LatticeNet: Fast Point Cloud Segmentation Using Permutohedral Lattices

Radu Alexandru Rosu Peer Schütt Jan Quenzel Sven Behnke

***Abstract*—Deep convolutional neural networks (CNNs) 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.**

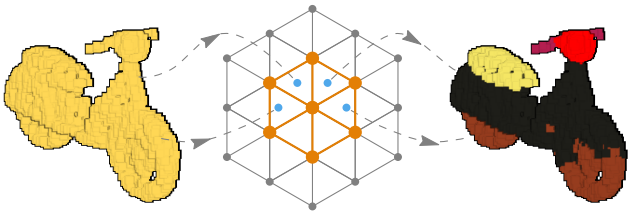


Fig. 1: Semantic segmentation: LatticeNet takes raw point clouds as input and embeds them into a sparse lattice where convolutions are applied. Features on the lattice are projected back onto the point cloud to yield a final segmentation.

I. 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 [19, 4, 18]. The task of semantically labelling 3D data is still an open area of research however 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 [34].

In this work, we propose LatticeNet, a novel approach for point cloud segmentation which alleviates the previously mentioned problems. 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).

II. Related Work

Semantic segmentation approaches applied to 3D data 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 [25] is one of the pioneering works. The approach 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++ [26] which applies PointNet hierarchically, capturing both local and global contextual information.

Chen et al. [5] 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.

Instead, PointCNN [17] 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: Voxel-based approaches discretize the space in cubic or tetrahedral volume elements which are used for

3D convolutions.

SEGCloud [34] 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 permutoherdral lattice stored sparsely. Additionally, their voxelization results in a loss of information due to the discretization of the space. Our approach avoids quantization issues by using a PointNet architecture to summarize the local neighborhood.

Rethage et al. [28] 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 [31] 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 [1]. 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: Mesh-based approaches operate on triangular or quadrilateral meshes. The connectivity information provided by the faces of the mesh allows to easily compute normal vectors and to establish local tangent planes.

GCNN [20] 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 [21] 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 [16] 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: Graph-based approaches operate on vertices of a graph connected in an arbitrary topology, without the restrictions of triangular or quadrilateral meshes.

Wang et al. [35] and Wu et al. [37] 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. [10] formulate CNNs in the context of spectral graph theory. They define the convolution in the Fourier domain with Chebychev 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. [24] 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.

TangentConv [33] 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. A heavy preprocessing step for normal calculation is required. In contrast, our approach can deal with raw point clouds without requiring normals.

III. 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.

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.

IV. 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 [1].

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 piecewise linear outputs as they use barycentric interpolation. In contrast, standard quantization in cubic voxels create piecewise constant outputs, leading to discretization artefacts.

V. 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.

A. 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 [31, 12].

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).

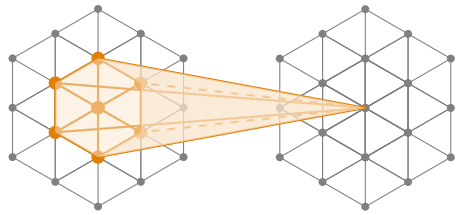


Fig. 2: Convolution: The neighboring vertices of a lattice are convolved similarly to standard 2D convolutions. If a neighbor is not allocated in the sparse structure, we assume that it has a value of zero.

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).

B. Proposed Operations on Permutohedral Lattice

The operations defined in section Sec. V-A are typically used in a cascade of splat-conv-slice to obtain dense predictions [31]. However, splatting and slicing act as Gaussian kernel low-pass filtering encoded information [1]. 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 lattices 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.

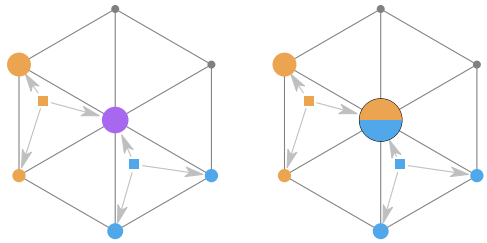
Instead, our distribute operators $\mathcal{D}_G$ and $\mathcal{D}_F$ concatenate coordinates and features of the contributing points:

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

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

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

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



(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.

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 [2]. 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 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.

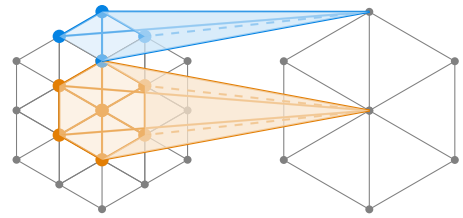


Fig. 4: Coarsen: Downsampling of the lattice is performed by embedding the coarse lattice in the finer one and convolving over the neighbors. This effectively performs a strided convolution. Transposed convolution is performed in an analogous manner by embedding a fine lattice into a coarse one.

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:

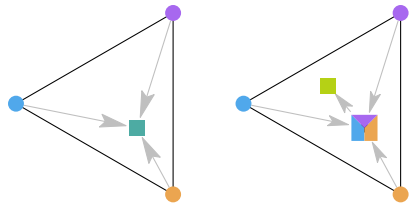
$$\Delta \mathbf{b}_p = \mathcal{F}(\mathbf{q}_p) = \sigma(\mathbf{q}_p \cdot \mathbf{W} + b). \tag{10}$$

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



(a) Slice (b) DeformSlice

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.

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 [27] and [40] of equivariant layers and design $\mathcal{F}$ as:

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

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

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

VI. 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.

Our network architecture has a U-Net structure [29] and is visualized in Fig. 6 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 [13], 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.

VII. Implementation

Our lattice is stored sparsely on a hash map structure, which allows for fast access of neighboring vertices. Unlike [31], 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,

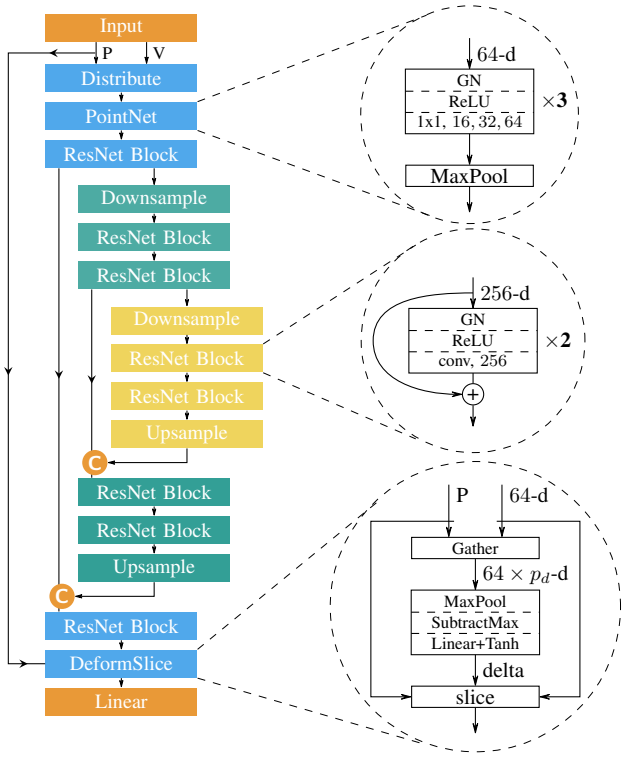


Fig. 6: Architecture: Our model follows a U-Net structure. For ease of representation, blocks which are repeated one after another are indicated with a multiplier on the right side of the operation.

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 [23].

Following recent works [14, 15], all convolutions are pre-activated using Group Normalization [38] and a ReLU unit. We chose Group Normalization instead of the standard batch normalization because it is more stable when the batch size is small. We use the default of 32 groups.

The models were trained using the Adam optimizer, using 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.

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

VIII. Experiments

We evaluate our proposed lattice network on three different datasets: ShapeNet [39], ScanNet [9] and SemanticKITTI [3]. For the task of semantic segmentation we report the mean Intersection over Union (mIoU). We use a shallow model for ShapeNet 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].

A. Evaluation of Segmentation Accuracy

ShapeNet part segmentation is a subset of the ShapeNet dataset [39] 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. The results for our method and five competing methods are gathered in Tab. I and visualized in Fig. 8.

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 for 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 [31] we deem that experimentation with more architectures or with different regularization strengths for individual objects would overfit the dataset.

ScanNet 3D segmentation [9] 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 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 $2\,\mathrm{cm}$ 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 $10\,\mathrm{cm}$.

SemanticKITTI [3] contains semantically annotated scans from the KITTI dataset which consists of laser scans from real urban environments. The scans are annotated with a total of 19 classes and each scan contains between 82K and 129K points. We process each scan entirely without any cropping. The results are provided in Tab. II. 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 [3], DarkNet21Seg [3] and SqueezeSegV2 [36] are methods that operate on a 2D image by wrapping the laser scans to 2D using spherical coordinates. In contrast, our method can operate on general point clouds, directly in 3D.

Bonn Activity Maps [32] 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. 7. 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.

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

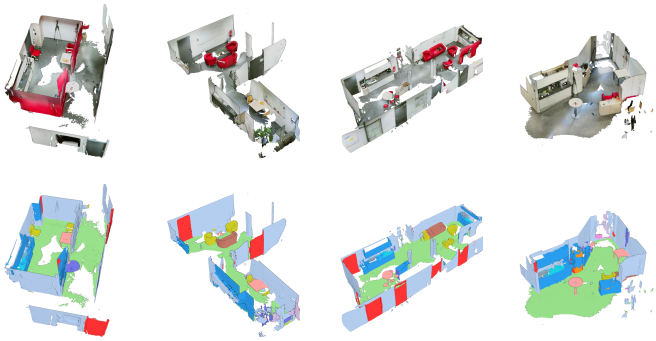


Fig. 7: Bonn Activity Maps segmentations. Colored meshes are reconstructed from KinectV2 data using volumetric integration [22, 30] and semantically segmented using LatticeNet. Color coding of semantic labels corresponds to the ScanNet dataset [9].

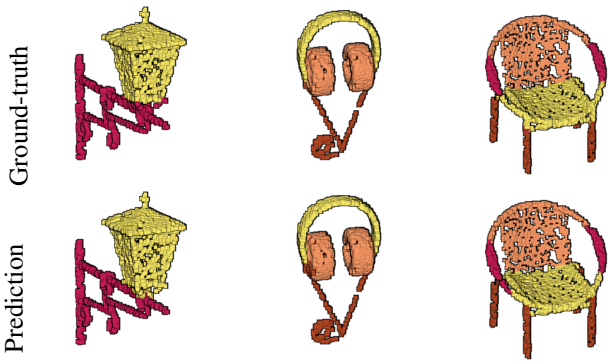


Fig. 8: ShapeNet [39] results of our method.

B. 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 does not use learned barycentric interpolation. We evaluate it on the 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:

$$L = \frac{1}{|P|} \sum_{p \in P} \left(\sum_{v \in I_p} \Delta b_{pv} \right)^2 . \tag{14}$$

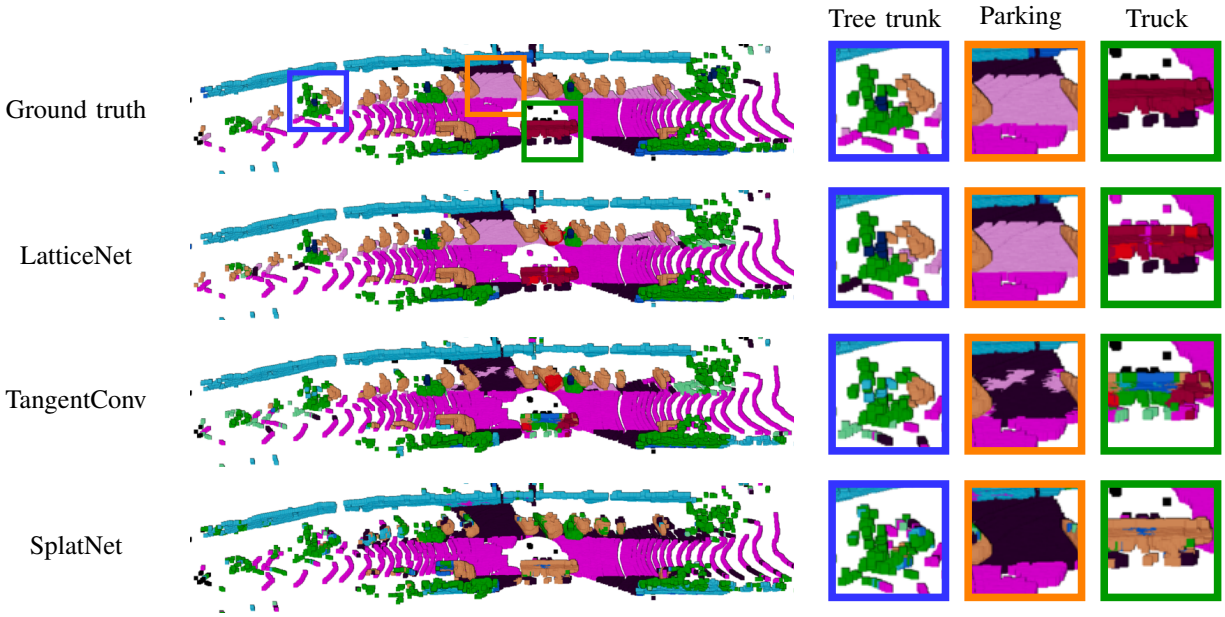


Fig. 9: SemanticKITTI results. We compare the prediction from our LatticeNet with the results from TangentConv [33] and SplatNet [31]. We can observe that our approach can better learn small objects like tree trunks, despite their relatively small number of points. Additionally, the network also effectively makes use of contextual information in order to correctly predict the parking place due to the existence of nearby cars.

TABLE I: Results on ShapeNet part segmentation [39].

#instances		2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271
	instance avg.	air-plane	bag	cap	car	chair	ear-phone	guitar	knife	lamp	laptop	motor-bike	mug	pistol	rocket	skate-board	table
PointNet [25]	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++ [26]	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 [31]	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 [31]	**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 [28]	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**

TABLE II: Results on SemanticKITTI [3].

Approach	mIoU	road	sidewalk	parking	other-ground	building	car	truck	bicycle	motorcycle	other-vehicle	vegetation	trunk	terrain	person	bicyclist	motorcyclist	fence	pole	traffic sign
PointNet [25]	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 [31]	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++ [26]	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) [7]	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 [36]	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 [33]	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 [3]	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 [3]	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

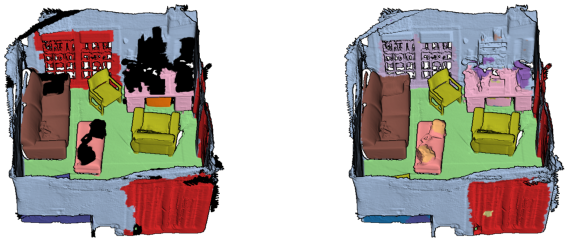


Fig. 10: ScanNet results. The left image shows the ground truth and the right one our prediction.

TABLE III: Results on ScanNet [9]

Method	mIOU
PointNet++ [26]	33.9
SplatNet [31]	39.3
TangetConv [33]	43.8
3DMV‡ [8]	48.4
MinkowskiNet42 (5cm) [7]	67.9
SparseConvNet [11]†	72.5
MinkowskiNet42 (2cm) [7]†	**73.4**
Ours	64.0

†: post-CVPR submissions. ‡: uses 2D images additionally.

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 Tab. V.

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 and in a higher-dimensional space by a PointNet-like architecture. The positions and features of the point cloud are treated separately 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 IoU 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.

Finally, naively performing a splat operation performs worst with a mere 37.8 IoU.

TABLE IV: 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

TABLE V: 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**	■	■	■	■

C. Performance

We report the time taken for a forward pass and the maximum memory used in our shallow and deep network on the three evaluated datasets. The performance was measured on a NVIDIA Titan X Pascal and the results are gathered in Tab. IV.

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 DeformSlice 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 [6] could be beneficial.

IX. Conclusion

We presented LatticeNet, a novel method for semantic segmentation of point clouds. 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 three datasets show state-of-the-art results, at a reduced time and memory budget. In the future, we would like to incorporate temporal information into our model in order to process sequential data.

References

[1] Jongmin Baek and Andrew Adams. Some useful properties of the permutohedral lattice for Gaussian filtering. *In other words*, 10(1):0, 2009.

[2] Jonathan T Barron, Andrew Adams, S YiChang, and Carlos Hernández. Fast bilateral-space stereo for synthetic defocus Supplemental material. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–15, 2015.

[3] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Juergen Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE Int. Conference on Computer Vision (ICCV)*, 2019.

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[5] Weikai Chen, Xiaoguang Han, Guanbin Li, Chao Chen, Jun Xing, Yajie Zhao, and Hao Li. Deep RBFNet: Point cloud feature learning using radial basis functions. *arXiv preprint arXiv:1812.04302*, 2018.

[6] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[7] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. *arXiv preprint arXiv:1904.08755*, 2019.

[8] Angela Dai and Matthias Nießner. 3DMV: Joint 3D-multi-view prediction for 3D semantic scene segmentation. In *Proc. of the European Conference on Computer Vision (ECCV)*, pages 452–468, 2018.

[9] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5828–5839, 2017.

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, pages 3844–3852, 2016.

[11] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9224–9232, 2018.

[12] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. HPLFlowNet: Hierarchical Permutohedral Lattice FlowNet for Scene Flow Estimation on Large-scale Point Clouds. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3254–3263, 2019.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proc. of the European Conference on Computer Vision (ECCV)*, pages 630–645, 2016.

[15] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[16] Jingwei Huang, Haotian Zhang, Li Yi, Thomas Funkhouser, Matthias Nießner, and Leonidas J Guibas. TextureNet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4440–4449, 2019.

[17] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on x-transformed points. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, pages 820–830, 2018.

[18] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. RefineNet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1925–1934, 2017.

[19] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[20] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *Workshop Proc. of the IEEE Int. Conference on Computer Vision (ICCV Workshops)*, pages 37–45, 2015.

[21] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5115–5124, 2017.

[22] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):1–11, 2013.

[23] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[24] Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. Real-time progressive 3D semantic segmentation for indoor scenes. In *Proc. of the IEEE Workshop on Applications of Computer Vision*, pages 1089–1098, 2019.

[25] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017.

[26] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, pages 5099–5108, 2017.

[27] Siamak Ravanbakhsh, Jeff G. Schneider, and Barnabás Póczos. Deep Learning with Sets and Point Clouds. *arXiv preprint arXiv:1611.04500*, 2016.

[28] Dario Rethage, Johanna Wald, Jurgen Sturm, Nassir Navab, and Federico Tombari. Fully-convolutional point networks for large-scale point clouds. In *Proc. of the European Conference on Computer Vision (ECCV)*, pages 596–611, 2018.

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 234–241, 2015.

[30] Patrick Stotko, Stefan Krumpen, Michael Weinmann, and Reinhard Klein. Efficient 3D Reconstruction and Streaming for Group-Scale Multi-Client Live Telepresence. In *Proc. of the IEEE Int. Symposium on Mixed and Augmented Reality (ISMAR)*, pages 19–25, 2019.

[31] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. SplatNet: Sparse lattice networks for point cloud processing. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2530–2539, 2018.

[32] Julian Tanke, Oh-Hun Kwon, Patrick Stotko, Radu Alexandru Rosu, Michael Weinmann, Hassan Errami, Sven Behnke, Maren Bennewitz, Reinhard Klein, Andreas Weber, et al. Bonn Activity Maps: Dataset Description. *arXiv preprint arXiv:1912.06354*, 2019.

[33] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3D. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3887–3896, 2018.

[34] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. SEGCloud: Semantic segmentation of 3D point clouds. In *Intl. Conf. on 3D Vision (3DV)*, pages 537–547. IEEE, 2017.

[35] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2589–2597, 2018.

[36] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. SqueezeSegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. *arXiv preprint arXiv:1809.08495*, 2018.

[37] Wenxuan Wu, Zhongang Qi, and Li Fuxin. PointConv: Deep convolutional networks on 3D point clouds. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9621–9630, 2019.

[38] Yuxin Wu and Kaiming He. Group normalization. In *Proc. of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[39] Li Yi, Vladimir G Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, Leonidas Guibas, et al. A scalable active framework for region annotation in 3D shape collections. *ACM Transactions on Graphics (ToG)*, 35(6):210, 2016.

[40] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, pages 3391–3401, 2017.