformed 3D points globally and locally.

Dynamic Neighborhood Visualization. To illustrate how our spatial transformers learn diverse neighborhoods for 3D shapes, we show the nearest neighbors of two query points and use corresponding colors to indicate corresponding neighborhoods. **(1)** Fig.11 shows that neighborhoods retrieved from deformed shapes encode additional semantic information, compared to neighborhoods from 3D coordinates. **(2)** Fig.2 shows that for table and earphone, different graphs enable the network to learn from diverse neighborhoods without incurring additional computational cost.

Global Visualization of Deformable Transformations. Fig.12 depicts some examples of learned deformable transformations in ShapeNet part segmentation. Each graph at a certain layer aligns the input 3D shape with similar semantic geometric transformations. For example, regardless of the shape of the rocket, graph 2 at layer 2 always captures the rocket wing information.

Local Distributions after Deformable Transformations. 3D Points often do not have balanced sampling, which makes point convolution challenging, as the k-NN graph does not accurately represents the exact neighborhood and 3D structure information. Our deformable spatial transformer gives every point flexibility and finds better neighborhoods.

We wonder if transformers make the point cloud closer to balanced sampling. We normalize the point coordinates for fair comparisons. Fig.13 visualizes the local distribution around a sample point on skateboard: After deformable transformation, the points are moved to a more uniform distribution. We analyze the standard deviation of raw and transformed point cloud coordinates in ShapeNet data. The standard deviation of point coordinates decreases 50.2% over all categories after spatial transformations, indicating a more balanced distribution of transformed points.

We check if the point coordinates are statistically different before and after the application of transformers. We perform t-test on the original and transformed point clouds. The t-score is 7.15 over all categories with p-value smaller than 1e-9. The transformed point cloud distribution is thus statistically different from the input point cloud distribution.

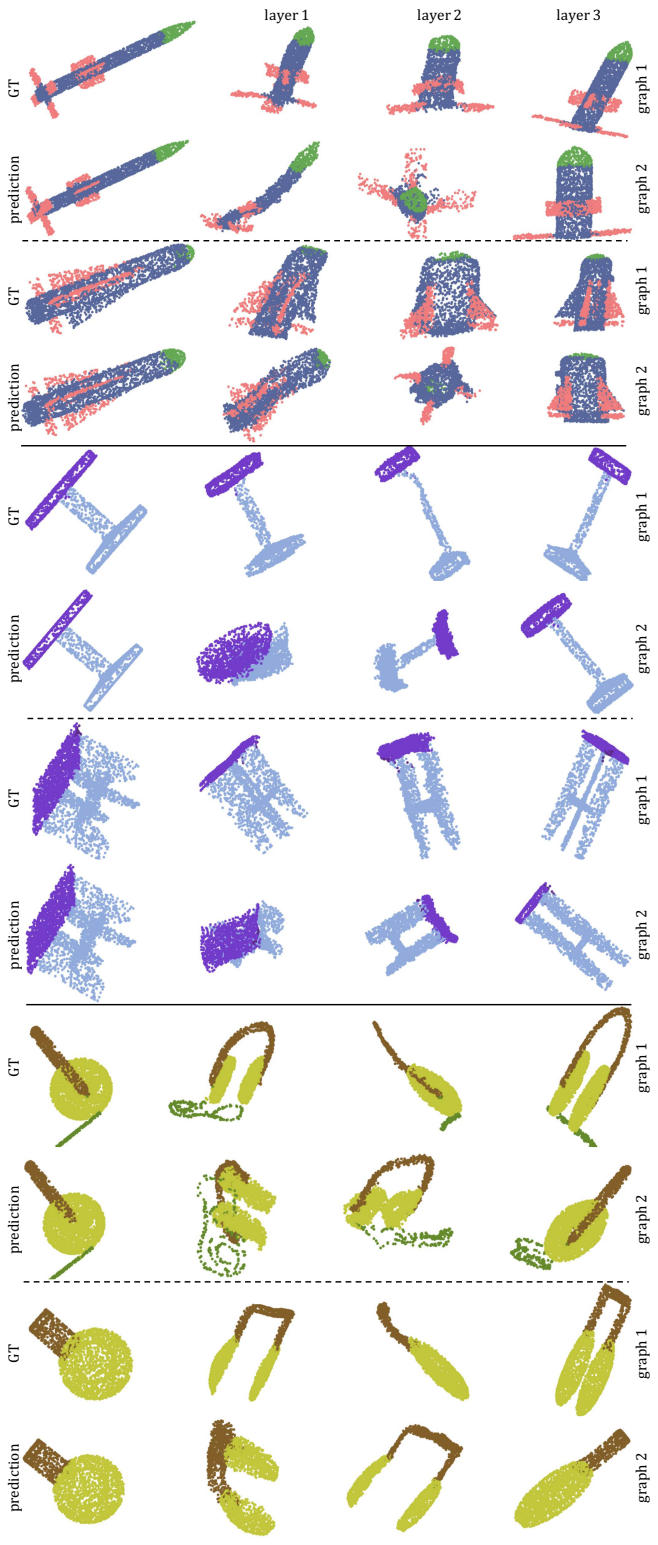

Fig. 12. Examples of learned deformable transformations in ShapeNet part segmentation. 3D shapes include rocket, table and earphone (from up to bottom). Every two rows depict an instance with learned transformations. We observe that each transformation at certain layer aligns input 3D shape with similar semantic geometric transformation, e.g., graph 2 at layer 2 in rocket examples captures rocket wings. Graph 2 at layer 1 in table examples captures table surfaces.

5 Conclusion

We propose novel spatial transformers for 3D point clouds that can be easily added onto to existing point cloud processing networks. They can dynamically alter local point neighborhoods for better feature learning.

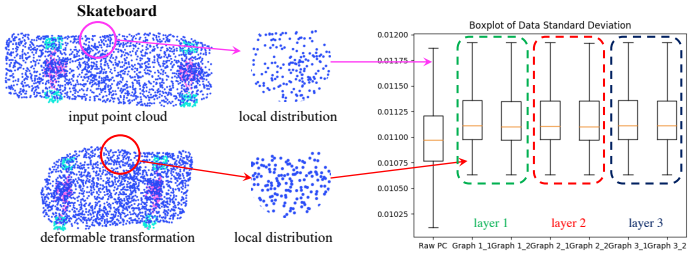

Fig. 13. Spatial transformers improve the point cloud processing efficiency by improving local distributions of points. We show local distributions of a point cloud without and with transformers. The standard deviation of the transformed point cloud is smaller, enhancing the local neighborhood grouping (e.g. when using k-NN for affinity matrices, more balanced point distributions make feature learning in each neighborhood suffer less variations and outliers) and feature learning efficiency.

We study one linear (affine) transformer and two non-linear (projective and deformable) transformers. We benchmark them on point-based [22], [23] and sampling-based [29] point cloud networks and on three large-scale 3D point cloud processing tasks (part segmentation, semantic segmentation and object detection). Our spatial transformers outperform the fix graph counterpart for state-of-the-art methods.

There are some limitations of our spatial transformers. First, there are not many constraints on deformable spatial transformers to capture the geometry of the 3D point clouds. More complex non-linear spatial transformers may further improve the performance. On the other hand, spatial transformers learn global transformations of 3D point clouds for altering local neighborhoods. It is unclear if combining both global and local transformations [40], [38] would further improve the learning capacity and task performance.

Acknowledgements. This research was supported, in part, by Berkeley Deep Drive and DARPA. The authors thank Utkarsh Singhal and Daniel Zeng for proofreading, and anonymous reviewers for their insightful comments.

References

[1] C.-H. Lin, Y. Chung, B.-Y. Chou, H.-Y. Chen, and C.-Y. Tsai, "A novel campus navigation app with augmented reality and deep learning," in *Proc. Int. Conf. App. Sys. Invent.*, 2018, pp. 1075–1077.

[2] J. R. Rambach, A. Tewari, A. Pagani, and D. Stricker, "Learning to fuse: A deep learning approach to visual-inertial camera pose estimation," in *Proc. Int. Symp. Mix. & Aug. Real.*, 2016, pp. 71–76.

[3] S. Tulsiani, S. Gupta, D. Fouhey, A. A. Efros, and J. Malik, "Factoring shape, pose, and layout from the 2d image of a 3d scene," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 302–310.

[4] S. Vasu, M. M. MR, and A. Rajagopalan, "Occlusion-aware rolling shutter rectification of 3d scenes," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 636–645.

[5] A. Dai, D. Ritchie, M. Bokeloh, S. Reed, J. Sturm, and M. Nießner, "Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans," in *Proc. Conf. Comput. Vis. Pat. Recog.*, vol. 1, 2018, p. 2.

[6] Y. Chen, J. Wang, J. Li, C. Lu, Z. Luo, H. Xue, and C. Wang, "Lidar-video driving dataset: Learning driving policies effectively," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 5870–5878.

[7] P. Li, T. Qin *et al.*, "Stereo vision-based semantic 3d object and ego-motion tracking for autonomous driving," in *Proc. EUR Conf. Comput. Vis.*, 2018, pp. 646–661.

[8] B. Wu, A. Wan, X. Yue, and K. Keutzer, "Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud," in *Proc. Int. Conf. Rob. & Auto.*, 2018, pp. 1887–1893.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2009, pp. 248–255.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Conf. NeruIPS*, 2012, pp. 1097–1105.

[11] 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. Conf. Comput. Vis. Pat. Recog.*, 2015, pp. 1912–1920.

[12] G. Riegler, A. O. Ulusoy, and A. Geiger, "Octnet: Learning deep 3d representations at high resolutions," in *Proc. Conf. Comput. Vis. Pat. Recog.*, vol. 3, 2017.

[13] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-cnn: Octree-based convolutional neural networks for 3d shape analysis," *ACM Transactions on Graphics*, vol. 36, no. 4, p. 72, 2017.

[14] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proc. Int. Conf. Comput. Vis.*, 2015, pp. 945–953.

[15] 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. Conf. Comput. Vis. Pat. Recog.*, 2016, pp. 5648–5656.

[16] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri, "3d shape segmentation with projective convolutional networks," in *Proc. Conf. Comput. Vis. Pat. Recog.*, vol. 1, no. 2, 2017, p. 8.

[17] L. Zhou, S. Zhu, Z. Luo, T. Shen, R. Zhang, M. Zhen, T. Fang, and L. Quan, "Learning and matching multi-view descriptors for registration of point clouds," in *Proc. EUR Conf. Comput. Vis.*, 2018, pp. 505–522.

[18] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *Proc. Conf. Comput. Vis. Pat. Recog.*, vol. 1, no. 2, p. 4, 2017.

[19] 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. Conf. NeurIPS*, 2017, pp. 5099–5108.

[20] D. Rethage, J. Wald, J. Sturm, N. Navab, and F. Tombari, "Fully-convolutional point networks for large-scale point clouds," in *Proc. EUR Conf. Comput. Vis.*, 2018, pp. 596–611.

[21] M. Gadelha, R. Wang, and S. Maji, "Multiresolution tree networks for 3d point cloud processing," in *Proc. EUR Conf. Comput. Vis.*, 2018, pp. 103–118.

[22] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions On Graphics*, vol. 38, no. 5, pp. 1–12, 2019.

[23] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," in *Proc. Conf. NeurIPS*, 2018, pp. 820–830.

[24] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, "Gvcnn: Group-view convolutional neural networks for 3d shape recognition," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 264–272.

[25] Z. Han, M. Shang, Z. Liu, C.-M. Vong, Y.-S. Liu, M. Zwicker, J. Han, and C. P. Chen, "Seqviews2seqlabels: Learning 3d global features via aggregating sequential views by rnn with attention," *IEEE Transactions on Image Processing*, vol. 28, no. 2, pp. 658–672, 2019.

[26] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Proc. Int. Conf. Intell. Rob. Sys.*, 2015, pp. 922–928.

[27] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs," in *Proc. Int. Conf. Comput. Vis.*, 2017, pp. 2088–2096.

[28] R. Klokov and V. Lempitsky, "Escape from cells: Deep kd-networks for the recognition of 3d point cloud models," in *Proc. Int. Conf. Comput. Vis.*, 2017, pp. 863–872.

[29] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz, "Splatnet: Sparse lattice networks for point cloud processing," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 2530–2539.

[30] A. Adams, J. Baek, and M. A. Davis, "Fast high-dimensional filtering using the permutohedral lattice," in *Computer Graphics Forum*, vol. 29, no. 2, 2010, pp. 753–762.

[31] V. Jampani, M. Kiefel, and P. V. Gehler, "Learning sparse high dimensional filters: Image filtering, dense crfs and bilateral neural networks," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2016, pp. 4452–4461.

[32] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 4558–4567.

[33] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun, "Deep parametric continuous convolutional neural networks," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 2589–2597.

[34] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou, "Tangent convolutions for dense prediction in 3d," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 3887–3896.

[35] H. Zhao, L. Jiang, C.-W. Fu, and J. Jia, "Pointweb: Enhancing local neighborhood features for point cloud processing," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2019, pp. 5565–5573.

[36] J. Li, B. M. Chen, and G. Hee Lee, "So-net: Self-organizing network for point cloud analysis," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 9397–9406.

[37] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Proc. Conf. NeruIPS*, 2015, pp. 2017–2025.

[38] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proc. Int. Conf. Comput. Vis.*, 2019, pp. 6411–6420.

[39] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2018, pp. 4490–4499.

[40] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proc. Int. Conf. Comput. Vis.*, 2017, pp. 764–773.

[41] 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.

[42] 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. Conf. Comput. Vis. Pat. Recog.*, 2016, pp. 1534–1543.

[43] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proc. Conf. Comput. Vis. Pat. Recog.*, 2012, pp. 3354–3361.

[44] Q. Huang, "Voxelnet: End-to-end learning for point cloud based 3d object detection - code," https://github.com/qianguih/voxelnet, 2019.