Autonomous Robots (2022) 46:45–60
https://doi.org/10.1007/s10514-021-09998-1

LatticeNet: fast spatio-temporal point cloud segmentation using permutohedral lattices

Radu Alexandru Rosu[1] · **Peer Schütt**[1] · **Jan Quenzel**[1] · **Sven Behnke**[1]

Received: 14 February 2021 / Accepted: 26 June 2021 / Published online: 19 October 2021

Abstract
Deep convolutional neural networks have shown outstanding performance in the task of semantically segmenting images. Applying the same methods on 3D data still poses challenges due to the heavy memory requirements and the lack of structured data. Here, we propose LatticeNet, a novel approach for 3D semantic segmentation, which takes raw point clouds as input. A PointNet describes the local geometry which we embed into a sparse permutohedral lattice. The lattice allows for fast convolutions while keeping a low memory footprint. Further, we introduce DeformSlice, a novel learned data-dependent interpolation for projecting lattice features back onto the point cloud. We present results of 3D segmentation on multiple datasets where our method achieves state-of-the-art performance. We also extend and evaluate our network for instance and dynamic object segmentation.

Keywords Semantic segmentation · Instance segmentation · Motion segmentation · Sequence segmentation · 3D point cloud

1 Introduction

Environment understanding is a crucial ability for autonomous agents. Perceiving not only the geometrical structure of the scene but also distinguishing between different classes of objects therein enables tasks like manipulation and interaction that were previously not possible. Within this field, semantic segmentation of 2D images is a mature research area, showing outstanding success in dense per pixel categorization on images (Long et al. 2015; Chen et al. 2017; Lin et al. 2017). However, the task of semantically labelling 3D data is still an open area of research as it poses several challenges that need to be addressed.

First, 3D data is often represented in an unstructured manner—unlike the grid-like structure of images. This raises difficulties for current approaches which assume a regular structure upon which convolutions are defined.

Second, the performance of current 3D networks is limited by their memory requirements. Storing 3D information in a dense structure is prohibitive for even high-end GPUs, clearly indicating the need for a sparse structure.

Third, discretization issues caused by imposing a regular grid onto point clouds can negatively affect the network's performance and interpolation is necessary to cope with quantization artifacts (Tchapmi et al. 2017).

In this work, we propose LatticeNet, a novel approach for point cloud segmentation which alleviates the previously mentioned problems. An overview of the input and output of our method can be seen in Fig. 1. Hence, our contributions are:

- A hybrid architecture which leverages the strength of PointNet to obtain low-level features and sparse 3D convolutions to aggregate global context,
- A framework suitable for sparse data onto which all common CNN operators are defined, and
- A novel slicing operator that is end-to-end trainable for mapping features of a regular lattice grid back onto an unstructured point cloud.

This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2070 - 390732324 and by the German Federal Ministry of Education and Research (BMBF) in the project " Kompetenzzentrum: Aufbau des Deutschen Rettungsrobotik-Zentrums" (A-DRZ).

This is one of the several papers published in Autonomous Robotscomprising the Special Issue on Robotics: Science and Systems 2020.

✉ Radu Alexandru Rosu
rosu@ais.uni-bonn.de

[1] Friedrich-Hirzebruch-Allee 8, Bonn, Germany

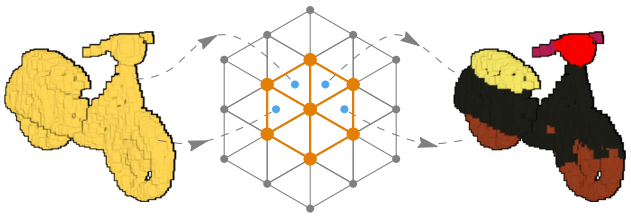


Fig. 1 Semantic segmentation: LatticeNet takes raw point clouds as input and embeds them into a sparse lattice where convolutions are applied. Features on the lattice are projected back onto the point cloud to yield a final segmentation

In addition to our *Robotics: Science and System* conference paper (Rosu et al. 2020) we make the following additional contributions:

- An extension with discriminative loss that allows LatticeNet to perform instance segmentation, and
- A network architecture capable of processing temporal information in order to improve semantic segmentation and to distinguish between dynamic and static objects within the scene.

2 Related work

2.1 Semantic segmentation

3D Semantic segmentation approaches can be categorized depending on data representation upon which they operate.
Point cloud networks The first category of networks operates directly on the raw point cloud.

From this area, PointNet (Qi et al. 2017a) is one of the pioneering works. The method processes raw point clouds by individually embedding the points into a higher-dimensional space and applying max-pooling for permutation-invariance to obtain a global scene descriptor. The descriptor can be used for both classification and semantic segmentation. However, PointNet does not take local information into account which is essential for the segmentation of highly-detailed objects. This has been partially solved in the subsequent work of PointNet++ (Qi et al. 2017b) which applies PointNet hierarchically, capturing both local and global contextual information.

Chen et al. (2018) use a similar approach but they input the point responses w.r.t. a sparse set of radial basis functions (RBF) scattered in 3D space. Optimizing jointly for the extent and center of the RBF kernels allows to obtain a more explicit modelling of the spatial distribution.

PointCNN (Li et al. 2018) deals with the permutation invariance not by using a symmetric aggregation function, but by learning a $K \times K$ matrix for the K input points that permutes the cloud into a canonical form.

Voxel networks 3D Convolutions in this category work on discretized cubic or tetrahedral volume elements.

SEGCloud (Tchapmi et al. 2017) voxelizes the point cloud into a uniform 3D grid and applies 3D convolutions to obtain per-voxel class probabilities. A conditional random field (CRF) is used to smooth the labels and enforce global consistency. The class scores are transferred back to the points using trilinear interpolation. The usage of a dense grid results in high memory consumption while our approach uses a permutohedral lattice stored sparsely. Additionally, their voxelization results in a loss of information due to the discretization of the space. We avoid quantization issues by using a PointNet architecture to summarize the local neighborhood.

Rethage et al. (2018) perform semantic segmentation on a voxelized point cloud and employ a PointNet architecture as a low-level feature extractor. The usage of a dense grid, however, leads to high memory usage and slow inference, requiring various seconds for medium-sized point clouds.

SplatNet (Su et al. 2018) is the work most closely related to ours. It alleviates the computational burden of 3D convolutions by using a sparse permutohedral lattice, performing convolutions only around the surfaces. It discretizes the space in uniform simplices and accumulates the features of the raw point cloud onto the vertices of the lattice using a splatting operation. Convolutions are applied on the lattice vertices and a slicing operation barycentrically interpolates the features of the vertices back onto the point cloud. A series of splat-conv-slice operations are applied to obtain contextual information. The main disadvantage is that splat and slice operations are not learned and repeated application slowly degrades the point clouds features as they act as Gaussian filters (Baek and Adams 2009). Furthermore, storing high-dimensional features for each point in the cloud is memory intensive which limits the maximum number of points that can be processed. In contrast, our approach has learned operations for splatting and slicing which brings more representational power to the network. We also restrict their usage to only the beginning and the end of the network, leaving the rest of the architecture fully convolutional.
Mesh networks The connectivity of triangular or quadrilateral mesh faces enables easy computation of normal vectors and establishes local tangent planes.

GCNN (Masci et al. 2015) operates on small local patches which are convolved using a series of rotated filters, followed by max-pooling to deal with the ambiguity in the patch orientation. However, the max-pooling disregards the orientation. MoNet (Monti et al. 2017) deals with the orientation ambiguity by aligning the kernels to the principal curvature of the surface. Yet, this does not solve cases in which the local curvature is not informative, e.g. for walls or ceilings. TextureNet (Huang et al. 2019) further improves on the idea by using a global 4-RoSy orientations field. This provides a

smooth orientation field at any point on the surface which is aligned to the edges of the mesh and has only a 4-direction ambiguity. Defining convolution on patches oriented according to the 4-RoSy field yields significantly improved results.
Graph networks These methods allow arbitrary topologies to connect vertices and lift the restriction of triangular or quadrilateral meshes.

Wang et al. (2018a) and Wu et al. (2019) define a convolution operator over non-grid structured data by having continuous values over the full vector space. The weights of these continuous filters are parametrized by an multi-layer perceptron (MLP).

Defferrard et al. (2016) formulate CNNs in the context of spectral graph theory. They define the convolution in the Fourier domain with Chebyshev polynomials to obtain fast localized filters. However, spectral approaches are not directly transferable to a new graph as the Fourier basis changes. Additionally, the learned filters are rotation invariant which can be seen as a limitation to the representational power of the network.
Multi-view networks The convolution operation is well defined in 2D and hence, there is an interest in casting 3D segmentation as a series of single-view segmentations which are fused together.

Pham et al. (2019a) simultaneously reconstruct the scene geometry and recover the semantics by segmenting sequences of RGB-D frames. The segmentation is transferred from 2D images to the 3D world and fused with previous segmentations. A CRF finally resolves noisy predictions.

Tatarchenko et al. (2018) assumes that the data is sampled from locally Euclidean surfaces and project the local surface geometry onto a tangent plane to which 2D convolutions can be applied. This requires a heavy preprocessing for normal calculation. In contrast, our approach can deal with raw point clouds without requiring normals.

2.2 Motion segmentation

For the task of motion segmentation two approaches have been widely used: Networks either incorporate multiple point clouds directly or accumulate a sequence of individually segmented point clouds.

Shi et al. (2020) present their U-Net based architecture SpSequenceNet for semantic segmentation on 4D point clouds. They input two point clouds and generate the output for the later one with a voxel-based method. They designed two modules, the Cross-frame Global Attention (CGA) and the Cross-frame Local Interpolation (CLI) module. The CGA acts as a teacher that uses the data from P_{t-1} to focus the network on the important features of P_t. The CLI module fuses information between both point clouds by combining the spatial and temporal information.

Kernel Point Convolution (KPConv) (Thomas et al. 2019) operates directly on the point clouds by facilitating convolution weights that are located in Euclidean space. Points in the vicinity of these kernels are weighted and summed together to feature vectors. KPConv (Thomas et al. 2019), DarkNet53Seg (Behley et al. 2019) and TangentConv (Tatarchenko et al. 2018) were previously used for the segmentation of 4D point clouds by accumulating multiple clouds of a sequence.

2.3 Instance segmentation

Researchers extended principles from 2D to obtain instances in 3D which can be roughly categorized in proposal-based and proposal-free methods.
Proposal-based This type solves the problem in two stages. The first network stage generates proposals of bounding boxes for the objects in the scene. A second stage performs foreground-background segmentation on the points within the bounding boxes in order to get valid instances.

Yang et al. (2019) present a single-stage method for instance segmentation that can train both the proposal and the point-mask prediction network in an end-to-end manner. Yi et al. (2019) alleviate some of the issues associated with wrong bounding box predictions by using an analysis-by-synthesis strategy.
Proposal-free Proposal-free methods tackle instance segmentation without the need of generating object proposals. They usually rely on predicting point embedding and apply clustering to recover the instances.

Many proposal-free approaches base their work on the 2D instance segmentation of De Brabandere et al. (2017) in which pixel embeddings are predicted. There, a discriminative loss encourages the embeddings that belong to the same instance to be clustered together while embeddings from different instances should be further apart.

SPGN (Wang et al. 2018b) learns a similarity matrix for all point pairs, based on which, similar points are merged to instances. VoteNet (Qi et al. 2019) uses a Hough voting mechanism where the points predict the offset towards the object center. A clustering algorithm finally recovers the object instances.

Neven et al. (2019) alleviate some of the issues associated with proposal-free methods by allowing also the clustering algorithm to be part of the training by jointly optimizing the spatial embeddings and the clustering bandwidth.

Wang et al. (2019) proposed a framework that allows for semantic and instances to be predicted simultaneously and for the two tasks to mutually benefit from each other. Similarly, Pham et al. (2019b) recover both instances and semantics and apply a CRF to improve the predictions accuracy.

Most of these works utilize a PointNet (Qi et al. 2017a) or PointNet++ (Qi et al. 2017b) network to predict the point

embeddings. In our case, we extend LatticeNet in a similar manner to other proposal-free methods but predict the embeddings using the lattice convolutions.

3 Notation

Throughout this paper, we use bold upper-case characters to denote matrices and bold lower-case characters to denote vectors.

The vertices of the d-dimensional permutohedral lattice are defined as a tuple $v = (\mathbf{c}_v, \mathbf{x}_v)$, with $\mathbf{c}_v \in \mathbb{Z}^{(d+1)}$ denoting the coordinates of the vertex and $\mathbf{x}_v \in \mathbb{R}^{v_d}$ representing the values stored at vertex v. The full lattice containing n vertices is denoted with $V = (\mathbf{C}, \mathbf{X})$, with $\mathbf{C} \in \mathbb{Z}^{n\times(d+1)}$ representing the coordinate matrix and $\mathbf{X} \in \mathbb{R}^{n\times v_d}$ the value matrix.

The points in a cloud are defined as a tuple $p = (\mathbf{g}_p, \mathbf{f}_p)$, with $\mathbf{g}_p \in \mathbb{R}^d$ denoting the coordinates of the point and $\mathbf{f}_p \in \mathbb{R}^{f_d}$ representing the features stored at point p (color, normals, etc.). The full point cloud containing m points is denoted by $P = (\mathbf{G}, \mathbf{F})$ with $\mathbf{G} \in \mathbb{R}^{m\times d}$ being the positions matrix and $\mathbf{F} \in \mathbb{R}^{m\times f_d}$ the feature matrix. The feature matrix $\mathbf{F}$ can also be empty in which case f_d is set to zero.

For motion segmentation we define a sequence of point clouds as $P_{seq} = (P_0, P_1, \ldots, P_n)$ with $P_n = (\mathbf{G}, \mathbf{F})$. We define a timestep as processing one cloud of this sequence.

We denote with I_p the set of lattice vertices of the simplex that contains point p. The set I_p always contains $d+1$ vertices as the lattice tessellates the space in uniform simplices with $d+1$ vertices each. Furthermore, we denote with J_v the set of points p for which vertex v is one of the vertices of the containing simplices. Hence, these are the points that contribute to vertex v through the splat operation.

We denote with $\mathcal{S}$ the splatting operation, with $\mathcal{Y}$ the slicing operation, with $\tilde{\mathcal{Y}}$ the deformable slicing, with $\mathcal{P}$ the PointNet module, with $\mathcal{D}_G$ and $\mathcal{D}_F$ the distribution of the point positions and the points features, respectively, and with $\mathcal{G}$ the gathering operation.

4 Permutohedral lattice

The d-dimensional permutohedral lattice is formed by projecting the scaled regular grid $(d+1)\mathbb{Z}^{d+1}$ along the vector $\mathbf{1} = [1, \ldots, 1]$ onto the hyperplane H_d: $\mathbf{p} \cdot \mathbf{1} = 0$.

The lattice tessellates the space into uniform d-dimensional simplices. Hence, for $d = 2$ the space is tessellated with triangles and for $d = 3$ into tetrahedra. The enclosing simplex of any point can be found by a simple rounding algorithm (Baek and Adams 2009).

Due to the scaling and projection of the regular grid, the coordinates $\mathbf{c}_v$ of each lattice vertex sum up to zero. Each vertex has $2(d+1)$ immediate neighboring vertices. The coordinates of these neighbors are separated by a vector of form $\pm[-1, \ldots, -1, d, -1, \ldots, -1] \in \mathbb{Z}^{d+1}$.

The vertices of the permutohedral lattice are stored in a sparse manner using a hash map in which the key is the coordinate $\mathbf{c}_v$ and the value is $\mathbf{x}_v$. Hence, we only allocate the simplices that contain the 3D surface of interest. This sparse allocation allows for efficient implementation of all typical operations in CNNs (convolution, pooling, transposed convolution, etc.).

The permutohedral lattice has several advantages w.r.t. standard cubic voxels. The number of vertices for each simplex is given by $d+1$ which scales linearly with increasing dimension, in contrast to the 2^d for standard voxels. This small number of vertices per simplex allows for fast splatting and slicing operations. Furthermore, splatting and slicing create piece-wise linear outputs as they use barycentric interpolation. In contrast, standard quantization in cubic voxels create piece-wise constant outputs, leading to discretization artefacts.

Spatial correspondences between lattice vertices are given by design and the hashmap: If the hashmap stays the same for the whole sequence, spatially identical lattice vertices of different point clouds are always mapped to the same entries. This is visualized in Fig. 9 where features from two different time-steps are fused together.

5 Method

The input to our method is a point cloud $P = (\mathbf{G}, \mathbf{F})$ containing coordinates and per-point features.

We define the scale of the lattice by scaling the positions $\mathbf{G}$ as $\mathbf{G}_s = \mathbf{G}/\boldsymbol{\sigma}$, where $\boldsymbol{\sigma} \in \mathbb{R}^d$ is the scaling factor. The higher the sigma the less number of vertices will be needed to cover the point cloud and the coarser the lattice will be. For ease of notation, unless otherwise specified, we refer to $\mathbf{G}_s$ as $\mathbf{G}$ as we usually only need the scaled version.

5.1 Common operations on permutohedral lattice

In this section, we will explain in detail the standard operations on a permutohedral lattice that are used in previous works (Su et al. 2018; Gu et al. 2019).

Splatting refers to the interpolation of point features onto the values of the lattice V using barycentric weighting (Fig. 3a). Each point splats onto $d+1$ lattice vertices and their weighted features are summed onto the vertices.

Convolving operates analogously to standard spatial convolutions in 2D or 3D, i.e. a weighted sum of the vertex values together with its neighbors is computed. We use convolutions that span over the 1-hop ring around a vertex and hence convolve the values of $2(d+1)+1$ vertices (Fig. 2).

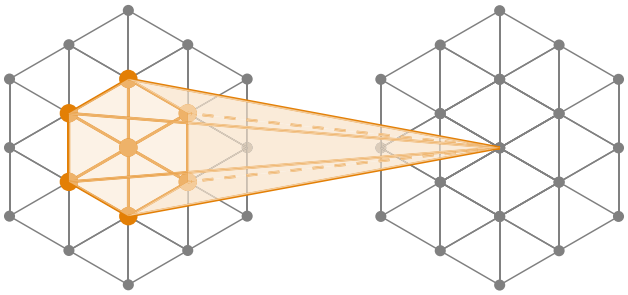


Fig. 2 Convolution: The neighboring vertices of a lattice are convolved similarly to standard 2D convolutions. If a neighbor is not allocated in the sparse structure, we assume that it has a value of zero

Slicing is the inverse operation to splatting. The vertex values of the lattice are interpolated back for each position with the same weights used during splatting. The weighted contributions from the simplexes $d+1$ vertices are summed up (Fig. 5a).

5.2 Proposed operations on permutohedral lattice

The operations defined in Sect. 5.1 are typically used in a cascade of splat-conv-slice to obtain dense predictions (Su et al. 2018). However, splatting and slicing act as Gaussian kernel low-pass filtering on encoded information (Baek and Adams 2009). Their repeated usage at every layer is detrimental to the accuracy of the network. Additionally, splatting acts as a weighted average on the feature vectors where the weights are only determined through barycentric interpolation. Including the weights as trainable parameter allows the network to decide on a better interpolation scheme. Furthermore, as the network grows deeper and feature vectors become higher-dimensional, slicing consumes increasingly more memory, as it assigns the features to the points. Since in most cases $|P| \gg |V|$, it is more efficient to store the features only in the lattice vertices.

To address these limitations, we propose four new operators on the permutohedral lattice which are more suitable for CNNs and dense prediction tasks.

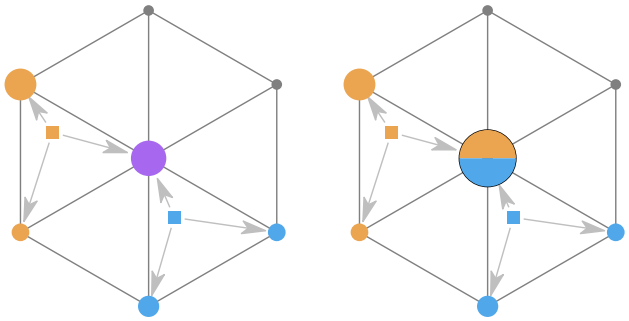
Distribute is defined as the list of features that each lattice vertex receives. However, they are not summed as done by splatting:

$$\mathbf{x}_v = \mathcal{S}(P, V) = \sum_{p \in J_v} b_{pv} \mathbf{f}_p, \tag{1}$$

where $\mathbf{x}_v$ is the value of lattice vertex v and b_{pv} is the barycentric weight between point p and lattice vertex v.

Instead, our distribute operators $\mathcal{D}_G$ and $\mathcal{D}_F$ concatenate coordinates and features of the contributing points:

$$\mathbf{x}_v = \mathcal{P}(\mathbf{D}_{v_g}; \mathbf{D}_{v_f}), \tag{2}$$



(a) Splat **(b)** Distribute

Fig. 3 Splat and Distribute operations: Splatting uses barycentric weighting to add the features of points onto neighboring vertices. The naïve summation can be detrimental to the network as splatting acts as a Gaussian filter. Distributing stores all the features of the contributing points, causing no loss of information and allows further processing by the network

$$\mathbf{D}_{v_g} = \mathcal{D}_G(P, V) = \{\, \mathbf{g}_p - \boldsymbol{\mu}_v \mid p \in J_v \,\}, \tag{3}$$

$$\mathbf{D}_{v_f} = \mathcal{D}_F(P, V) = \{\, \mathbf{f}_p \mid p \in J_v \,\}, \tag{4}$$

$$\boldsymbol{\mu}_v = \frac{1}{|J_v|} \sum_{p \in J_v} \mathbf{g}_p, \tag{5}$$

where $\mathbf{D}_{v_g} \in \mathbb{R}^{|J_v| \times d}$ and $\mathbf{D}_{v_f} \in \mathbb{R}^{|J_v| \times f_d}$ are matrices containing the distributed coordinates and features, respectively, for the contributing points into a vertex v. The matrices are concatenated and processed by a PointNet $\mathcal{P}$ to obtain the final vertex value $\mathbf{x}_v$. Fig. 3 illustrates the difference between splatting and distributing.

Note that we use a different distribute function for coordinates then for point features. For coordinates, we subtract the mean of the contributing coordinates. The intuition behind this is that coordinates by themselves are not very informative w.r.t. the potential semantic class. However, the local distribution is more informative as it gives a notion of the geometry.

Downsampling refers to a coarsening of the lattice, by reducing the number of vertices. This allows the network to capture more contextual information. Downsampling consists of two steps: creation of a coarse lattice and obtaining its values. Coarse lattices are created by repeatedly dividing the point cloud positions by 2 and using them to create new lattice vertices (Barron et al. 2015). The values of the coarse lattice are obtained by convolving over the finer lattice from the previous level (Fig. 4). Hence, we must embed the coarse lattice inside the finer one by scaling the coarse vertices by 2. Afterwards, the neighbors vertices over which we convolve are separated by a vector of form $\pm[-1, \ldots, -1, d, -1, \ldots, -1] \in \mathbb{Z}^{d+1}$. The downsampling operation effectively performs a strided convolution.

Upsampling follows a similar reasoning. The fine vertices need first to be embedded in the coarse lattice using a

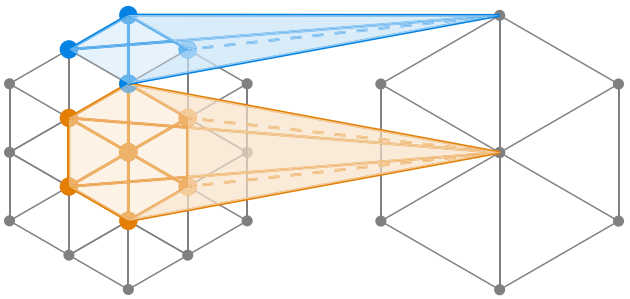


Fig. 4 Coarsen: Downsampling of the lattice is performed by embedding the coarse lattice in the finer one and convolving over the neighbors. This effectively performs a strided convolution. Transposed convolution is performed in an analogous manner by embedding a fine lattice into a coarse one

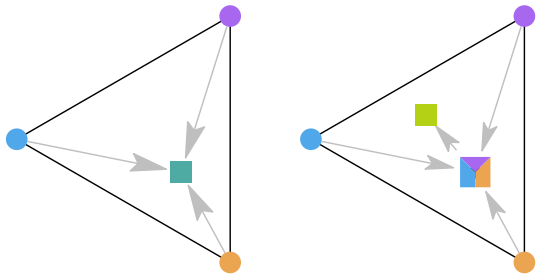


Fig. 5 Slice and DeformSlice: Slicing barycentrically interpolates the vertex values back onto a point. DeformSlice allows for the network to directly affect the interpolated value by learning offsets of the barycentric coordinates

division by 2. Afterwards, the neighboring vertices over which we convolve are separated by a vector of form $\pm[-0.5, \ldots, -0.5, d/2, -0.5, \ldots, -0.5]$. The careful reader will notice that in this case, the coordinates of the neighboring vertices may not be integer anymore; they may have a fractional part and will, therefore, lie in the middle of a coarser simplex. In this case we ignore the contribution of this neighboring vertices and only take the contribution of the center vertex. The upsampling operation effectively performs a transposed convolution.

DeformSlicing While the slicing operation $\mathcal{Y}$ barycentrically interpolates the values back to the points by using barycentric coordinates:

$$f_p = \mathcal{Y}(P, V) = \sum_{v \in I_p} b_{pv} \mathbf{x}_v, \tag{6}$$

we propose the DeformSlicing $\tilde{\mathcal{Y}}$ which allows the network to directly modify the barycentric coordinates and shift the position within the simplex for data-dependent interpolation:

$$f_p = \tilde{\mathcal{Y}}(P, V) = \sum_{v \in I_p} (b_{pv} + \Delta b_{pv}) \mathbf{x}_v. \tag{7}$$

Here, Δb_{pv} are offsets that are applied to the original barycentric coordinates. A parallel branch within our network first gathers the values from all the vertices in a simplex and regresses the Δb_{pv}:

$$\mathbf{q}_p = \mathcal{G}(P, V) = \{\, b_{pv} \mathbf{x}_v \mid v \in I_p \,\}, \tag{8}$$

$$\Delta \mathbf{b}_p = \mathcal{F}(\mathbf{q}_p), \tag{9}$$

where $\mathbf{q}_p$ is a set containing the weighted values of all the vertices of the simplex containing p and the prediction $\Delta \mathbf{b}_p = \{\, \Delta b_{pv} \mid v \in I_p \,\}$ is a set of offsets to the barycentric coordinates towards the $d+1$ vertices. With a slight abuse of notation—due to the fact that the vertices of a simplex are always enumerated in a consistent manner, we can regard $\mathbf{b}_p$ and $\mathbf{q}_p$ as vectors in $\mathbb{R}^{(d+1)}$ and $\mathbb{R}^{(d+1)v_d}$, respectively, and cast the prediction of offsets as a fully connected layer followed by a non-linearity:

$$\Delta \mathbf{b}_p = \mathcal{F}(\mathbf{q}_p) = \sigma(\mathbf{q}_p \cdot \mathbf{W} + b). \tag{10}$$

However, this prediction has the disadvantage of not being permutation equivariant; therefore, permutation of the vertices would not imply the same permutation in the barycentric offsets:

$$\mathcal{F}(\pi \mathbf{q}_p) \neq \pi \mathcal{F}(\mathbf{q}_p), \tag{11}$$

where π is the set of all permutations of the $d+1$ vertices.

It is important for our prediction to be permutation equivariant because the vertices may be arranged in any order and the barycentric offsets need to keep a consistent preference towards a certain vertexes' features, regardless of its position within a simplex.

In order for the prediction of the offsets to be consistent with permutations of the vertices, we take inspiration from the work of Ravanbakhsh et al. (2016) and Zaheer et al. (2017) of equivariant layers and design $\mathcal{F}$ as:

$$\Delta b_{pv} = \sigma(b + (b_{pv} \mathbf{x}_v - \max_{d \in I_p} \{b_{pd} \mathbf{x}_d\}) \cdot \mathbf{W}), \tag{12}$$

$$\Delta \mathbf{b}_p = \mathcal{F}(\mathbf{q}_p) = \{\, \Delta b_{pv} \mid v \in I_p \,\}, \tag{13}$$

where $\mathbf{W} \in \mathbb{R}^{v_d \times 1}$ is a weight matrix and $b \in \mathbb{R}$ corresponds to a scalar bias. In other words, we subtract from each weighted vertex the maximum of the weighted values of all the other vertices in the simplex. Since the max operation is invariant to permutations of the input, the regression of the offsets is *equivariant* to permutations of the vertices.

The difference between the slicing and our DeformSlicing is visualized in Fig. 5

6 Segmentation methods

Due to the flexibility of LatticeNet various segmentation methods can be implemented. In this section, we detail the methods used for each one.

6.1 Semantic segmentation

Semantic segmentation uses the default U-Net architecture described in the Network Architecture section. It is trained with an equal part combination of cross entropy loss and Lovász loss (Berman et al. 2018). The Lovász loss acts as a surrogate for the intersection-over-union score and is especially useful for dealing with class imbalance.

6.2 Instance segmentation

Our instance segmentation network follows the work of other proposal-free methods like (De Brabandere et al. 2017). We use LatticeNet to predict for each 3D point p_i in the point cloud an embedding x_i. A discriminative loss encourages closeness in embeddings space for points of the same instance while promoting distance between different instances. Finally, we apply mean-shift clustering on the points in embeddings space. Points belonging to the same cluster are defined as an Instances.

This discriminative loss can be expressed with three terms:

- Variance term: The intra-cluster pull force that draws the embeddings towards the mean embedding.
- Distance term: An inter-cluster push force that forces the clusters to be far apart from each other in embedding space.
- Regularization term: A small force that pulls the cluster centers towards the origin in order to keep the activations bounded.

The full loss is then defined as:

$$L_{var} = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{N_c}\sum_{i=1}^{N_c}[\|\mu_c - x_i\| - \delta_v]^2 \tag{14}$$

$$L_{dist} = \frac{1}{C(C-1)}\sum_{\substack{c_A=1\\ c_A\neq c_B}}^{C}\sum_{c_B=1}^{C}\left[2\delta_{\mathrm{d}} - \|\mu_{c_A} - \mu_{c_B}\|\right]_+^2 \tag{15}$$

$$L_{reg} = \frac{1}{C}\sum_{c=1}^{C}\|\mu_c\| \tag{16}$$

$$L = \alpha \cdot L_{var} + \beta \cdot L_{dist} + \gamma \cdot L_{reg} \tag{17}$$

We define C as the number of clusters in the ground truth, N_c as the number of elements in cluster c, x_i as the embedding vector for point p_i and μ_c as the mean or cluster center for cluster c. The δ_{v} and δ_{d} are the margins for the variance and distance loss respectively. We set $\alpha = \beta = 1$ and $\gamma = 0.001$

A visualization of the pipeline for instance segmentation can be seen in Fig. 6.

6.3 Motion segmentation

Motion segmentation distinguishes between dynamic and static objects within a point cloud. For this, the network needs temporal information. We extend the original LatticeNet U-Net architecture with a recursive architecture that can process a sequence of point clouds P_{seq} at times $t, t-1, \ldots, t-n$ and learn to distinguish for example between a moving car and a parked car.

The dynamic objects are considered as additional classes. Hence, we use the same loss as in the case of semantic segmentation. We also explore multiple ways to perform the fusion of temporal information which we detail in the Network Architecture section.

7 Network architecture

Input to our network is a point cloud P which may contain per-point features stored in $\mathbf{F}$. The output is class probabilities for each point p. In the recurrent network the input is an ordered set of point clouds P_{seq} and the output are class probabilities for the last point cloud of the sequence. Moving and static objects are considered as different semantic classes.

Our network architecture has a U-Net structure (Ronneberger et al. 2015) and is visualized in Fig. 7 together with the used individual blocks.

The first layers distribute the point features onto the lattice and use a PointNet to obtain local features. Afterwards, a series of ResNet blocks (He et al. 2016a), followed by repeated downsampling, aggregates global context. The decoder branch mirrors the encoder architecture and upsamples through transposed convolutions. Finally, a DeformSlicing propagates lattice features onto the original point cloud. Skip connections are added by concatenating the encoder feature maps with matching decoder features.

7.1 Temporal fusion

Incorporating temporal information for motion prediction over a sequence of point clouds relies on fusing information between multiple time-steps. For this purpose, the feature vectors of the timesteps $t-1$ and t are passed through a Temporal Fusion block, as shown in Fig. 8. This fusion consists of a concatenation of both feature vectors and a linear layer followed by a non-linearity (Fig. 9). Each new timestep allocates additional vertices in the lattice corresponding

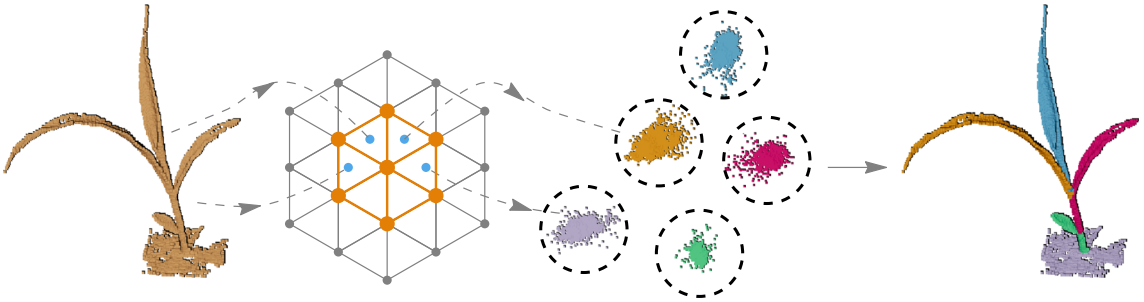


Fig. 6 Instance segmentation: LatticeNet takes raw point clouds as input and embeds them into a sparse lattice where convolutions are applied. Features on the lattice are projected onto a 2D space where clustering is performed. The clusters define the instances of each object type in the original cloud

to newly explored areas in the map. For correct fusion, the features from the previous time-step need to be zero-padded so that the sizes match.

Additionally, we performed experiments with a single Temporal Fusion block in the network and max-pooling over both feature vectors instead of the linear layer, but found that three Temporal Fusion blocks achieved overall superior results (Fig. 10).

It should be noted that our approach for temporal fusion relies on a sequence of clouds that are transformed into a common coordinate frame. The required scan poses for transformation can be obtained e.g. from GPS or SLAM.

8 Implementation

Our lattice is stored sparsely on a hash map structure, which allows for fast access of neighboring vertices. Unlike (Su et al. 2018), we construct the hash map directly on the GPU, saving us from incurring an expensive CPU to GPU memory copy.

For memory savings, we implemented the DeformSlice and the last linear classification layer in one fused operation, avoiding the storage of high-dimensional feature vectors for each point in the point cloud.

All of the lattice operators containing forwards and backwards passes are implemented on the GPU and exposed to PyTorch (Paszke et al. 2017).

Following recent works (He et al. 2016b; Huang et al. 2017), all convolutions are pre-activated using Group Normalization (Wu and He 2018) and a ReLU unit. We chose Group Normalization instead of the standard batch normalization due to greater stability for small batch sizes. We use the default of 32 groups.

The models were trained using the Adam optimizer with a learning rate of 0.001 and a weight decay of $10^{-}4$. The learning rate was reduced by a factor of 10 when the loss plateaued.

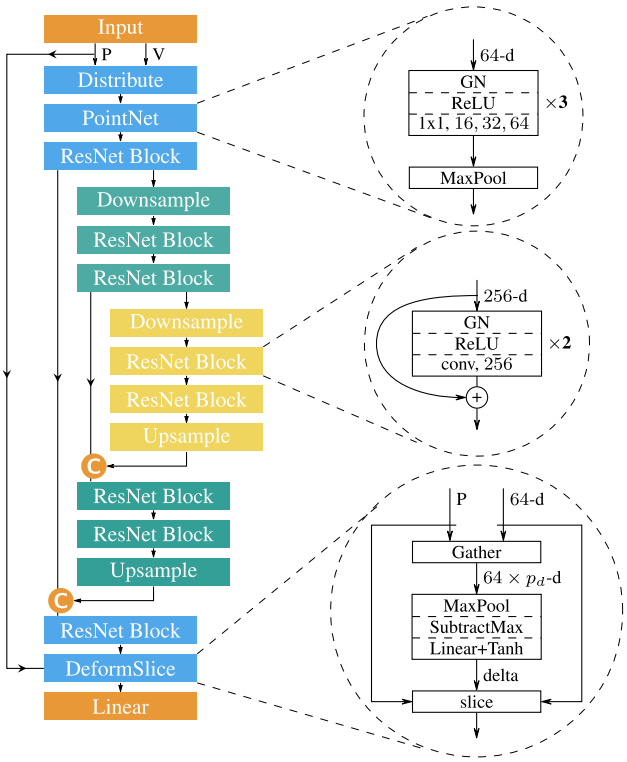


Fig. 7 Architecture: Our model follows a U-Net structure. For ease of representation, blocks which are repeated one after another are indicated with a multiplier on the right side of the operation

We share the PyTorch implementation of LatticeNet at https://github.com/AIS-Bonn/lattice_net.

9 Experiments

We evaluate our proposed lattice network on four different datasets: ShapeNet (Yi et al. 2016), ScanNet (Dai et al. 2017), SemanticKITTI (Behley et al. 2019) and Pheno4D (https://www.ipb.uni-bonn.de/data/pheno4d/). For the task of semantic segmentation and motion segmentation we report the mean Intersection-over-Union (mIoU). For the

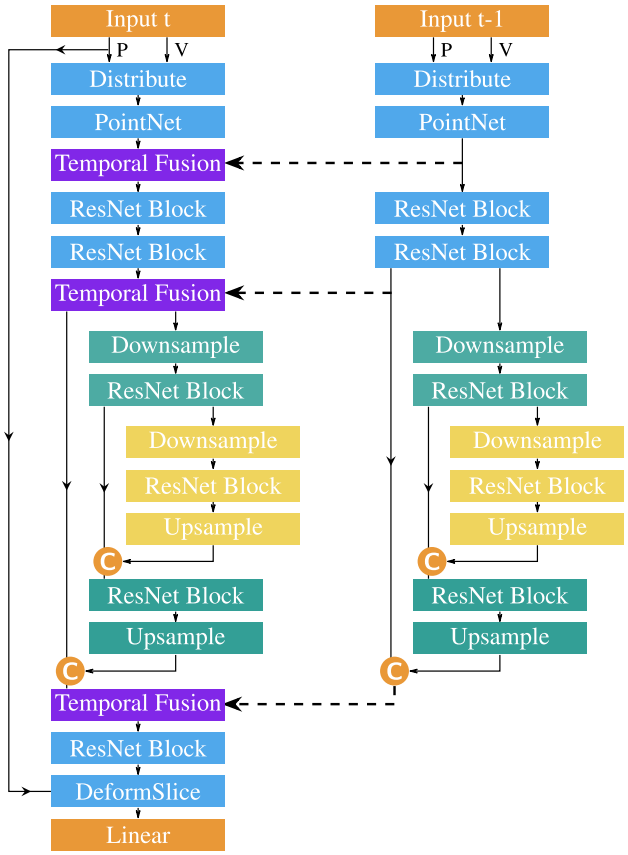


Fig. 8 Recurrent architecture: The features from previous time-steps are fused in the current time-step at multiple levels of the network. This allows the network to distinguish dynamic objects from static ones

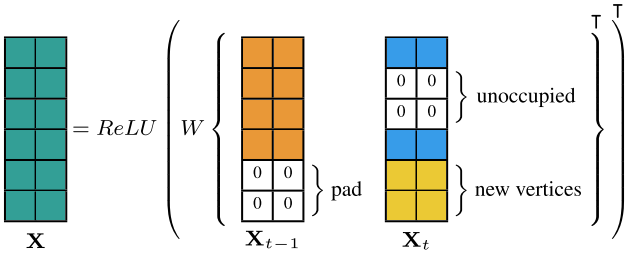


Fig. 9 Temporal fusion: The features from the previous time-step are zero-padded in order to account for the new vertices that were allocated at the current time-step. The features are afterwards concatenated and passed through a linear layer followed by a non-linearity

task of instance segmentation, we report the Symmetric Best Dice (SBD) (De Brabandere et al. 2017). SBD measures the accuracy of the instance segmentation by averaging for each input label the ground truth label yielding the maximum Dice score.

We use a shallow model for ShapeNet and Pheno4D and a deeper model for ScanNet and SemanticKITTI as the datasets are larger. We augment all data using random mirroring and translations in space. For ScanNet, we also apply random color jitter. A video with additional footage of the experiments is available online [1].

9.1 Evaluation of segmentation accuracy

ShapeNet part segmentation is a subset of the ShapeNet dataset (Yi et al. 2016) which contains objects from 16 different categories each segmented into 2–6 parts. The dataset consists of points sampled from the surface of the objects, together with the ground truth label of the corresponding object part. The objects have an average of 2613 points. We train and evaluate our network on each object individually. We use the official train/test splits as defined by the dataset containing a total of 12 137 training objects and 2874 test objects. The results for our and five competing methods are gathered in Table 1 and visualized in Fig. 11.

We observe that for some classes, we obtain state-of-the-art performance and for other objects, the IoU is slightly lower than for other approaches. We ascribe this to the fact that training one fixed architecture size for each individual object is suboptimal as some objects like the ”cap” have as few as 55 examples while others like the table have more than 5K. This causes the network to be prone to overfitting on the easy object or underfitting on the difficult ones. A fair evaluation would require finding an architecture that performs well for all objects on average. However, due to various issues with mislabeled ground truths (Su et al. 2018) we deem that experimentation with more architectures or with different regularization strengths for individual objects would overfit the dataset.

ScanNet 3D segmentation Dai et al. (2017) consists of 3D reconstructions of real rooms. It contains ≈ 1500 rooms segmented into 20 classes (bed, furniture, wall, etc.). The rooms have between 9K and 537K points—on average 145K. We segment an entire room at once without cropping. We use the official train/test splits as defined by the dataset containing a total of 1201 training rooms and 100 test objects. Results are gathered in Table 2 and visualized in Fig. 12. We obtain an IoU of 64.0 which is significantly higher than the most similar related work of SplatNet. It is to be noted that MinkowskiNet achieves a higher IoU but at the expense of an extremely high spatial resolution of 2 cm per voxel. In contrast, our approach allocates lattice vertices so that each vertex covers approximately 30 points. On this dataset, this corresponds to a spatial extent of approximately 10 cm.

SemanticKITTI Behley et al. (2019) consists of semantically annotated LiDAR scans of real urban environments. The annotation covers a total of 19 classes for single scan evaluation and a total of 25 classes for multiple scan evaluation. Each scan contains between 82K and 129K points. We process each scan entirely without any cropping. We use the

[1] http://www.ais.uni-bonn.de/videos/RSS_2020_Rosu/.

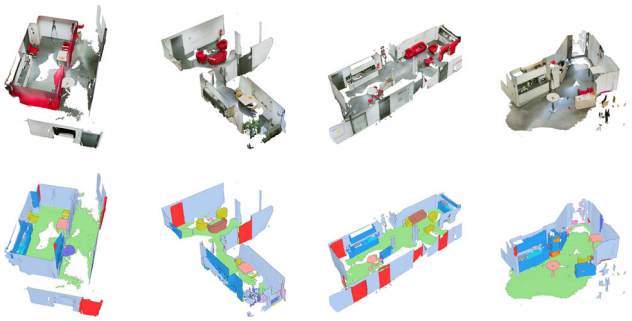


Fig. 10 Bonn Activity Maps segmentations. Colored meshes are reconstructed from KinectV2 data using volumetric integration (Nießner et al. 2013; Stotko et al. 2019) and semantically segmented using LatticeNet. Color coding of semantic labels corresponds to the ScanNet dataset (Dai et al. 2017)

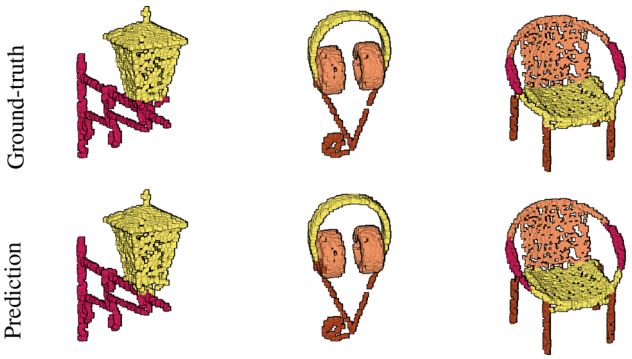


Fig. 11 ShapeNet (Yi et al. 2016) results of our method

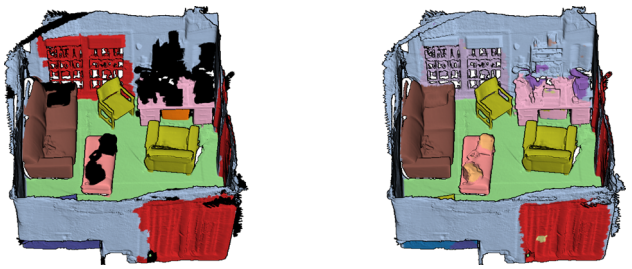


Fig. 12 ScanNet results. The left image shows the ground truth and the right one our prediction

official train/validation splits as defined by the dataset. The test set is not publicly available and testing can only be done through the benchmark server.

The results for single scan are provided in Table 3 and visualized in Fig. 13. Our LatticeNet outperforms all other methods—in case of the most similar SplatNet by more than a factor of two. It is to be noted that DarkNet53Seg (Behley et al. 2019), DarkNet21Seg (Behley et al. 2019) and SqueezeSegV2 (Wu et al. 2018) are methods that operate on a 2D image by wrapping the LiDAR scans to 2D using spherical coordinates. In contrast, our method can operate on general point clouds, directly in 3D.

For motion segmentation we take as input three point clouds at consecutive time steps and output the segmentation for the final, most recent cloud. We overlap this time window so that every clouds gets to be segmented. For the first few clouds, the time window is reduced as there are no clouds from previous time-steps to give as input. The results for the motion segmentation are provided in Table 4 and visualized in Fig. 14.

We observe that for motion segmentation we outperform other approaches except for KPConv (Thomas et al. 2019), which has higher IoU. However, it is to be noted that KPconv cannot process a full point cloud at once due to memory constraints and rather processes sub-clouds centered around random spheres in the scene. The spheres are chosen randomly in the scene to ensure each point is tested multiple times by different sphere locations. Finally, a voting scheme gives the final prediction. In contrast, our approach can process a full point cloud without requiring neighborhood searching or partitioning in sub-clouds.

Bonn Activity Maps (Tanke et al. 2019) is a dataset for human tracking, activity recognition and anticipation of multiple persons. It contains annotations of persons, their trajectories and activities. The 3D reconstruction of the four kitchen scenarios is however of more interest to us. The environments are reconstructed as 3D colored meshes and have no ground truth semantic annotations. We trained our LatticeNet on the ScanNet dataset and evaluate it on the 4 kitchens in order to provide an annotation for each vertex of the mesh. The results are shown in Fig. 10. We can observe that our network generalizes well to unseen datasets, recorded with different sensors and with different noise properties as the semantic segmentations look plausible and exhibit sharp borders between classes.

Pheno4D https://www.ipb.uni-bonn.de/data/pheno4d/ is a spatio-temporal dataset of point clouds of maize and tomato plants with instance annotations of leaves. We use a shallow version of LatticeNet to compute per-point embeddings and cluster them using mean-shift to recover the instances. We compare with PointNet and PointNet++ as they are popular methods for computing per-point embeddings. Since the dataset contains 7 maize and 7 tomato plants, we train on the first 5 plants for each type and test on the remaining two. The results are gathered in Table 5. We observe that our method is capable of computing more meaningful embeddings that create more distinctive clusters between each plant organ.

9.2 Ablation studies

We perform various ablations regarding our contribution to judge how much they affect the network's performance.

DeformSlice We assess the impact that DeformSlice has on the network by comparing it with the Slice operator which

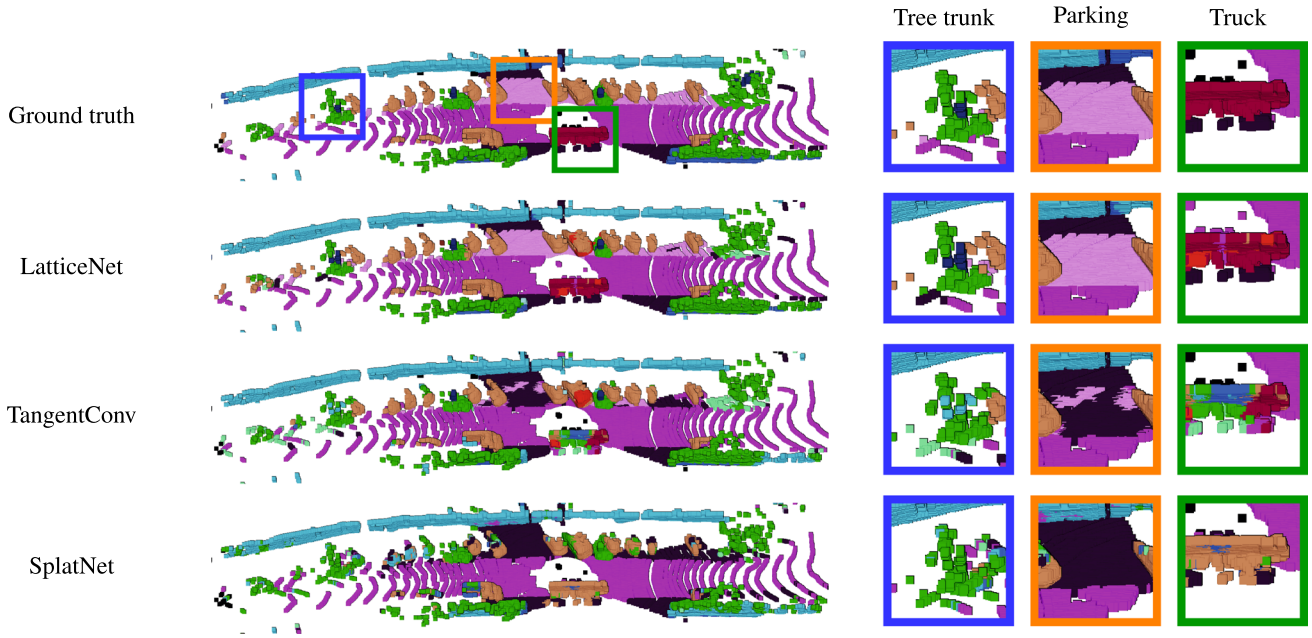


Fig. 13 SemanticKITTI results. We compare the prediction from our LatticeNet with the results from TangentConv (Tatarchenko et al. 2018) and SplatNet (Su et al. 2018). We can observe that our approach can better learn small objects like tree trunks, despite their relatively small number of points. Additionally, the network also effectively makes use of contextual information in order to correctly predict the parking place due to the existence of nearby cars

Table 1 Results on ShapeNet part segmentation (Yi et al. 2016)

#Instances	Instance avg.	2690 Air-plane	76 Bag	55 Cap	898 Car	3758 Chair	69 Ear-phone	787 Guitar	392 Knife	1547 Lamp	451 Laptop	202 Motor-bike	184 Mug	283 Pistol	66 Rocket	152 Skate-board	5271 Table
PointNet (Qi et al. 2017a)	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++ (Qi et al. 2017b)	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
SplatNet 3D (Su et al. 2018)	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
SplatNet 2D-3D (Su et al. 2018)	**85.4**	83.2	84.3	**89.1**	80.3	90.7	**75.5**	92.1	87.1	83.9	96.3	75.6	95.8	83.8	64.0	75.5	81.8
FCPN (Rethage et al. 2018)	84.0	**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
Ours	83.9	82.3	**84.8**	79.1	81.0	86.9	71.0	91.9	**89.4**	**84.7**	96.6	77.2	**95.8**	86.0	**70.5**	79.3	**87.0**

Bold signifies the highest mean intersection-over-union (mIoU)

Table 2 Results on ScanNet (Dai et al. 2017)

Method	mIOU
PointNet++ (Qi et al. 2017b)	33.9
SplatNet (Su et al. 2018)	39.3
TangetConv (Tatarchenko et al. 2018)	43.8
3DMV[‡] (Dai and Nießner 2018)	48.4
MinkowskiNet42 (5cm) (Choy et al. 2019)	67.9
SparseConvNet (Graham et al. 2018)[†]	72.5
MinkowskiNet42 (2cm) (Choy et al. 2019)[†]	**73.4**
Ours	64.0

Bold signifies the highest mean intersection-over-union (mIoU)

does not use learned barycentric interpolation. We evaluate this on SemanticKITTI, the largest dataset that we are using.

We also evaluate a version of DeformSlice which ensures that the new barycentric coordinates still sum up to one by adding an additional loss term:

$$L = \frac{1}{|P|} \sum_{p \in P} \left(\sum_{v \in I_p} \Delta b_{pv} \right)^2 . \tag{18}$$

However, we observe little change after adding this regularization term and hence, use the default version of DeformSlice for the rest of the experiments. The results are gathered in Table 6.

Distribute and PointNet Another contribution of our work is the usage of a Distribute operator to provide values to the lattice vertices which are later embedded in a higher-dimensional space by a PointNet-like architecture. The positions and features of the point cloud are treated sep-

Table 3 Results on SemanticKITTI (Behley et al. 2019)

Approach	mIoU	Road	Sidewalk	Parking	Other-ground	Building	Car	Truck	Bicycle	Motorcycle	Other-vehicle	Vegetation	Trunk	Terrain	Person	Bicyclist	Motor-cyclist	Fence	Pole	Traffic sign
PointNet (Qi et al. 2017a)	14.6	61.6	35.7	15.8	1.4	41.4	46.3	0.1	1.3	0.3	0.8	31.0	4.6	17.6	0.2	0.2	0.0	12.9	2.4	3.7
SplatNet (Su et al. 2018)	18.4	64.6	39.1	0.4	0.0	58.3	58.2	0.0	0.0	0.0	0.0	71.1	9.9	19.3	0.0	0.0	0.0	23.1	5.6	0.0
PointNet++ (Qi et al. 2017b)	20.1	72.0	41.8	18.7	5.6	62.3	53.7	0.9	1.9	0.2	0.2	46.5	13.8	30.0	0.9	1.0	0.0	16.9	6.0	8.9
Minkowski34(25cm) (Choy et al. 2019)	33.0	80.8	43.0	36.9	0.5	73.5	83.0	**42.9**	2.0	2.9	7.8	74.4	42.9	36.7	11.2	22.8	4.4	37.2	35.4	28.6
SqueezeSegV2 (Wu et al. 2018)	39.7	88.6	67.6	45.8	17.7	73.7	81.8	13.4	18.5	17.9	14.0	71.8	35.8	60.2	20.1	25.1	3.9	41.1	20.2	36.3
TangentConv (Tatarchenko et al. 2018)	40.9	83.9	63.9	33.4	15.4	83.4	90.8	15.2	2.7	16.5	12.1	79.5	49.3	58.1	23.0	28.4	8.1	49.0	35.8	28.5
DarkNet21Seg (Behley et al. 2019)	47.4	91.4	74.0	57.0	26.4	81.9	85.4	18.6	**26.2**	26.5	15.6	77.6	48.4	63.6	31.8	33.6	4.0	52.3	36.0	50.0
DarkNet53Seg (Behley et al. 2019)	49.9	**91.8**	**74.6**	**64.8**	**27.9**	84.1	86.4	25.5	24.5	**32.7**	**22.6**	78.3	50.1	**64.0**	**36.2**	33.6	4.7	55.0	38.9	**52.2**
Ours	**52.9**	90.0	74.1	59.4	22.0	**88.2**	**92.9**	26.6	16.6	22.2	21.4	**81.7**	**63.6**	63.1	35.6	**43.0**	**46.0**	**58.8**	**51.9**	48.4

Bold signifies the highest mean intersection-over-union (mIoU)

Table 4 Motion segmentation IoU results on SemanticKITTI (Behley et al. 2019) using a sequence of multiple past scans (in %)

Approach	car	truck	other-vehicle	person	bicyclist	motorcyclist	mIoU
TangentConv [41]	84.9	21.1	18.5	1.6	0.0	0.0	34.1
	40.3	42.2	30.1	6.4	1.1	1.9	
DarkNet53Seg [4]	84.1	20.0	20.7	7.5	0.0	0.0	41.6
	61.5	37.8	28.9	15.2	14.1	0.2	
SpSequenceNet [37]	88.5	29.2	22.7	6.3	0.0	0.0	43.1
	53.2	0.1	2.3	26.2	41.2	36.2	
KPConv [43]	93.7	70.3	38.6	21.6	0.0	0.0	51.2
	69.4	5.8	4.7	67.5	67.4	47.2	
Ours	91.1	65.4	23.1	6.8	0.0	0.0	45.2
	54.8	3.5	0.6	49.9	44.6	64.3	

Shaded cells correspond to the IoU of the moving classes, while unshaded entries are the non-moving classes

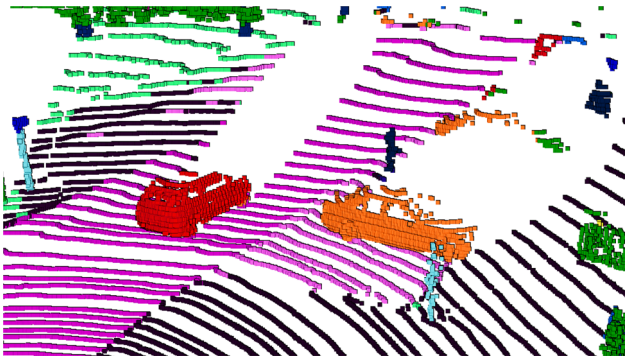


Fig. 14 Motion segmentation results on SemanticKITTI. The moving car on the road (red) is correctly distinguished from the parked car (orange) (Color figure online)

arately where the features (normals, color) are distributed directly. From the positions, we substract the locally averaged position as we assume that the local point distribution is more important than the coordinates in the global reference frame. We evaluate the impact of elevating the point features to a higher-dimensional space and subtracting the local mean against a simple splatting operator which just averages the features of the points around each corresponding vertex.

We observe that not subtracting the local mean, and just using the xyz coordinates as features, heavily degrades the performance, causing the mIoU to drop from 52.9 to 43.0. This further reinforces the idea that the local point distribution is a good local feature to use in the first layers of the network.

Not elevating the point cloud features to a higher-dimensional space before applying the max-pool operation also hurts performance but not as severely. In our experiments, we elevate the features to 64 dimensions by using a series of fully connected layers.

Table 5 Instance segmentation performance on the maize and tomato plants of the Pheno4D dataset

	SBD Maize	Tomato
PointNet (Qi et al. 2017a)	69.7	47.3
PointNet++ (Qi et al. 2017b)	74.8	56.1
LatticeNet (ours)	**80.6**	**74.2**

Bold signifies the highest mean intersection-over-union (mIoU)

Table 6 Ablation study of the various components of LatticeNet. Various features are disabled (indicated in red) and the impact to the IoU is evaluated

	mIoU	LocalAvg	PointNet elevate	DeformSlice	Offsets zero sum
LN splat	37.8				
LN no local avg	43.0				
LN no elevate	46.8				
LN slice	50.4				
LN reg	52.7				
LatticeNet	**52.9**				

Finally, naive application of the splat operation performs worst with a mere 37.8 mIoU.

9.3 Performance

We report the time taken for a forward pass and the maximum memory used in our shallow and deep network on the first three evaluated datasets. The performance was measured on

Table 7 Average time used by the forward pass and the maximum memory used during training. An X indicates a method that failed to process the whole cloud due to memory limitations

	ShapeNet		ScanNet		SemanticKITTI	
	[ms]	[GB]	[ms]	[GB]	[ms]	[GB]
SplatNet	129	0.6	X	X	2931	8.9
Ours	49	0.5	180	6.5	143	3.5

Bold signifies the highest mean intersection-over-union (mIoU)

a NVIDIA Titan X Pascal and the results are gathered in Table 7.

In the case of motion segmentation, the inference times and memory used are the same as in the case of a single scan, as we use the same backbone network to extract features and the computational cost of fusing the temporal information is minimum. However for training, the network requires more memory with increasing time window due to the back-propagation through time. This scales linearly with the time window size and the amount of points in the cloud.

Despite the reduced memory usage compared to SplatNet and increased speed of execution, there are still memory savings possible by fusing the Distribute and PointNet operators into one GPU operation. This is similar to fusing our Deform-Slice and the classification layer. Additionally, we expect the network to become even faster as further advances on highly optimized kernels for convolution on sparse lattices become available. At the moment, the convolutions are performed by our custom CUDA kernels. Tighter integration however with highly optimized libraries like cuDNN (Chetlur et al. 2014) could be beneficial.

10 Conclusion

We presented LatticeNet, a novel method for point cloud segmentation. A sparse permutohedral lattice allows us to efficiently process large point clouds. The usage of PointNet together with a data-dependent interpolation alleviates the quantization issues of other methods. Experiments on four datasets show state-of-the-art results, at a reduced time and memory budget.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

A large scale spatio-temporal dataset of point clouds of maize and tomato plants. https://www.ipb.uni-bonn.de/data/pheno4d/. Accessed: 2021-01-1.

Baek, J., & Adams, A. (2009). Some useful properties of the permutohedral lattice for Gaussian filtering. *Other Words 10*(1).

Barron, J.T., Adams, A., YiChang, S., & Hernández, C. (2015). Fast bilateral-space stereo for synthetic defocus—Supplemental material. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–15.

Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., & Gall, J. (2019) SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Berman, M., Triki, A.R., & Blaschko, M.B. (2018). The Lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4413–4421.

Chen, L-C., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587.

Chen, W., Han, X., Li, G., Chen, C., Xing, J., Zhao, Y., & Li, H. (2018). Deep RBFNet: Point cloud feature learning using radial basis functions. arXiv preprint arXiv:1812.04302.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., & Shelhamer, E. (2014). cuDNN: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759.

Choy, C., Gwak, J., & Savarese, S. (2019). 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. arXiv preprint arXiv:1904.08755.

Dai, A., & Nießner, M. (2018). 3DMV: Joint 3D-multi-view prediction for 3D semantic scene segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 452–468.

Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., & Nießner, M. (2017). ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5828–5839.

De Brabandere, B., Neven, D., & Van Gool, L. (2017). Semantic instance segmentation with a discriminative loss function. arXiv preprint arXiv:1708.02551.

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 3844–3852.

Graham, B., Engelcke, M., & van der Maaten, L. (2018). 3D semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9224–9232.

Gu, X., Wang, Y., Wu, C., Lee, Y.J., & Wang, P. (2019). HPLFlowNet: Hierarchical Permutohedral Lattice FlowNet for Scene Flow Estimation on Large-scale Point Clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp 3254–3263.

He, K., Zhang, X., Ren, S., & Sun, J. (2016a) Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

He, K., Zhang, X., Ren, S., & Sun, J. (2016b) Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 630–645.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K.Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708.

Huang, J., Zhang, H., Yi, L., Funkhouser, T., Nießner, M., & Guibas, L.J. (2019). TextureNet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4440–4449.

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., & Chen, B. (2018). PointCNN: Convolution on x-transformed points. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 820–830.

Lin, G., Milan, A., Shen, C., & Reid, I. (2017). RefineNet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1925–1934.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440.

Masci, J., Boscaini, D., Bronstein, M., & Vandergheynst, P. (2015). Geodesic convolutional neural networks on Riemannian manifolds. In *Workshop Proceedings of the IEEE International Conference on Computer Vision (ICCV Workshops)*, pp. 37–45.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., & Bronstein, M.M. (2017). Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5115–5124.

Neven, D., De Brabandere, B., Proesmans, M., & Van Gool, L. (2019). Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8837–8845.

Nießner, M., Zollhöfer, M., Izadi, S., & Stamminger, M. (2013). Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG), 32*(6), 1–11.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Pham, Q-H., Hua, B-S., Nguyen, T., & Yeung, S-K. (2019a). Real-time progressive 3D semantic segmentation for indoor scenes. In *Proceedings of the IEEE Workshop on Applications of Computer Vision*, pp. 1089–1098.

Pham, Q-H., Nguyen, T., Hua, B-S., Roig, G., & Yeung, S-K. (2019b). JSIS3D: Joint semantic-instance segmentation of 3D point clouds with multi-task pointwise networks and multi-value conditional random fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8827–8836.

Qi, C.R., Su, H., Mo, K., & Guibas, L.J. (2017a). PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660.

Qi, C.R., Yi, L., Su, H., & Guibas, L.J. (2017b). PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, pp. 5099–5108.

Qi, C.R., Litany, O., He, K., & Guibas, L.J. (2019) Deep Hough voting for 3D object detection in point clouds. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 9277–9286.

Ravanbakhsh, S., Schneider, J.G., & Póczos, B. (2016). Deep Learning with Sets and Point Clouds. arXiv preprint arXiv:1611.04500.

Rethage, D., Wald, J., Sturm, J., Navab, N., & Tombari, F. (2018). Fully-convolutional point networks for large-scale point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 596–611.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pp. 234–241.

Rosu, R.A., Schütt, P., Quenzel, J., & Behnke, S. (2020). LatticeNet: Fast point cloud segmentation using permutohedral lattices. *Proceedings of Robotics: Science and Systems*.

Shi, H., Lin, G., Wang, H., Hung, T-Y., & Wang, Z. (2020). SpSequenceNet: Semantic Segmentation Network on 4D Point Clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4574–4583.

Stotko, P., Krumpen, S., Weinmann, M., & Klein, R. (2019). Efficient 3D Reconstruction and Streaming for Group-Scale Multi-Client Live Telepresence. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 19–25.

Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M-H., & Kautz, J. (2018). SplatNet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2530–2539.

Tanke, J., Kwon, O-H., Stotko, P., Rosu, R.A., Weinmann, M., Errami, H., Behnke, S., Bennewitz, M., Klein, R., Weber, A., et al. (2019). Bonn Activity Maps: Dataset Description. arXiv preprint arXiv:1912.06354.

Tatarchenko, M., Park, J., Koltun, V., & Zhou, Q-Y. (2018). Tangent convolutions for dense prediction in 3D. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3887–3896.

Tchapmi, L., Choy, C., Armeni, I., Gwak, J., & Savarese, S. (2017). SEGCloud: Semantic segmentation of 3D point clouds. In *International Conference on 3D Vision (3DV)*, pp. 537–547. IEEE.

Thomas, H., Qi, C.R., Deschaud, J-E., Marcotegui, B., Goulette, F., & Guibas, L.J. (2019). KPConv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE Int. Conference on Computer Vision (ICCV)*, pp. 6411–6420.

Wang, S., Suo, S., Ma, W.C., Pokrovsky, A., & Urtasun, R. (2018a). Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2589–2597.

Wang, W., Yu, R., Huang, Q., & Neumann, U. (2018b). SGPN: Similarity group proposal network for 3D point cloud instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2569–2578.

Wang, X., Liu, S., Shen, X., Shen, C., & Jia, J. (2019). Associatively segmenting instances and semantics in point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4096–4105.

Wu, B., Zhou, X., Zhao, S., Yue, X., & Keutzer, K. (2018). SqueezeSegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. arXiv preprint arXiv:1809.08495.

Wu, W., Qi, Z., & Fuxin, L. (2019). PointConv: Deep convolutional networks on 3D point clouds. In *Proceedings of the IEEE Con-*

ference on Computer Vision and Pattern Recognition (CVPR), pp. 9621–9630.
Wu, Y., & He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19.
Yang, B., Wang, J., Clark, R., Hu, Q., Wang, S., Markham, A., & Trigoni, N. (2019). Learning object bounding boxes for 3D instance segmentation on point clouds. arXiv preprint arXiv:1906.01140.
Yi, L., Kim, L. G., Ceylan, D., Shen, I., Yan, M., Su, H., et al. (2016). A scalable active framework for region annotation in 3D shape collections. *ACM Transactions on Graphics (ToG), 35*(6), 210.
Yi, L., Zhao, W., Wang, H., Sung, M., & Guibas, L.J. (2019). GSPN: Generative shape proposal network for 3D instance segmentation in point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3947–3956.
Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., & Smola, A.J. (2017). Deep sets. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, pp. 3391–3401.

Radu Alexandru Rosu is a PhD student in the group of Autonomous Intelligent Systems, University of Bonn, Germany. He holds a master's degree in computer science from the University of Bonn and a bachelor's degree in computer science from the University of Salamanca, Spain. His research interest are in the area of 3D deep learning. He seeks to create novel neural network models capable of understanding, processing and reconstructing 3D data.

Peer Schütt is a computer science master's student at the University of Bonn. He has worked in the Autonomous Intelligent Systems group since 2017. His research interests are in the area of deep learning and augmented reality.

Jan Quenzel received his M.Sc. degree in Computer Science from the University of Lübeck in 2015. Since August 2015, he is a member of the Autonomous Intelligent Systems Group at the University of Bonn. His research focuses on LiDAR and visual odometry for micro aerial vehicles, surface reconstruction, and sensor calibration.

Sven Behnke received his Ph.D. from Freie Universität Berlin in 2002. He worked in 2003 as postdoctoral researcher at the International Computer Science Institute, Berkeley. From 2004 to 2008, he headed the Humanoid Robots Group at Albert-Ludwigs-Universi tat Freiburg. Since 2008, he is professor for Autonomous Intelligent Systems at the University of Bonn. His research interests include micro aerial vehicles, cognitive robotics, computer vision, and machine learning.