4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks

Christopher Choy
chrischoy@stanford.edu

JunYoung Gwak
jgwak@stanford.edu

Silvio Savarese
ssilvio@stanford.edu

Abstract

In many robotics and VR/AR applications, 3D-videos are readily-available sources of input (a continuous sequence of depth images, or LIDAR scans). However, these 3D-videos are processed frame-by-frame either through 2D convnets or 3D perception algorithms in many cases. In this work, we propose 4-dimensional convolutional neural networks for spatio-temporal perception that can directly process such 3D-videos using high-dimensional convolutions. For this, we adopt sparse tensors [8, 9] and propose the generalized sparse convolution which encompasses all discrete convolutions. To implement the generalized sparse convolution, we create an open-source auto-differentiation library for sparse tensors that provides extensive functions for high-dimensional convolutional neural networks.[1] We create 4D spatio-temporal convolutional neural networks using the library and validate them on various 3D semantic segmentation benchmarks and proposed 4D datasets for 3D-video perception. To overcome challenges in the high-dimensional 4D space, we propose the hybrid kernel, a special case of the generalized sparse convolution, and the trilateral-stationary conditional random field that enforces spatio-temporal consistency in the 7D space-time-chroma space. Experimentally, we show that convolutional neural networks with only generalized sparse convolutions can outperform 2D or 2D-3D hybrid methods by a large margin.[2] Also, we show that on 3D-videos, 4D spatio-temporal convolutional neural networks are robust to noise, outperform 3D convolutional neural networks and are faster than the 3D counterpart in some cases.

感知？

1. Introduction

In this work, we are interested in 3D-video perception. A 3D-video is a temporal sequence of 3D scans such as a video from a depth camera, a sequence of LIDAR scans, or a multiple MRI scans of the same object or a body part (Fig. 1). As LIDAR scanners and depth cameras become more affordable and widely used for robotics applications, 3D-videos became readily-available sources of input for robotics systems or AR/VR applications.

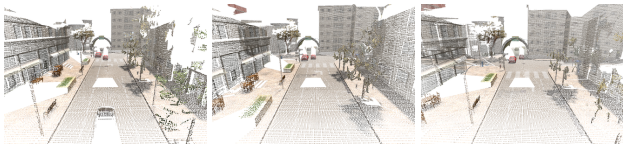

Figure 1: An example of 3D video: 3D scenes at different time steps. Best viewed on display.

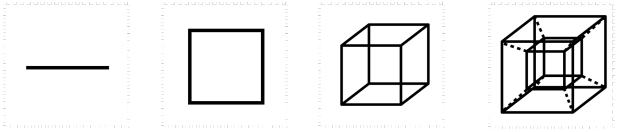

Figure 2: 2D projections of hypercubes in various dimensions

NotAvailableNow

However, there are many technical challenges in using 3D-videos for high-level perception tasks. First, 3D data requires heterogeneous representations and processing those either alienates users or makes it difficult to integrate into larger systems. Second, the performance of the 3D convolutional neural networks is worse or on-par with 2D convolutional neural networks. Third, there are limited number of open-source libraries for fast large-scale 3D data.

To resolve most, if not all, of the challenges in the high-dimensional perception, we adopt a sparse tensor [8, 9] for our problem and propose the generalized sparse convolutions. The generalized sparse convolution encompasses all discrete convolutions as its subclasses and is crucial for high-dimensional perception. We implement the generalized sparse convolution and all standard neural network functions in Sec. 4 and open-source the library.

We adopt the sparse representation for several reasons. Currently, there are various concurrent works for 3D perception: a dense 3D convolution [5], pointnet-variants [22, 23], continuous convolutions [11, 15], surface convolutions [20, 29], and an octree convolution [24]. Out of these representations, we chose a sparse tensor due to its expressiveness and generalizability for high-dimensional spaces. Also, it allows

[1] https://github.com/StanfordVL/MinkowskiEngine

[2] At the time of submission, we achieved the best performance on ScanNet [5] with 67.9% mIoU

homogeneous data representation within traditional neural network libraries since most of them support sparse tensors.

Second, the sparse convolution closely resembles the standard convolution (Sec. 3) which is proven to be successful in 2D perception as well as 3D reconstruction [4], feature learning [33], and semantic segmentation [30].

Third, the sparse convolution is efficient and fast. It only computes outputs for predefined coordinates and saves them into a compact sparse tensor (Sec. 3). It saves both memory and computation especially for 3D scans or high-dimensional data where most of the space is empty.

Thus, we adopt the sparse representation for the our problem and create the first large-scale 3D/4D networks or Minkowski networks.[3] We named them after the space-time continuum, Minkowski space, in Physics.

However, even with the efficient representation, merely scaling the 3D convolution to high-dimensional spaces results in significant computational overhead and memory consumption due to the curse of dimensionality. A 2D convolution with kernel size 5 requires $5^2 = 25$ weights which increases exponentially to $5^3 = 125$ in a 3D cube, and 625 in a 4D tesseract (Fig. 2). This exponential increase, however, does not necessarily lead to better performance and slows down the network significantly. To overcome this challenge, we propose custom kernels with non-(hyper)-cubic shapes using the generalized sparse convolution.

Finally, the predictions from the 4D spatio-temporal generalized sparse convnets are not necessarily consistent throughout the space and time. To enforce consistency, we propose high-dimensional conditional random fields defined in a 7D trilateral space (space-time-color) with a stationary pairwise consistency function. We use variational inference to convert the conditional random field to differentiable recurrent layers which can be implemented in as a 7D generalized sparse convnet and train both the 4D and 7D networks end-to-end.

Experimentally, we use various 3D benchmarks that cover both indoor [5, 2] and outdoor spaces [27, 25]. First, we show that networks with only generalized sparse 3D conv nets can outperform 2D or hybrid deep-learning algorithms by a large margin.[4] Also, we create 4D datasets from Synthia [27] and Varcity [25] and report ablation studies of temporal components. Experimentally, we show that the generalized sparse conv nets with the hybrid kernel outperform sparse convnets with tesseract kernels. Also, the 4D generalized sparse convnets are more robust to noise and sometimes more efficient in some cases than the 3D counterpart.

[3] At the time of submission, our proposed method was the first very deep 3D convolutional neural networks with more than 20 layers.

[4] We achieved 67.9% mIoU on the ScanNet benchmark outperforming all algorithms including the best peer-reviewed work [6] by 19% mIoU at the time of submission.

2. Related Work

The 4D spatio-temporal perception fundamentally requires 3D perception as a slice of 4D along the temporal dimension is a 3D scan. However, as there are no previous works on 4D perception using neural networks, we will primarily cover 3D perception, specifically 3D segmentation using neural networks. We categorized all previous works in 3D as either (a) 3D-convolutional neural networks or (b) neural networks without 3D convolutions. Finally, we cover early 4D perception methods. Although 2D videos are spatio-temporal data, we will not cover them in this paper as 3D perception requires radically different data processing, implementation, and architectures.

3D-convolutional neural networks. The first branch of 3D-convolutional neural networks uses a rectangular grid and a dense representation [30, 5] where the empty space is represented either as **0** or the signed distance function. This straightforward representation is intuitive and is supported by all major public neural network libraries. However, as the most space in 3D scans is empty, it suffers from high memory consumption and slow computation. To resolve this, OctNet [24] proposed to use the Octree structure to represent 3D space and convolution on it.

The second branch is sparse 3D-convolutional neural networks [28, 9]. There are two quantization methods used for high dimensions: a rectangular grid and a permutohedral lattice [1]. [28] used a permutohedral lattice whereas [9] used a rectangular grid for 3D classification and semantic segmentation.

The last branch is 3D pseudo-continuous convolutional neural networks [11, 15]. Unlike the previous works, they define convolutions using continuous kernels in a continuous space. However, finding neighbors in a continuous space is expensive, as it requires KD-tree search rather than a hash table, and are susceptible to uneven distribution of point clouds.

Neural networks without 3D convolutions. Recently, we saw a tremendous increase in neural networks without 3D convolutions for 3D perception. Since 3D scans consist of thin observable surfaces, [20, 29] proposed to use 2D convolutions on the surface for semantic segmentation.

Another direction is PointNet-based methods [22, 23]. PointNets use a set of input coordinates as features for a multi-layer perceptron. However, this approach processes a limited number of points and thus a sliding window for cropping out a section from an input was used for large spaces making the receptive field size rather limited. [14] tried to resolve such shortcomings with a recurrent network on top of multiple pointnets, and [15] proposed a variant of 3D continuous convolution for lower layers of a PointNet and got a significant performance boost.

4D perception. The first 4D perception algorithm [18] proposed a dynamic deformable balloon model for 4D car-

diac image analysis. Later, [16] used a 4D Markov Random Fields for cardiac segmentation. Recently, a "Spatio-Temporal CNN" [34] combined a 3D-UNet with a 1D-AutoEncoder for temporal data and applied the model for auto-encoding brain fMRI images, but it is not a 4D-convolutional neural network.

In this paper, we propose the first high-dimensional convolutional neural networks for 4D spatio-temporal data, or 3D videos and the 7D space-time-chroma space. Compared with other approaches that combine temporal data with a recurrent neural network or a shallow model (CRF), our networks use a homogeneous representation and convolutions consistently throughout the networks. Instead of using an RNN, we use convolution for the temporal axis since it is proven to be more effective in sequence modeling [3].

3. Sparse Tensor and Convolution

In traditional speech, text, or image data, features are extracted densely. Thus, the most common representations of these data are vectors, matrices, and tensors. However, for 3-dimensional scans or even higher-dimensional spaces, such dense representations are inefficient due to the sparsity. Instead, we can only save the non-empty part of the space as its coordinates and the associated features. This representation is an N-dimensional extension of a sparse matrix; thus it is known as a sparse tensor. There are many ways to save such sparse tensors compactly [31], but we follow the COO format as it is efficient for neighborhood queries (Sec. 3.1). Unlike the traditional sparse tensors, we augment the sparse tensor coordinates with the batch indices to distinguish points that occupy the same coordinate in different batches [9]. Concisely, we can represent a set of 4D coordinates as $\mathcal{C} = \{(x_i, y_i, z_i, t_i)\}_i$ or as a matrix C and a set of associated features $\mathcal{F} = \{\mathbf{f}_i\}_i$ or as a matrix F. Then, a sparse tensor can be written as

$$C = \begin{bmatrix} x_1 & y_1 & z_1 & t_1 & b_1 \\ & & \vdots & & \\ x_N & y_N & z_N & t_N & b_N \end{bmatrix}, \; F = \begin{bmatrix} \mathbf{f}_1^T \\ \vdots \\ \mathbf{f}_N^T \end{bmatrix} \quad (1)$$

where b_i and $\mathbf{f}_i$ are the batch index and the feature associated to the i-th coordinate. In Sec. 6, we augment the 4D space with the 3D chromatic space and create a 7D sparse tensor for *trilateral* filtering.

3.1. Generalized Sparse Convolution

In this section, we generalize the sparse convolution [8, 9] for generic input and output coordinates and for arbitrary kernel shapes. The generalized sparse convolution encompasses not only all sparse convolutions but also the conventional dense convolutions. Let $x_{\mathbf{u}}^{\text{in}} \in \mathbb{R}^{N^{\text{in}}}$ be an N^{in}-dimensional input feature vector in a D-dimensional space at $\mathbf{u} \in \mathbb{R}^D$ (a D-dimensional coordinate), and convolution kernel weights be $\mathbf{W} \in \mathbb{R}^{K^D \times N^{\text{out}} \times N^{\text{in}}}$. We break down the weights into spatial weights with K^D matrices of size $N^{\text{out}} \times N^{\text{in}}$ as $W_{\mathbf{i}}$ for $|\{\mathbf{i}\}_{\mathbf{i}}| = K^D$. Then, the conventional dense convolution in D-dimension is

$$\mathbf{x}_{\mathbf{u}}^{\text{out}} = \sum_{\mathbf{i} \in \mathcal{V}^D(K)} W_{\mathbf{i}} \mathbf{x}_{\mathbf{u}+\mathbf{i}}^{\text{in}} \text{ for } \mathbf{u} \in \mathbb{Z}^D, \quad (2)$$

where $\mathcal{V}^D(K)$ is the list of offsets in D-dimensional hypercube centered at the origin. e.g. $\mathcal{V}^1(3) = \{-1, 0, 1\}$. The generalized sparse convolution in Eq. 3 relaxes Eq. 2.

$$\mathbf{x}_{\mathbf{u}}^{\text{out}} = \sum_{\mathbf{i} \in \mathcal{N}^D(\mathbf{u}, \mathcal{C}^{\text{in}})} W_{\mathbf{i}} \mathbf{x}_{\mathbf{u}+\mathbf{i}}^{\text{in}} \text{ for } \mathbf{u} \in \mathcal{C}^{\text{out}} \quad (3)$$

where $\mathcal{N}^D$ is a set of offsets that define the shape of a kernel and $\mathcal{N}^D(\mathbf{u}, \mathcal{C}^{\text{in}}) = \{\mathbf{i} | \mathbf{u} + \mathbf{i} \in \mathcal{C}^{\text{in}}, \mathbf{i} \in \mathcal{N}^D\}$ as the set of offsets from the current center, $\mathbf{u}$, that exist in $\mathcal{C}^{\text{in}}$. $\mathcal{C}^{\text{in}}$ and $\mathcal{C}^{\text{out}}$ are predefined input and output coordinates of sparse tensors. First, note that the input coordinates and output coordinates are not necessarily the same. Second, we define the shape of the convolution kernel arbitrarily with $\mathcal{N}^D$. This generalization encompasses many special cases such as the dilated convolution and typical hypercubic kernels. Another interesting special case is the "sparse submanifold convolution" [9] when we set $\mathcal{C}^{\text{out}} = \mathcal{C}^{\text{in}}$ and $\mathcal{N}^D = \mathcal{V}^D(K)$. If we set $\mathcal{C}^{\text{in}} = \mathcal{C}^{\text{out}} = \mathbb{Z}^D$ and $\mathcal{N}^D = \mathcal{V}^D(K)$, the generalized sparse convolution becomes the conventional dense convolution (Eq. 2). If we define the $\mathcal{C}^{\text{in}}$ and $\mathcal{C}^{\text{out}}$ as multiples of a natural number and $\mathcal{N}^D = \mathcal{V}^D(K)$, we have a strided dense convolution.

4. Minkowski Engine

In this section, we propose an open-source auto-differentiation library for sparse tensors and the generalized sparse convolution (Sec. 3). As it is an extensive library with many functions, we will only cover essential forward-pass functions.

4.1. Sparse Tensor Quantization

The first step in the sparse convolutional neural network is the data processing to generate a sparse tensor, which converts an input into unique coordinates, associated features, and optionally labels when training for semantic segmentation. In Alg. 1, we list the GPU function for this process. When a dense label is given, it is important that we ignore voxels with more than one unique labels. This can be done by marking these voxels with `IGNORE_LABEL`. First, we convert all coordinates into hash keys and find all unique hashkey-label pairs to remove collisions. Note that `SortByKey`, `UniqueByKey`, and `ReduceByKey` are all standard `Thrust` library functions [19]. The reduction function $f((l_x, i_x), (l_y, i_y)) \Rightarrow$ (`IGNORE_LABEL`, i_x) takes

Algorithm 1 GPU Sparse Tensor Quantization

Inputs: coordinates $C_p \in \mathbb{R}^{N \times D}$, features $F_p \in \mathbb{R}^{N \times N_f}$, target labels $\mathbf{l} \in \mathbb{Z}_+^N$, quantization step size v_l
$C'_p \leftarrow \text{floor}(C_p \,/\, v_l)$
$\mathbf{k} \leftarrow \text{hash}(C'_p)$, $\mathbf{i} \leftarrow \text{sequence(N)}$
$((\mathbf{i}', \mathbf{l}'), k') \leftarrow \text{SortByKey}((\mathbf{i}, \mathbf{l}), \text{key=}\mathbf{k})$
$(\mathbf{i}'', (\mathbf{k}'', \mathbf{l}'')) \leftarrow \text{UniqueByKey}(\mathbf{i}', \text{key=}(\mathbf{k}', \mathbf{l}'))$
$(\mathbf{l}''', \mathbf{i}''') \leftarrow \text{ReduceByKey}((\mathbf{l}'', \mathbf{i}''), \text{key=}\mathbf{k}'', \text{fn=}f)$
return $C'_p[\mathbf{i}''', :], F_p[\mathbf{i}''', :], \mathbf{l}'''$

label-key pairs and returns the `IGNORE_LABEL` since at least two label-key pairs in the same key means there is a label collision. A CPU-version works similarly except that all reduction and sorting are processed serially.

4.2. Generalized Sparse Convolution

The next step in the pipeline is generating the output coordinates $\mathcal{C}^{\text{out}}$ given the input coordinates $\mathcal{C}^{\text{in}}$ (Eq. 3). When used in conventional neural networks, this process requires only a convolution stride size, input coordinates, and the stride size of the input sparse tensor (the minimum distance between coordinates). The algorithm is presented in the supplementary material. We create this output coordinates dynamically allowing an arbitrary output coordinates $\mathcal{C}^{\text{out}}$ for the generalized sparse convolution.

Next, to convolve an input with a kernel, we need a mapping to identify which inputs affect which outputs. This mapping is not required in conventional dense convolutions as it can be inferred easily. However, for sparse convolution where coordinates are scattered arbitrarily, we need to specify the mapping. We call this mapping the kernel maps and define them as pairs of lists of input indices and output indices, $\mathbf{M} = \{(I_\mathbf{i}, O_\mathbf{i})\}_\mathbf{i}$ for $\mathbf{i} \in \mathcal{N}^D$. Finally, given the input and output coordinates, the kernel map, and the kernel weights $W_\mathbf{i}$, we can compute the generalized sparse convolution by iterating through each of the offset $\mathbf{i} \in \mathcal{N}^D$ (Alg. 2) where $I[n]$ and $O[n]$ indicate the n-th element of the list of

Algorithm 2 Generalized Sparse Convolution

Require: Kernel weights $\mathbf{W}$, input features F^i, output feature placeholder F^o, convolution mapping $\mathbf{M}$,
1: $F^o \leftarrow \mathbf{0}$ // set to 0
2: **for all** $W_\mathbf{i}, (I_\mathbf{i}, O_\mathbf{i}) \in (\mathbf{W}, \mathbf{M})$ **do**
3: $\quad F_{\text{tmp}} \leftarrow W_\mathbf{i}[F^i_{I_\mathbf{i}[1]}, F^i_{I_\mathbf{i}[2]}, ..., F^i_{I_\mathbf{i}[n]}]$ // (cu)BLAS
4: $\quad F_{\text{tmp}} \leftarrow F_{\text{tmp}} + [F^o_{O_\mathbf{i}[1]}, F^o_{O_\mathbf{i}[2]}, ..., F^o_{O_\mathbf{i}[n]}]$
5: $\quad [F^o_{O_\mathbf{i}[1]}, F^o_{O_\mathbf{i}[2]}, ..., F^o_{O_\mathbf{i}[n]}] \leftarrow F_{\text{tmp}}$
6: **end for**

indices I and O respectively and F^i_n and F^o_n are also n-th input and output feature vectors respectively. The transposed generalized sparse convolution (deconvolution) works similarly except that the role of input and output coordinates is reversed.

4.3. Max Pooling

Unlike dense tensors, on sparse tensors, the number of input features varies per output. Thus, this creates non-trivial implementation for a max/average pooling. Let $\mathbf{I}$ and $\mathbf{O}$ be the vector that concatenated all $\{I_\mathbf{i}\}_\mathbf{i}$ and $\{O_\mathbf{i}\}_\mathbf{i}$ for $\mathbf{i} \in \mathcal{N}^D$ respectively. We first find the number of inputs per each output coordinate and indices of the those inputs. Alg. 3 reduces the input features that map to the same output coordinate. `Sequence`(n) generates a sequence of integers from 0 to n - 1 and the reduction function $f((k_1, v_1), (k_2, v_2)) = \min(v_1, v_2)$ which returns the minimum value given two key-value pairs. `MaxPoolKernel` is a custom CUDA kernel that reduces all features at a specified channel using $\mathbf{S}'$, which contains the first index of $\mathbf{I}$ that maps to the same output, and the corresponding output indices $\mathbf{O}$".

Algorithm 3 GPU Sparse Tensor MaxPooling

Input: input feature F, output mapping $\mathbf{O}$
$(\mathbf{I}', \mathbf{O}') \leftarrow \text{SortByKey}(\mathbf{I}, \text{key=}\mathbf{O})$
$\mathbf{S} \leftarrow \text{Sequence}(\text{length}(\mathbf{O}'))$
$\mathbf{S}', \mathbf{O}" \leftarrow \text{ReduceByKey}(\mathbf{S}, \text{key=}\mathbf{O}', \text{fn=}f)$
return MaxPoolKernel($\mathbf{S}'$, $\mathbf{I}'$, $\mathbf{O}"$, F)

4.4. Global / Average Pooling, Sum Pooling

An average pooling and a global pooling layer compute the average of input features for each output coordinate for average pooling or one output coordinate for global pooling. This can be implemented in multiple ways. We use a sparse matrix multiplication since it can be optimized on hardware or using a faster sparse BLAS library. In particular, we use the `cuSparse` library for sparse matrix-matrix (`cusparse_csrmm`) and matrix-vector multiplication (`cusparse_csrmv`) to implement these layers. Similar to the max pooling algorithm, $\mathbf{M}$ is the $(\mathbf{I}, \mathbf{O})$ input-to-output kernel map. For the global pooling, we create the kernel map that maps all inputs to the origin and use the same Alg. 4. The transposed pooling (unpooling) works similarly.

On the last line of the Alg. 4, we divide the pooled features by the number of inputs mapped to each output. However, this process could remove density information. Thus, we propose a variation that does not divide the number of inputs and named it the sum pooling.

4.5. Non-spatial Functions

For functions that do not require spatial information (coordinates) such as ReLU, we can apply the functions directly

Algorithm 4 GPU Sparse Tensor AvgPooling

Input: mapping $\mathbf{M} = (\mathbf{I}, \mathbf{O})$, features F, one vector $\mathbf{1}$
S_M = coo2csr(row=$\mathbf{O}$, col=$\mathbf{I}$, val=$\mathbf{1}$)
F' = cusparse_csrmm(S_M, F)
N = cusparse_csrmv(S_M, $\mathbf{1}$)
return F'/N

to the features F. Also, for batch normalization, as each row of F represents a feature, we could use the 1D batch normalization function directly on F.

5. Minkowski Convolutional Neural Networks

In this section, we introduce 4-dimensional spatio-temporal convolutional neural networks for spatio-temporal perception. We treat the time dimension as an extra spatial dimension and create networks with 4-dimensional convolutions. However, there are unique problems arising from high-dimensional convolutions. First, the computational cost and the number of parameters in the networks increase exponentially as we increase the dimension. However, we experimentally show that these increases do not necessarily lead to better performance. Second, the networks do not have an incentive to make the prediction consistent throughout the space and time with conventional cross-entropy loss alone.

To resolve the first problem, we make use of a special property of the generalized sparse convolution and propose non-conventional kernel shapes that not only save memory and computation, but also perform better. Second, to enforce spatio-temporal consistency, we propose a high-dimensional conditional random field (7D space-time-color space) that filters network predictions. We use variational inference to train both the base network and the conditional random field end-to-end.

5.1. Tesseract Kernel and Hybrid Kernel

The surface area of 3D data increases linearly to time and quadratically to the spatial resolution. However, when we use a conventional 4D hypercube, or a tesseract (Fig. 2), for a convolution kernel, the exponential increase in the number of parameters leads to over-parametrization, overfitting, as well as high computational-cost and memory consumption. Instead, we propose a hybrid kernel (non-hypercubic, non-permutohedral) to save computation. We use the arbitrary kernel offsets $\mathcal{N}^D$ of the generalized sparse convolution to implement the hybrid kernel.

The hybrid kernel is a combination of a cross-shaped kernel a conventional cubic kernel (Fig. 3). For spatial dimensions, we use a cubic kernel to capture the spatial geometry accurately. For the temporal dimension, we use the cross-shaped kernel to connect the same point in space across time. We experimentally show that the hybrid kernel outperforms the tesseract kernel both in speed and accuracy.

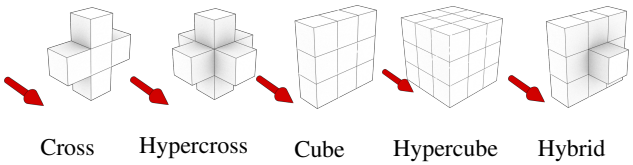

Figure 3: Various kernels in space-time. The red arrow indicates the temporal dimension and the other two axes are for spatial dimensions. The third spatial dimension is hidden for better visualization.

5.2. Residual Minkowski Networks

The generalized sparse convolution allows us to define strides and kernel shapes arbitrarily. Thus, we can create a high-dimensional network only with generalized sparse convolutions, making the implementation easier and generic. In addition, it allows us to adopt recent architectural innovations in 2D directly to high-dimensional networks. To demonstrate, we create a high-dimensional version of a residual network on Fig. 4. For the first layer, instead of a 7×7 2D convolution, we use a $5 \times 5 \times 5 \times 1$ generalized sparse convolution. However, for the rest of the networks, we follow the original network architecture.

For the U-shaped variants, we add multiple strided sparse convolutions and strided sparse transpose convolutions with skip connections connecting the layers with the same stride size (Fig. 5) on the base residual network. We use multiple

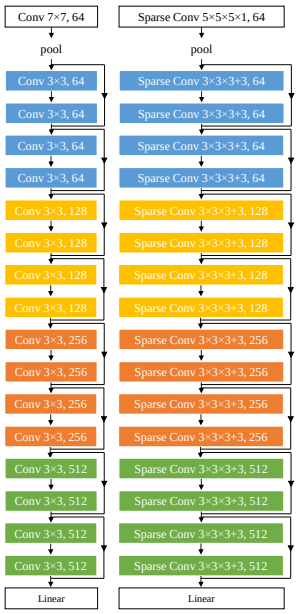

Figure 4: Architecture of ResNet18 (left) and MinkowskiNet18 (right). Note the structural similarity. $\times$ indicates a hypercubic kernel, $+$ indicates a hypercross kernel. (best viewed on display)

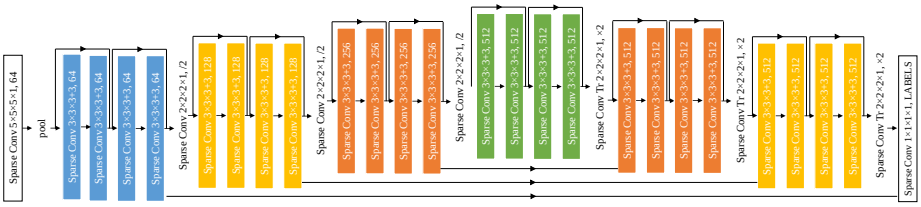

Figure 5: Architecture of MinkowskiUNet32. $\times$ indicates a hypercubic kernel, $+$ indicates a hypercross kernel. (best viewed on display)

variations of the same architecture for semantic segmentation experiments.

6. Trilateral Stationary-CRF

For semantic segmentation, the cross-entropy loss is applied for each pixel or voxel. However, the loss does not enforce consistency as it does not have pair-wise terms. To make such consistency more explicit, we propose a high-dimensional conditional random field (CRF) similar to the one used in image semantic segmentation [35]. In image segmentation, the bilateral space that consists of 2D space and 3D color is used for the CRF. For 3D-videos, we use the trilateral space that consists of 3D space, 1D time, and 3D chromatic space. The color space creates a "spatial" gap between points with different colors that are spatially adjacent (e.g., on a boundary). Thus, it prevents information from "leaking out" to different regions. Unlike conventional CRFs with Gaussian edge potentials and dense connections [13, 35], we do not restrict the compatibility function to be a Gaussian. Instead, we relax the constraint and only apply the stationarity condition.

To find the global optima of the distribution, we use the variational inference and convert a series of fixed point update equations to a recurrent neural network similar to [35]. We use the generalized sparse convolution in 7D space to implement the recurrence and jointly train both the base network that generates unary potentials and the CRF end-to-end.

6.1. Definition

Let a CRF node in the 7D (space-time-chroma) space be x_i and the unary potential be $\phi_u(x_i)$ and the pairwise potential as $\phi_p(x_i, x_j)$ where x_j is a neighbor of x_i, $\mathcal{N}^7(x_i)$. The conditional random field is defined as

$$P(\mathbf{X}) = \frac{1}{Z} \exp \sum_i \left(\phi_u(x_i) + \sum_{j \in \mathcal{N}^7(x_i)} \phi_p(x_i, x_j) \right)$$

where Z is the partition function; X is the set of all nodes; and ϕ_p must satisfy the *stationarity* condition $\phi_p(\mathbf{u}, \mathbf{v}) = \phi_p(\mathbf{u} + \tau_\mathbf{u}, \mathbf{v} + \tau_\mathbf{v})$ for $\tau_\mathbf{u}, \tau_\mathbf{v} \in \mathbb{R}^D$. Note that we use the camera extrinsics to define the spatial coordinates of a node x_i in the world coordinate system. This allows stationary points to have the same coordinates throughout the time.

6.2. Variational Inference

The optimization $\arg\max_\mathbf{X} P(X)$ is intractable. Instead, we use the variational inference to minimize divergence between the optimal $P(\mathbf{X})$ and an approximated distribution $Q(\mathbf{X})$. Specifically, we use the mean-field approximation, $Q = \prod_i Q_i(x_i)$ as the closed form solution exists. From the Theorem 11.9 in [12], Q is a local maximum if and only if

$$Q_i(x_i) = \frac{1}{Z_i} \exp \mathop{\mathbf{E}}_{\mathbf{X}_{-i} \sim Q_{-i}} \left[\phi_u(x_i) + \sum_{j \in \mathcal{N}^7(x_i)} \phi_p(x_i, x_j) \right].$$

$\mathbf{X}_{-i}$ and Q_{-i} indicate all nodes or variables except for the i-th one. The final fixed-point equation is Eq. 4. The derivation is in the supplementary material.

$$Q_i^+(x_i) = \frac{1}{Z_i} \exp \left\{ \phi_u(x_i) + \sum_{j \in \mathcal{N}^7(x_i)} \sum_{x_j} \phi_p(x_i, x_j) Q_j(x_j) \right\} \tag{4}$$

6.3. Learning with 7D Sparse Convolution

Interestingly, the weighted sum $\phi_p(x_i, x_j) Q_j(x_j)$ in Eq. 4 is equivalent to a generalized sparse convolution in the 7D space since ϕ_p is *stationary* and each edge between x_i, x_j can be encoded using $\mathcal{N}^7$. Thus, we convert fixed point update equation Eq. 4 into an algorithm in Alg. 5.

Algorithm 5 Variational Inference of TS-CRF

Require: Input: Logit scores ϕ_u for all x_i; associated coordinate C_i, color F_i, time T_i
$Q^0(X) = \exp \phi_u(X)$, $C_\text{crf} = [C, F, T]$
for n from 1 to N **do**
  $\tilde{Q}^n = \text{SparseConvolution}((C_\text{crf}, Q^{n-1}), \text{kernel}=\phi_p)$
  $Q^n = \text{Softmax}(\phi_u + \tilde{Q}^n)$
end for
return Q^N

Finally, we use ϕ_u as the logit predictions of a 4D Minkowski network and train both ϕ_u and ϕ_p *end-to-end* using one 4D and one 7D Minkowski network using Eq. 5.

$$\frac{\partial L}{\partial \phi_p} = \sum_n^N \frac{\partial L}{\partial Q^{n+}} \frac{\partial Q^{n+}}{\partial \phi_p}, \quad \frac{\partial L}{\partial \phi_u} = \sum_n^N \frac{\partial L}{\partial Q^{n+}} \frac{\partial Q^{n+}}{\partial \phi_u} \tag{5}$$

7. Experiments

To validate the proposed high-dimensional networks, we first use multiple standard 3D benchmarks for 3D semantic segmentation. It allows us to gauge the performance of the high-dimensional networks with the same architecture with other state-of-the-art methods. Next, we create multiple 4D datasets from 3D datasets that have temporal sequences and analyze each of the proposed components for ablation study.

7.1. Implementation

We implemented the Minkowski Engine (Sec. 4) using C++/CUDA and wrap it with PyTorch [21]. Data is prepared in parallel data processes that load point clouds, apply data augmentation, and quantize them with Alg. 1 on the fly. For non-spatial functions, we use the PyTorch functions directly.

7.2. Training and Evaluation

We use Momentum SGD with the Poly scheduler to train networks from learning rate 1e-1 and apply data augmentation including random scaling, rotation around the gravity axis, spatial translation, spatial elastic distortion, and chromatic translation and jitter.

For evaluation, we use the standard mean Intersection over Union (mIoU) and mean Accuracy (mAcc) for metrics following the previous works. To convert voxel-level predictions to point-level predictions, we simply propagated predictions from the nearest voxel center.

7.3. Datasets

ScanNet. The ScanNet [5] 3D segmentation benchmark consists of 3D reconstructions of real rooms. It contains 1.5k rooms, some repeated rooms captured with different sensors. We feed an entire room to a MinkowskiNet fully convolutionally without cropping.

Stanford 3D Indoor Spaces (S3DIS). The dataset [2] contains 3D scans of six floors of three different buildings. We use the Fold #1 split following many previous works. We use 5cm and 2cm voxel for the experiment.

RueMonge 2014 (Varcity). The RueMonge 2014 dataset [25] provides semantic labels for a multi-view 3D reconstruction of the Rue Mongue. To create a 4D dataset, we crop the 3D reconstruction on-the-fly to generate a temporal sequence. We use the official split for all experiments.

Synthia 4D. We use the Synthia dataset [27] to create 3D video sequences. We use 6 sequences of driving scenarios in 9 different weather conditions. Each sequence consists of 4 stereo RGB-D images taken from the top of a car. We back-project the depth images to the 3D space to create 3D videos. We visualized a part of a sequence in Fig. 1.

We use the sequence 1-4 except for sunset, spring, and fog for the train split; the sequence 5 foggy weather for validation; and the sequence 6 sunset and spring for the test.

Table 1: 3D Semantic Label Benchmark on ScanNet† [5]

Method	mIOU
ScanNet [5]	30.6
SSC-UNet [10]	30.8
PointNet++ [23]	33.9
ScanNet-FTSDF	38.3
SPLATNet [28]	39.3
TangetConv [29]	43.8
SurfaceConv [20]	44.2
3DMV‡ [6]	48.4
3DMV-FTSDF‡	50.1
PointNet++SW	52.3
MinkowskiNet42 (5cm)	**67.9**
MinkowskiNet42 (2cm)†	72.1
SparseConvNet [10]†	**72.5**

†: post-CVPR submissions. ‡: uses 2D images additionally. Per class IoU in the supplementary material. The parenthesis next to our methods indicate the voxel size.

In total, the train/val/test set contain 20k/815/1886 3D scenes respectively.

Since the dataset is purely synthetic, we added various noise to the input point clouds to simulate noisy observations. We used elastic distortion, Gaussian noise, and chromatic shift in the color for the noisy 4D Synthia experiments.

7.4. Results and Analysis

Table 2: Segmentation results on the 4D Synthia dataset

Method	mIOU	mAcc
3D MinkNet20	76.24	89.31
3D MinkNet20 + TA	77.03	89.20
4D Tesseract MinkNet20	75.34	89.27
4D MinkNet20	77.46	88.013
4D MinkNet20 + TS-CRF	78.30	90.23
4D MinkNet32 + TS-CRF	**78.67**	**90.51**

TA denotes temporal averaging. Per class IoU in the supplementary material.

ScanNet & Stanford 3D Indoor The ScanNet and the Stanford Indoor datasets are one of the largest non-synthetic datasets, which make the datasets ideal test beds for 3D segmentation. We were able to achieve +19% mIOU on ScanNet, and +7% on Stanford compared with the best-published works by the CVPR deadline. This is due to the depth of the networks and the fine resolution of the space. We trained the same network for 60k iterations with 2cm voxel and achieved 72.1% mIoU on ScanNet after the deadline. For all evaluation, we feed an entire room to a network and process it fully convolutionally.

Table 3: Segmentation results on the noisy Synthia 4D dataset

IoU	Building	Road	Sidewalk	Fence	Vegetation	Pole	Car	Traffic Sign	Pedestrian	Lanemarking	Traffic Light	mIoU
3D MinkNet42	87.954	97.511	78.346	84.307	96.225	94.785	87.370	42.705	**66.666**	52.665	55.353	76.717
3D MinkNet42 + TA	87.796	97.068	78.500	83.938	96.290	94.764	85.248	43.723	62.048	50.319	54.825	75.865
4D Tesseract MinkNet42	89.957	96.917	81.755	82.841	96.556	96.042	91.196	52.149	51.824	70.388	57.960	78.871
4D MinkNet42	**88.890**	**97.720**	**85.206**	**84.855**	**97.325**	**96.147**	**92.209**	**61.794**	61.647	**55.673**	**56.735**	**79.836**

TA denotes temporal averaging. As the input pointcloud coordinates are noisy, averaging along the temporal dimension introduces noise.

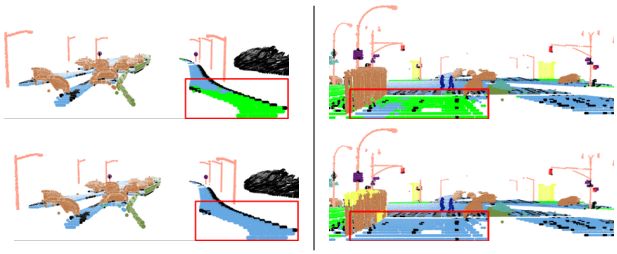

Figure 6: Visualizations of 3D (top), and 4D networks (bottom) on Synthia. A road (blue) far away from the car is often confused as sidewalks (green) with a 3D network, which persists after temporal averaging. However, 4D networks accurately captured it.

Figure 7: Visualization of Scannet predictions. From the top, a 3D input pointcloud, a network prediction, and the ground-truth.

4D analysis The RueMongue dataset is a small dataset that ranges one section of a street, so with the smallest network, we were able to achieve the best result (Tab. 5). However, the results quickly saturate. On the other hand, the Synthia 4D dataset has an order of magnitude more 3D scans than any other datasets, so it is more suitable for the ablation study.

We use the Synthia datasets with and without noise for 3D and 4D analysis and results are presented in Tab. 2 and Tab. 3. We use various 3D and 4D networks with and without TS-CRF. Specifically, when we simulate noise in sensory inputs on the 4D Synthia dataset, we can observe that the 4D networks are more robust to noise. Note that the number of parameters added to the 4D network compared with the 3D network is less than 6.4 % and 6e-3 % for the TS-CRF. Thus, with a small increase in computation, we could get

Table 4: Stanford Area 5 Test (Fold #1) (S3DIS) [2]

Method	mIOU	mAcc
PointNet [22]	41.09	48.98
SparseUNet [9]	41.72	64.62
SegCloud [30]	48.92	57.35
TangentConv [29]	52.8	60.7
3D RNN [32]	53.4	71.3
PointCNN [15]	57.26	63.86
SuperpointGraph [14]	58.04	66.5
MinkowskiNet20	62.60	69.62
MinkowskiNet32	**65.35**	**71.71**

Per class IoU in the supplementary material.

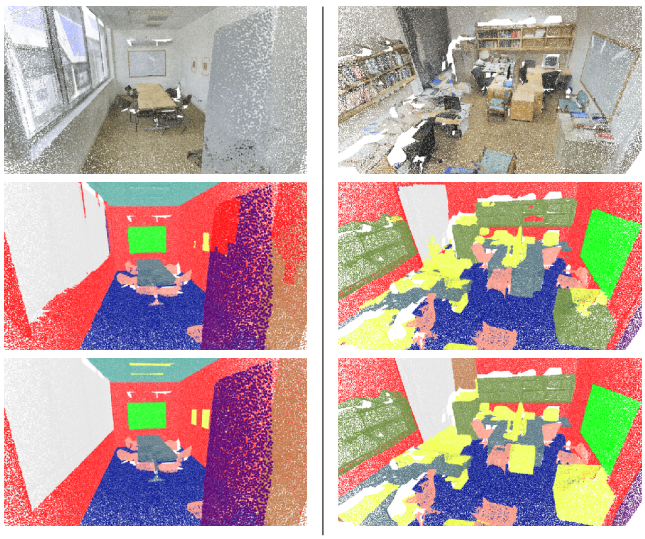

Figure 8: Visualization of Stanford dataset Area 5 test results. From the top, RGB input, prediction, ground truth.

Table 5: RueMonge 2014 dataset (Varcity) TASK3 [25]

Method	mIOU
MV-CRF [26]	42.3
Gradde et al. [7]	54.4
RF+3D CRF [17]	56.4
OctNet (256^3) [24]	59.2
SPLATNet (3D) [28]	65.4
3D MinkNet20	66.46
4D MinkNet20	66.56
4D MinkNet20 + TS-CRF	**66.59**

The performance saturates quickly due to the small training set. Per class IoU in the supplementary material.

a more robust algorithm with higher accuracy. In addition, when we process temporal sequence using the 4D networks, we could even get small speed gain as we process data in a batch mode. On Tab. 6, we vary the voxel size and the sequence length and measured the runtime of the 3D and 4D networks, as well as the 4D networks with TS-CRF. Note that for large voxel sizes, we tend to get small speed gain on the 4D networks compared with 3D networks.

Table 6: Time (s) to process 3D videos with 3D and 4D MinkNet, the volume of a scan at each time step is 50m$\times$ 50m $\times$ 50m

Voxel Size	0.6m			0.45m			0.3m		
Video Length (s)	3D	4D	4D-CRF	3D	4D	4D-CRF	3D	4D	4D-CRF
3	0.18	0.14	0.17	0.25	0.22	0.27	0.43	0.49	0.59
5	0.31	0.23	0.27	0.41	0.39	0.47	0.71	0.94	1.13
7	0.43	0.31	0.38	0.58	0.61	0.74	0.99	1.59	2.02

8. Conclusion

In this paper, we propose a generalized sparse convolution and an auto-differentiation library for sparse tensors and the generalized sparse convolution. Using these, we create 4D convolutional neural networks for spatio-temporal perception. Experimentally, we show that 3D convolutional neural networks alone can outperform 2D networks and 4D perception can be more robust to noise.

9. Acknowledgements

Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. We acknowledge the support of the System X Fellowship and the companies sponsored: NEC Corporation, Nvidia, Samsung, and Tencent. Also, we want to acknowledge the academic hardware donation from Nvidia.

References

[1] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast high-dimensional filtering using the permutohedral lattice. In *Computer Graphics Forum*, volume 29, pages 753–762. Wiley Online Library, 2010. 2

[2] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016. 2, 7, 8

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018. 3

[4] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 2

[5] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 1, 2, 7

[6] Angela Dai and Matthias Nießner. 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2, 7

[7] Raghudeep Gadde, Varun Jampani, Renaud Marlet, and Peter Gehler. Efficient 2d and 3d facade segmentation using auto-context. *IEEE transactions on pattern analysis and machine intelligence*, 2017. 9

[8] Benjamin Graham. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014. 1, 3

[9] Ben Graham. Sparse 3d convolutional neural networks. *British Machine Vision Conference*, 2015. 1, 2, 3, 8

[10] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018. 7

[11] P. Hermosilla, T. Ritschel, P-P Vazquez, A. Vinacua, and T. Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2018)*, 2018. 1, 2

[12] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. 6

[13] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in Neural Information Processing Systems 24*, 2011. 6

[14] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. *arXiv preprint arXiv:1711.09869*, 2017. 2, 8

[15] Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. Pointcnn. *arXiv preprint arXiv:1801.07791*, 2018. 1, 2, 8

[16] Maria Lorenzo-Valdés, Gerardo I Sanchez-Ortiz, Andrew G Elkington, Raad H Mohiaddin, and Daniel Rueckert. Segmentation of 4d cardiac mr images using a probabilistic atlas and the em algorithm. *Medical Image Analysis*, 8(3):255–265, 2004. 3

[17] Andelo Martinovic, Jan Knopp, Hayko Riemenschneider, and Luc Van Gool. 3d all the way: Semantic segmentation of urban scenes from start to end in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015. 9

[18] Tim McInerney and Demetri Terzopoulos. A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4d image analysis. *Computerized Medical Imaging and Graphics*, 19(1):69–83, 1995. 2

[19] Nvidia. Thrust: Parallel algorithm library. 3

[20] Hao Pan, Shilin Liu, Yang Liu, and Xin Tong. Convolutional neural networks on 3d surfaces using parallel frames. *arXiv preprint arXiv:1808.04952*, 2018. 1, 2, 7

[21] 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. 2017. 7

[22] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016. 1, 2, 8

[23] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, 2017. 1, 2, 7

[24] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 1, 2, 9

[25] Hayko Riemenschneider, András Bódis-Szomorú, Julien Weissenberg, and Luc Van Gool. Learning where to classify in multi-view semantic segmentation. In *European Conference on Computer Vision*. Springer, 2014. 2, 7, 9

[26] Hayko Riemenschneider, András Bódis-Szomorú, Julien Weissenberg, and Luc Van Gool. Learning where to classify in multi-view semantic segmentation. In *European Conference on Computer Vision*, 2014. 9

[27] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2, 7

[28] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Vangelis Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. *arXiv preprint arXiv:1802.08275*, 2018. 2, 7, 9

[29] Maxim Tatarchenko*, Jaesik Park*, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3D. *CVPR*, 2018. 1, 2, 7, 8

[30] Lyne P Tchapmi, Christopher B Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. *International Conference on 3D Vision (3DV)*, 2017. 2, 8

[31] Parker Allen Tew. *An investigation of sparse tensor formats for tensor libraries*. PhD thesis, Massachusetts Institute of Technology, 2016. 3

[32] Xiaoqing Ye, Jiamao Li, Hexiao Huang, Liang Du, and Xiaolin Zhang. 3d recurrent neural networks with context fusion for point cloud semantic segmentation. In *The European Conference on Computer Vision (ECCV)*, September 2018. 8

[33] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3dmatch: Learning the matching of local 3d geometry in range scans. In *CVPR*, 2017. 2

[34] Yu Zhao, Xiang Li, Wei Zhang, Shijie Zhao, Milad Makkie, Mo Zhang, Quanzheng Li, and Tianming Liu. Modeling 4d fmri data via spatio-temporal convolutional neural networks (st-cnn). *arXiv preprint arXiv:1805.12564*, 2018. 3

[35] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, 2015. 6

The supplementary material for 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks

1. Minkowski Engine

In this section, we provide detailed breakdowns of few algorithms in the main paper as well as the coordinate initialization and the hash function. Please note that the actual implementation may not faithfully reflect the algorithms presented here for optimization.

1.1. Sparse Quantization

First, we provide line-by-line comments for the sparse quantization algorithm. Overall, the algorithm finds unique voxels and creates labels associated with them. If there are different labels within the same voxel, we assign the `IGNORE_LABEL`. This allows the network to ignore the cross-entropy loss for those voxels during training.

The Alg. 1 starts by converting all coordinates to hash keys and generating a sequence $\mathbf{i}$, which is used to identify the original indices. Next, we sort the sequence, label pairs by the hash key (L5). This will place indices and labels that fall into the same cell to be adjacent in the GPU memory. Next, we reduce the index sequence with the key-label pair (L6). This step will collapse all points with the same labels. However, if there are different labels within the same cell, we will have at least two key-label pairs in the same cell. Thus, in L7, we use the reduction function $f((l_x, i_x), (l_y, i_y)) => (\texttt{IGNORE\_LABEL}, i_x)$ will simply assign `IGNORE_LABEL` to the cell. Finally, we use the reduced unique indices $\mathbf{i}'''$ to return unique coordinates, features, and transformed labels.

Algorithm 1 GPU Sparse Tensor Quantization

1: Inputs: coordinates $C_p \in \mathbb{R}^{N \times D}$, features $F_p \in \mathbb{R}^{N \times N_f}$, target labels $\mathbf{l} \in \mathbb{Z}_+^N$, quantization step size v_l
2: $C'_p \leftarrow$ floor(C_p / v_l)
3: $\mathbf{k} \leftarrow$ hash(C'_p)
4: $\mathbf{i} \leftarrow$ sequence(N)
5: $((\mathbf{i}', \mathbf{l}'), k') \leftarrow$ SortByKey(($\mathbf{i}, \mathbf{l}$), key=$\mathbf{k}$)
6: $(\mathbf{i}'', (\mathbf{k}'', \mathbf{l}'')) \leftarrow$ UniqueByKey($\mathbf{i}'$, key=($\mathbf{k}', \mathbf{l}'$))
7: $(\mathbf{l}''', \mathbf{i}''') \leftarrow$ ReduceByKey(($\mathbf{l}'', \mathbf{i}''$), key=$\mathbf{k}''$, fn=$f$)
8: **return** $C'_p[\mathbf{i}''', :], F_p[\mathbf{i}''', :], \mathbf{l}'''$

1.2. Coordinate Hashing

We used a variation of Fowler–Noll–Vo hash function, *FNV64-1A*, and modified it for coordinates in `int32`. We list the algorithm in Alg. 2.

1.3. Creating output coordinates

The first step in the generalized sparse convolution is defining the output coordinates $\mathcal{C}^{\text{out}}$. In most cases, we use the same coordinates for the output coordinates. However, when we use a strided convolution or a strided pooling, we use the Alg. 3 to create new output coordinates. For transposed strided convolutions, we use the same coordinates with the same input-stride to preserve the order of the coordinates.

Algorithm 2 Coordinate Hashing

Require: Inputs: coordinate $\mathbf{c} \in \mathbb{Z}^D$ in `int32`

function HASH($\mathbf{c}$)
  h = UINT64(14695981039346656037)
  for all $\mathbf{c}_i \in \mathbf{c}$ **do**
    $h = h \oplus$ UINT32(c_i)
    $h = h \times$ UINT64(1099511628211)
  end for
  return h
end function

Algorithm 3 Create output coordinates

Require: input stride s_i, layer stride s_l, input coordinates $\mathbf{C}$

function CREATEOUTPUTCOORDINATES($\mathbf{C}$, s_i, s_l)
  if $s_l > 1$ **then**
    $s \leftarrow s_l \times s_i$
    $\mathbf{C}' \leftarrow \{\}$ // Create an empty set
    for all $\mathbf{c}_i \in \mathbf{C}$ **do**
      $\mathbf{c} \leftarrow \lfloor \mathbf{c}_i / s \rfloor \times s$
      if $\mathbf{c}$ is not in $\mathbf{C}'$ **then**
        Append $\mathbf{c}$ to $\mathbf{C}'$
      end if
    end for
    return $\mathbf{C}'$
  else
    return $\mathbf{C}$
  end if
end function

2. Derivation of the Trilateral Stationary-Conditional Random Field Fixed Point Update Equation

The TS-CRF optimization $\arg\max_{\mathbf{X}} P(X)$ is intractable. Instead, we use the variational inference to minimize divergence between the optimal $P(\mathbf{X})$ and an approximated distribution $Q(\mathbf{X})$. Commonly, the divergence is measured using the I-projection: $\mathbf{D}(Q\|P)$, the number of bits lost when coding a distribution Q using P. To simplify the approximated distribution, we use the mean-field approximation, $Q = \prod_i Q_i(x_i)$ as the closed form solution exists. From the Theorem 11.9 in [4], Q is a local maximum if and only if

$$Q_i(x_i) = \frac{1}{Z_i} \exp \mathop{\mathbf{E}}_{\mathbf{X}_{-i} \sim Q_{-i}} \left[\phi_u(x_i) + \sum_{j \in \mathcal{N}(x_i)} \phi_p(x_i, x_j) \right] \tag{1}$$

. where $\mathbf{X}_{-i}$ and Q_{-i} indicate all variables except for the i-th variable.

$$Q_i(x_i) = \frac{1}{Z_i} \exp \mathop{\mathbf{E}}_{\mathbf{x}_{-i} \sim Q_{-i}} \left[\phi_u(x_i) + \sum_{j \in \mathcal{N}(x_i)} \phi_p(x_i, x_j) \right] \tag{2}$$

$$= \frac{1}{Z_i} \exp \left[\mathop{\mathbf{E}}_{\mathbf{x}_{-i} \sim Q_{-i}} \phi_u(x_i) + \mathop{\mathbf{E}}_{\mathbf{x}_{-i} \sim Q_{-i}} \sum_{j \in \mathcal{N}(x_i)} \phi_p(x_i, x_j) \right] \tag{3}$$

$$= \frac{1}{Z_i} \exp \left[\phi_u(x_i) + \sum_{j \in \mathcal{N}(x_i)} \mathbf{E}_{j \in \mathcal{N}(x_i)} \phi_p(x_i, x_j) \right] \tag{4}$$

$$= \frac{1}{Z_i} \exp \left[\phi_u(x_i) + \sum_{j \in \mathcal{N}(x_i)} \sum_{x_j} \phi_p(x_i, x_j) Q_j(x_j) \right] \tag{5}$$

Thus, the final fixed-point update equation is

$$Q_i^+(x_i) = \frac{1}{Z_i} \exp \left\{ \phi_u(x_i) + \sum_{j \in \mathcal{N}(x_i)} \sum_{x_j} \phi_p(x_i, x_j) Q_j(x_j) \right\} \tag{6}$$

.

3. Experiments and Analysis

We present per-class IoU numbers for all experiments and more qualitative results. We also visualize the entire RueMongue 2014 (Varcity) dataset on Fig. 1. The dataset is quite small for deep learning so the Minkowski Networks results are all saturated around 66% mIoU.

Figure 1: Visualization of the RueMonge 2014 dataset ground-truth. The left half of the pointcloud is the training set and the right half is the test set (Black: `IGNORE_LABEL`).

Table 1: ScanNet [1] 3D Segmentation Benchmark Results

Method	bath	bed	bksf	cab	chair	cntr	curt	desk	door	floor	othr	pic	ref	show	sink	sofa	tab	toil	wall	wind	**mIoU**
ScanNet [1]	20.3	36.6	50.1	31.1	52.4	21.1	0.2	34.2	18.9	78.6	14.5	10.2	24.5	15.2	31.8	34.8	30.0	46.0	43.7	18.2	30.6
SSC-UNet [3]	35.3	29.0	27.8	16.6	55.3	16.9	28.6	14.7	14.8	90.8	18.2	6.4	2.3	1.8	35.4	36.3	34.5	54.6	68.5	27.8	30.8
PointNet++ [9]	58.4	47.8	45.8	25.6	36.0	25.0	24.7	27.8	26.1	67.7	18.3	11.7	21.2	14.5	36.4	34.6	23.2	54.8	52.3	25.2	33.9
ScanNet+SDF	29.7	49.1	43.2	35.8	61.2	27.4	11.6	41.1	26.5	90.4	22.9	7.9	25.0	18.5	32.0	51.0	38.5	54.8	59.7	39.4	38.3
SPLATNet [11]	47.2	51.1	60.6	31.1	65.6	24.5	40.5	32.8	19.7	92.7	22.7	0.0	0.1	24.9	27.1	51.0	38.3	59.3	69.9	26.7	39.3
TangetConv [12]	43.7	64.6	47.4	36.9	64.5	35.3	25.8	28.2	27.9	91.8	29.8	14.7	28.3	29.4	48.7	56.2	42.7	61.9	63.3	35.2	43.8
SurfaceConv [7]	50.5	62.2	38.0	34.2	65.4	22.7	39.7	36.7	27.6	92.4	24.0	19.8	35.9	26.2	36.6	58.1	43.5	64.0	66.8	39.8	44.2
3DMV [2]	48.4	53.8	64.3	42.4	60.6	31.0	57.4	43.3	37.8	79.6	30.1	21.4	53.7	20.8	47.2	50.7	41.3	69.3	60.2	53.9	48.4
3DMV-FTSDF	55.8	60.8	42.4	47.8	69.0	24.6	58.6	46.8	45.0	91.1	39.4	16.0	43.8	21.2	43.2	54.1	47.5	74.2	72.7	47.7	50.1
MinkowskiNet42 (5cm)	81.1	73.4	73.9	64.1	80.4	41.3	75.9	69.6	54.5	93.8	51.8	14.1	62.3	75.7	68.0	72.3	68.4	89.6	82.1	65.1	67.9
MinkowskiNet42 (2cm)	83.7	80.4	80.0	72.1	84.3	46.0	83.5	64.7	59.7	95.3	54.2	21.4	74.6	91.2	70.5	77.1	64.0	87.6	84.2	67.2	72.1

Test labels not available publicly. All numbers from http://kaldir.vc.in.tum.de/scannet_benchmark/. Entries without citation are from unpublished works. ‡: uses 2D images additionally.

Table 2: Results on the Stanford 3D Indoor Spaces Dataset Area 5 Test (Fold #1) (S3DIS)

Method	ceiling	floor	wall	beam	clmn	windw	door	chair	table	bkcase	sofa	board	clutter	**mIOU**	**mAcc**
PointNet [8]	88.80	97.33	69.80	0.05	3.92	46.26	10.76	52.61	58.93	40.28	5.85	26.38	33.22	41.09	48.98
SegCloud [13]	90.06	96.05	69.86	0.00	18.37	38.35	23.12	75.89	70.40	58.42	40.88	12.96	41.60	48.92	57.35
TangentConv* [12]	90.47	97.66	73.99	0.0	20.66	38.98	31.34	77.49	69.43	57.27	38.54	48.78	39.79	52.8	60.7
3D RNN [15]	95.2	98.6	77.4	0.8	9.83	52.7	27.9	76.8	78.3	58.6	27.4	39.1	51.0	53.4	71.3
PointCNN [6]	92.31	98.24	79.41	0.00	17.60	22.77	62.09	80.59	74.39	66.67	31.67	62.05	56.74	57.26	63.86
SuperpointGraph [5]	89.35	96.87	78.12	0.0	42.81	48.93	61.58	84.66	75.41	69.84	52.60	2.1	52.22	58.04	66.5
PCCN [14]	90.26	96.20	75.89	0.27	5.98	69.49	63.45	66.87	65.63	47.28	68.91	59.10	46.22	58.27	67.01
MinkowskiNet20	91.55	98.49	84.99	0.8	26.47	46.18	55.82	88.99	80.52	71.74	48.29	62.98	57.72	62.60	69.62
MinkowskiNet32	91.75	98.71	86.19	0.0	34.06	48.90	62.44	89.82	81.57	74.88	47.21	74.44	58.57	**65.35**	**71.71**

*Data from the authors

Table 3: Semantic segmentation results on the 4D Synthia dataset

Method	Bldn	Road	Sdwlk	Fence	Vegittn	Pole	Car	T. Sign	Pedstrn	Bicycl	Lane	T. Light	mIOU	mAcc
MinkNet20	89.394	97.684	69.425	86.519	98.106	97.256	93.497	79.450	92.274	0.000	44.609	66.691	76.24	89.31
MinkNet20 + TA	88.096	97.790	78.329	87.088	96.540	97.486	94.203	78.831	92.489	0.000	46.407	67.071	77.03	89.198
4D MinkNet20	90.125	98.262	73.467	87.187	99.099	97.502	94.010	79.041	92.622	0.000	50.006	68.138	77.46	88.013
4D Tesseract MinkNet20	89.346	97.291	60.712	86.451	98.000	96.632	93.191	74.906	91.030	0.000	47.113	69.343	75.335	89.272
4D MinkNet20 + TS-CRF	89.438	98.291	79.938	86.254	98.707	97.142	95.045	81.592	91.540	0.000	54.596	67.067	78.30	90.23
4D MinkNet32 + TS-CRF	89.694	98.632	86.893	87.801	98.776	97.284	94.039	80.292	92.300	0.000	49.299	69.060	**78.67**	**90.51**

Table 4: Semantic segmentation results on the RueMongue [10] (Varcity) dataset

Method	window	wall	balcony	door	roof	sky	shop	mIOU
3D MinkNet20	61.978	80.813	64.303	47.256	61.486	76.901	72.498	66.462
4D MinkNet20	61.938	81.591	65.394	35.089	68.554	80.546	72.846	66.565
4D MinkNet20 + TS-CRF	62.600	81.821	63.599	40.795	66.229	80.464	70.639	66.592

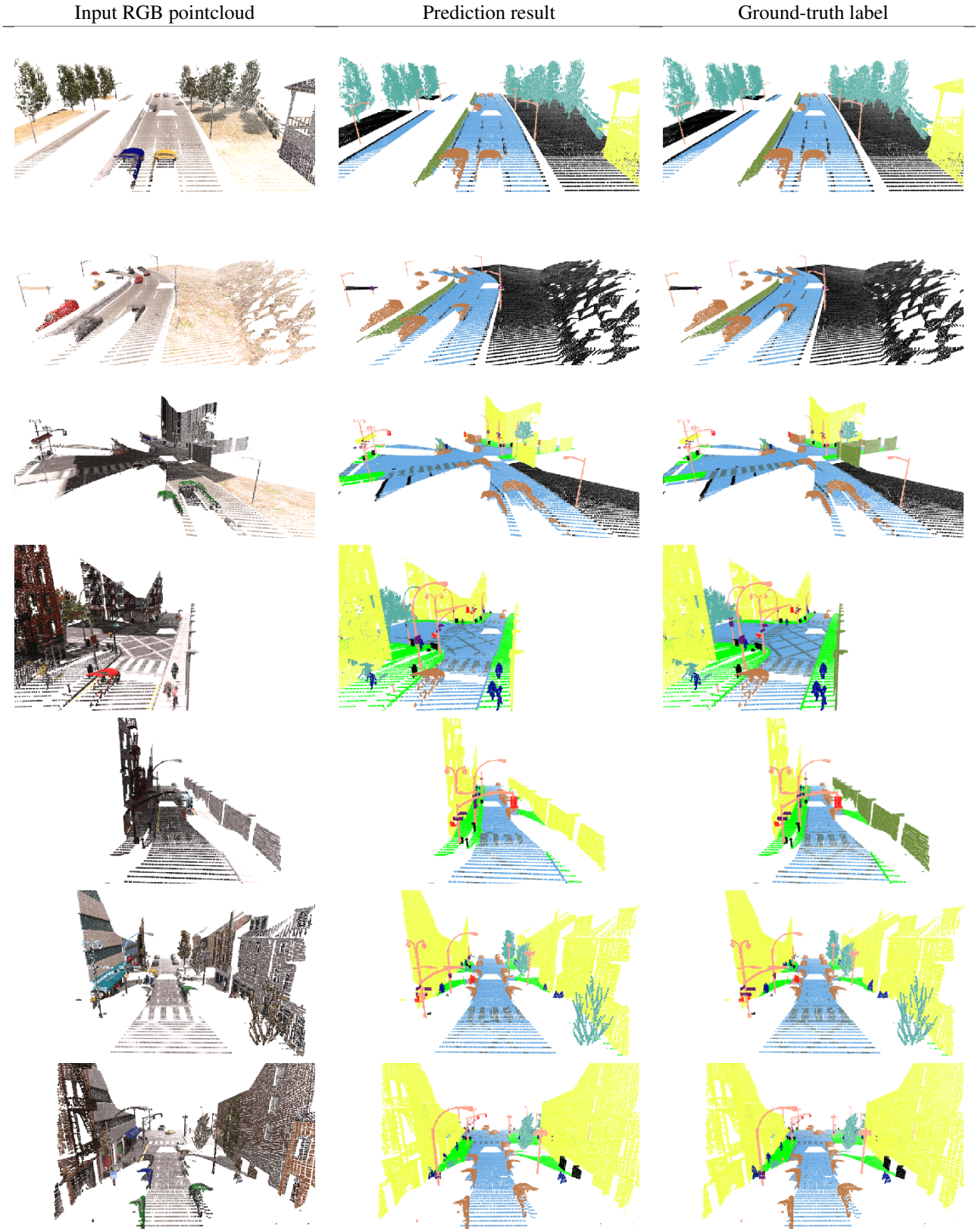

Figure 2: Visualization of the 4D Synthia predictions and ground truth labels.

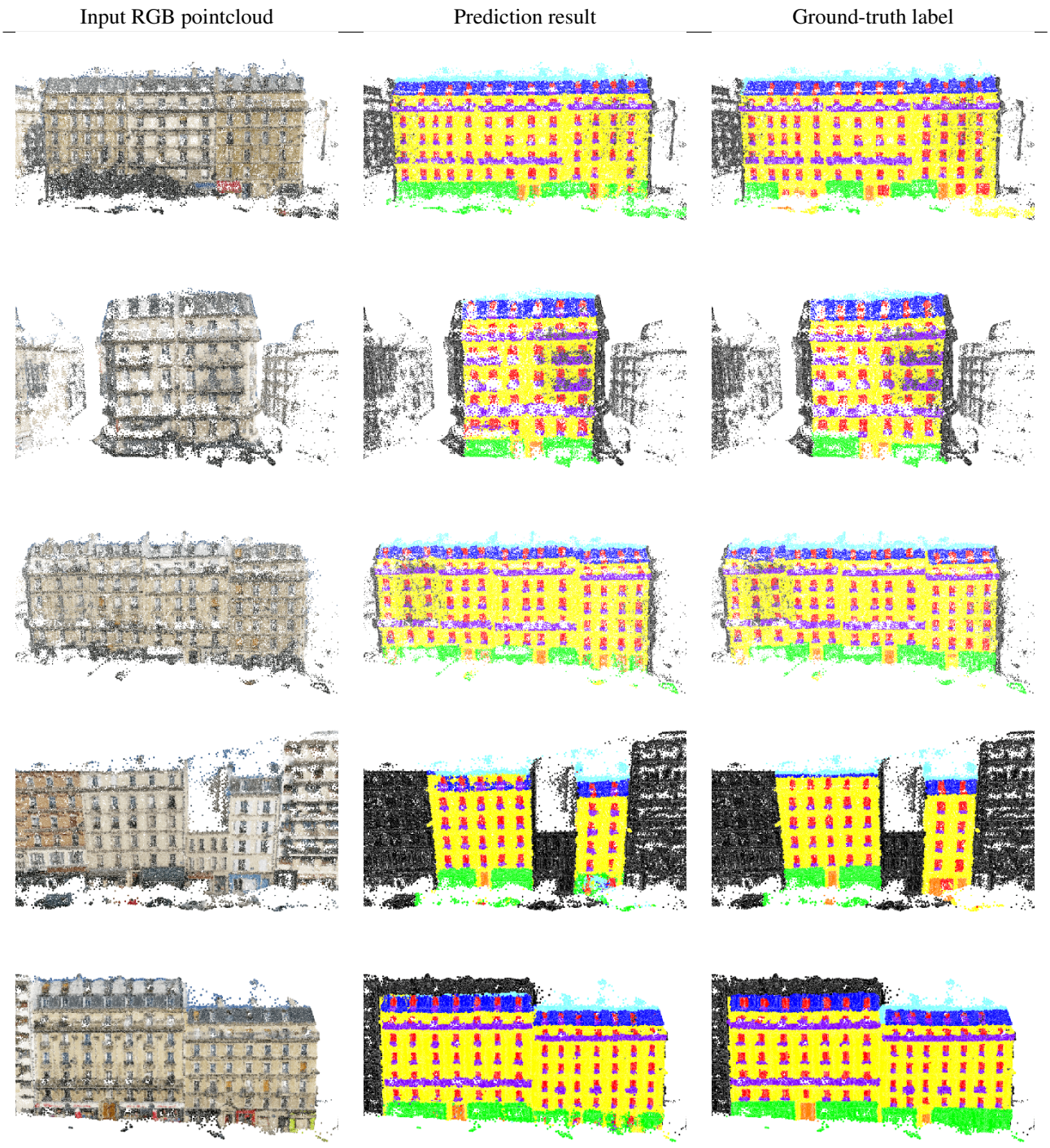

Figure 3: Visualization of the RueMonge 2014 dataset TASK3 results.

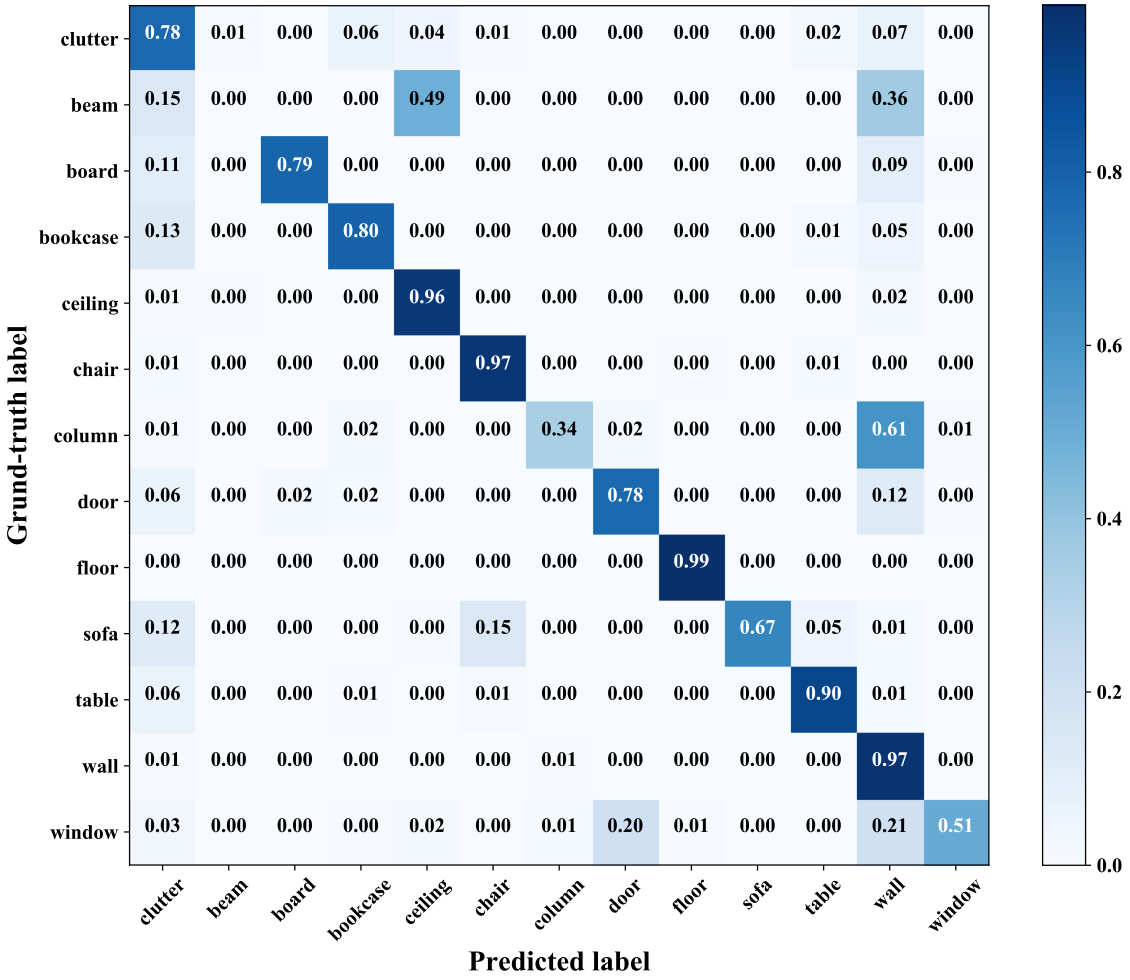

Figure 4: The confusion matrix of the MinkowskiNet32 predictions on the Stanford dataset Area 5 (Fold #1).

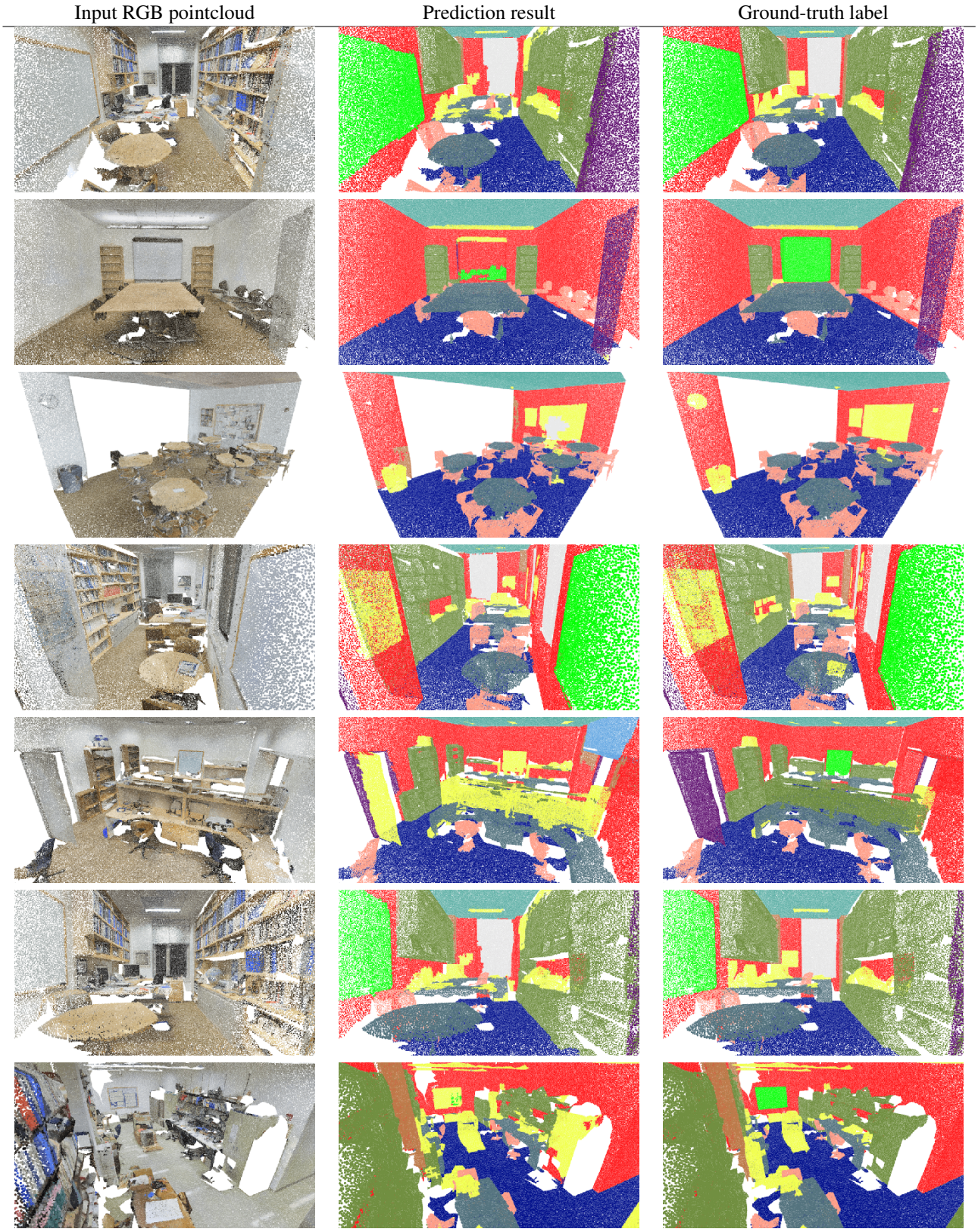

Figure 5: Visualization of the Stanford dataset Area 5 test result.

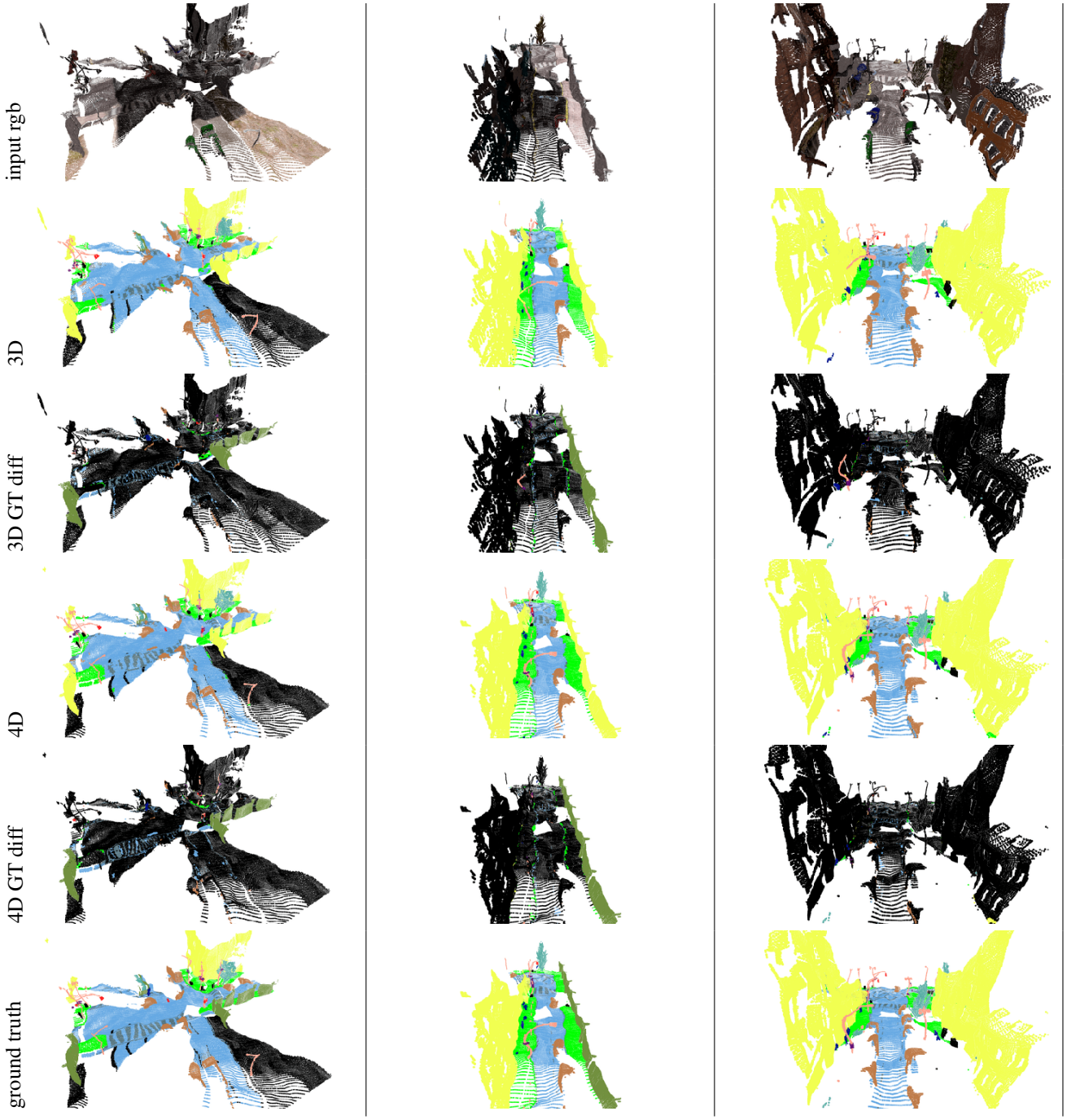

Figure 6: Visualization of the noisy 4D Synthia dataset and predictions.

Figure 7: Visualization of MinkNet predictions on the Scannet validation set.

References

[1] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 3

[2] Angela Dai and Matthias Nießner. 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 3

[3] Ben Graham. Sparse 3d convolutional neural networks. *British Machine Vision Conference*, 2015. 3

[4] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. 2

[5] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. *arXiv preprint arXiv:1711.09869*, 2017. 4

[6] Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. Pointcnn. *arXiv preprint arXiv:1801.07791*, 2018. 4

[7] Hao Pan, Shilin Liu, Yang Liu, and Xin Tong. Convolutional neural networks on 3d surfaces using parallel frames. *arXiv preprint arXiv:1808.04952*, 2018. 3

[8] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016. 4

[9] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, 2017. 3

[10] Hayko Riemenschneider, András Bódis-Szomorú, Julien Weissenberg, and Luc Van Gool. Learning where to classify in multi-view semantic segmentation. In *European Conference on Computer Vision*. Springer, 2014. 4

[11] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Vangelis Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. *arXiv preprint arXiv:1802.08275*, 2018. 3

[12] Maxim Tatarchenko*, Jaesik Park*, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3D. *CVPR*, 2018. 3, 4

[13] Lyne P Tchapmi, Christopher B Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. *International Conference on 3D Vision (3DV)*, 2017. 4

[14] Shenlong Wang, Simon Suo, Wei-Chiu Ma3 Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018. 4

[15] Xiaoqing Ye, Jiamao Li, Hexiao Huang, Liang Du, and Xiaolin Zhang. 3d recurrent neural networks with context fusion for point cloud semantic segmentation. In *The European Conference on Computer Vision (ECCV)*, September 2018. 4