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 input sources (a sequence of depth images, or LIDAR scans). However, in many cases, the 3D-videos are processed frame-by-frame either through 2D convnets or 3D perception algorithms. 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 generalized sparse convolutions that encompass all discrete convolutions. To implement the generalized sparse convolution, we create an open-source auto-differentiation library for sparse tensors[1] *that provides extensive functions for high-dimensional convolutional neural networks. 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 4D space, we propose the hybrid kernel, a special case of the generalized sparse convolution, and trilateral-stationary conditional random fields that enforce spatio-temporal consistency in the 7D space-time-chroma space. Experimentally, we show that a convolutional neural network with only generalized 3D 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 and outperform the 3D convolutional neural network.*

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-vidoes became readily-available sources of input for robotics systems or AR/VR applications.

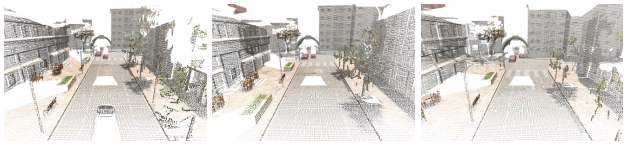

Figure 1: An example of 3D video: 3D scenes at different time steps. Best viewed on display.

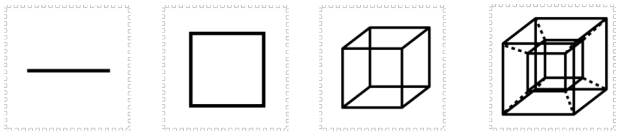

Figure 2: 2D projections of hypercubes in various dimensions

However, there are many technical challenges in using 3D-videos for high-level perception tasks. First, 3D data requires a heterogeneous representation and processing that 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, the limited number of open-source libraries for fast large-scale 3D data is another factor.

To resolve most, if not all, of the challenges in high-dimensional perception, we adopt a sparse tensor [8, 9] for our problem and propose the generalized sparse convolutions for sparse tensors and publish an open-source auto-diff library for sparse tensors with comprehensive standard neural network functions.

We adopt the sparse representation for a few reasons. Currently, there are various concurrent works for 3D perception: a dense 3D convolution [5], pointnet-variants [23, 24], continuous convolutions [12, 16], surface convolutions [21, 30], and an octree convolution [25]. Out of these representations, we chose a sparse tensor due to its expressiveness and generalizability for high-dimensional spaces. Also, it allows 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

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

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

2D perception as well as 3D reconstruction [4], feature learning [34], and semantic segmentation [5]. As the generalized sparse convolution is a direct high-dimensional extension of the standard 2D convolution, we can re-purpose all architectural innovations such as residual connections, batch normalization, and many others with little to no modification for high-dimensional problems.

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.

Using the proposed library, we create the first large-scale 3D/4D networks[3] and named them Minkowski networks 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 3D, and 625 in 4D (Fig. 2). This exponential increase, however, does not necessarily translate to better performance and slows down the network significantly. To overcome this challenge, we propose custom kernels with non-(hyper)-cubic shapes.

Finally, the 4D spatio-temporal predictions are not necessarily consistent throughout the space and time. To enforce consistency, we propose the conditional random fields defined in a 7D trilateral space (space-time-color) with a stationary consistency function. We use variational inference to convert the conditional random field to differentiable recurrent layers which can be implemented in as a 7D Minkowski network 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 [28, 26]. First, we show that a pure 3D method without a 2D convolutional neural network can outperform 2D or hybrid deep-learning algorithms by a large margin.[4] Also, we create 4D datasets from Synthia [28] and Varcity [26] and report ablation studies of temporal components.

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 [31, 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 [25] proposed to use the Octree structure to represent 3D space and convolution on it.

The second branch is sparse 3D-convolutional neural networks [29, 9]. There are two quantization methods used for high dimensions: a rectangular grid and a permutohedral lattice [1]. [29] 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 [12, 16]. 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, [21, 30] proposed to use 2D convolutions on the surface for semantic segmentation.

Another direction is PointNet-based methods [23, 24]. 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. [15] tried to resolve such shortcomings with a recurrent network on top of multiple pointnets, and [16] 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 [19] proposed a dynamic deformable balloon model for 4D cardiac image analysis. Later, [17] used a 4D Markov Random Fields for cardiac segmentation. Recently, [35] combined a 3D-UNet for spatial data with a 1D-AutoEncoder for temporal data and applied the model for auto-encoding brain fMRI images.

In this paper, we propose the first convolutional neural networks for high-dimensional spaces including the 4D

[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.

spatio-temporal data, or 3D videos. Compared with other approaches that combine temporal data with a recurrent neural network or a shallow model, our networks use a homogeneous representation, convolutions, and other neural network consistently throughout the networks. Specifically, convolutions are proven to be effective in numerous 2D/3D spatial perception as well as temporal or sequence modeling [3].

3. Sparse Tensor and Convolution

In traditional speech, text, or image data, features are extracted densely. However, for 3-dimensional scans, such dense representation is inefficient since most of the space is empty. Instead, we can save non-empty space as its coordinate and the associated feature. This representation is an N-dimensional extension of a sparse matrix. In particular, we follow the COO format [32] as it is efficient for neighborhood queries (Sec. 3.1). The last axis is reserved for the batch indices to dissociate points at the same location in different batch [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 the 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 is the batch indices of i the coordinate and $\mathbf{f}_i$ is a vector. In Sec. 6, we augment the 4D space with the chromatic space and create a 7D sparse tensor for *trilateral* filtering.

3.1. Generalized Sparse Convolution

In this section, we generalize sparse convolutions proposed in [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 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 equivalent. Second, we define the convolution kernel shape arbitrarily with $\mathcal{N}^D$. This generalization encompasses many special cases such as dilated convolution kernels and hypercubic kernels. Another interesting special case is when $\mathcal{C}^{\text{out}} = \mathcal{C}^{\text{in}}$ and $\mathcal{N}^D = \mathcal{V}^D(K)$, we have the "sparse submanifold convolution" [9]. If we have $\mathcal{C}^{\text{in}} = \mathcal{C}^{\text{out}} = \mathbb{Z}^D$ and $\mathcal{N}^D = \mathcal{V}^D(K)$, the generalized sparse convolution is equivalent to the dense convolution (Eq. 2). For strided convolutions, $\mathcal{C}^{\text{out}} \neq \mathcal{C}^{\text{in}}$.

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 some functions that are essential. In particular, forward GPU functions that require non-trivial engineering.

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 and associated features. In Alg. 1, we list the GPU function for this process. Specifically, for semantic segmentation, we want to generate a label for each input coordinate-feature pair. If there are more than one different semantic labels within a voxel, we ignore this voxel during training by marking it with the `IGNORE_LABEL`. First, we convert all coordinates into hash keys and find the unique hashkey-label pairs to remove collision. Note that `SortByKey`, `UniqueByKey`, and `ReduceByKey` are all standard `Thrust` library functions [20]. The reduction function $f((l_x, i_x), (l_y, i_y)) =>$ (`IGNORE_LABEL`, i_x) takes 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.

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
$C'_p \leftarrow$ floor(C_p / v_l)
$\mathbf{k} \leftarrow$ hash(C'_p), $\mathbf{i} \leftarrow$ sequence(N)
$((\mathbf{i}', \mathbf{l}'), k') \leftarrow$ SortByKey((i, l), key=k)
$(\mathbf{i}'', (\mathbf{k}'', \mathbf{l}'')) \leftarrow$ UniqueByKey($\mathbf{i}'$, key=($\mathbf{k}', \mathbf{l}'$))
$(\mathbf{l}''', \mathbf{i}''') \leftarrow$ ReduceByKey(($\mathbf{l}'', \mathbf{i}''$), key=$\mathbf{k}''$, fn=$f$)
return $C'_p[\mathbf{i}''', :], F_p[\mathbf{i}''', :], \mathbf{l}'''$

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 the convolution (or pooling) layer stride size, the input coordinates, and the input sparse tensor stride size (the minimum distance between coordinates). The algorithm is presented in the supplementary material. In addition, we also support dynamically setting an arbitrary output coordinates $\mathcal{C}^{\text{out}}$ for the generalized sparse convolution.

Next, to convolve inputs with a kernel, we need a mapping to identify which inputs affect which outputs. 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

Algorithm 2 Generalized Sparse Convolution

Require: Kernel weights $\mathbf{W}$, input features F^i, output feature placeholder F^o, convolution mapping $\mathbf{M}$,

$F^o \leftarrow \mathbf{0}$ // set to 0
for all $W_{\mathbf{i}}, (I_{\mathbf{i}}, O_{\mathbf{i}}) \in (\mathbf{W}, \mathbf{M})$ **do**
  $F_{\text{tmp}} \leftarrow W_{\mathbf{i}}[F^i_{I_{\mathbf{i}}[1]}, F^i_{I_{\mathbf{i}}[2]}, ..., F^i_{I_{\mathbf{i}}[n]}]$ // (cu)BLAS
  $F_{\text{tmp}} \leftarrow F_{\text{tmp}} + [F^o_{O_{\mathbf{i}}[1]}, F^o_{O_{\mathbf{i}}[2]}, ..., F^o_{O_{\mathbf{i}}[n]}]$
  $[F^o_{O_{\mathbf{i}}[1]}, F^o_{O_{\mathbf{i}}[2]}, ..., F^o_{O_{\mathbf{i}}[n]}] \leftarrow F_{\text{tmp}}$
end for

the n-th element of the list of 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 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 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 features using $\mathbf{S}'$, which contains the beginning index of $\mathbf{I}$, and the corresponding output indices $\mathbf{O}$".

4.4. Global / Average Pooling, Sum Pooling

Average pooling and global pooling computes average of the input features for each output coordinate. This can be

Algorithm 3 GPU Sparse Tensor MaxPooling

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

implemented in multiple ways. One way is to create a sparse tensor that defines the kernel map for sparse matrix multiplication. If we do not divide the number of inputs for each output coordinate, this information can encode the density of the region, so we propose a variation that do not divide the number of inputs as the sum pooling. We use the `cuSparse` library for sparse matrix-matrix (`cusparse_csrmm`) and matrix-vector multiplication (`cusparse_csrmv`) to implement these layers. Same as the max pooling, $\mathbf{M}$ is the $(\mathbf{I}, \mathbf{O})$ input-to-output kernel map. For global pooling, we create the kernel map as all input coordinates to the origin for each batch and use the same Alg. 4. The transposed pooling (unpooling) works similarly. For sum pooling, we

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

do not compute N and do not divide the final features by N.

4.5. Non-spatial Functions

For functions that do not require spatial information (coordinates), we can apply the functions directly to the features F. For example, non-linearities do not require spatial information such as ReLU. 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 the 4-dimensional spatio-temporal convolutional neural network. We treat the time dimension as an extra spatial dimension and create a neural network with 4-dimensional convolutions. However, there are unique problems arising from such high-dimensional convolutions. First, the computational cost and the number of parameters in a network increases exponentially as we increase the dimension. However, we experimentally show that these increases do not necessarily lead to better performance. Second, the network does 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 save memory and computation with better generalization. Second, for spatio-temporal consistency, we propose a high-dimensional conditional random field (in 7D space-time-color space) that can enforce consistency and 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, if we use a 4D hypercube, or a tesseract (Fig. 2), for convolution kernels, the exponential increase in number of parameters most likely leads to over-parametrization, overfitting, as well as high computational-cost and memory consumption. Instead, we propose a hybrid kernel (non-hypercubic, non-permutohedral) that makes use of the arbitrary kernel shape of the generalized sparse convolution, $\mathcal{N}^D$.

Specifically, we define cross-shaped kernels and cubic kernels (Fig. 3) as well as hybrid kernels. For spatial dimensions, we use a cubic kernel to capture the spatial geometry accurately. And for the temporal dimension, we use the cross-shaped kernel to connect the same point in space across time. We call this kernel the hybrid kernel and is visualized in Fig. 3. We experimentally show that the hybrid kernel outperforms the tesseract kernel while being much faster.

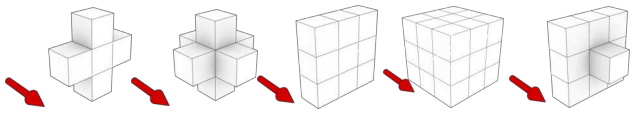

Figure 3: Various kernels in space-time. The red arrow indicates the temporal dimension and the other two axes are the spatial dimensions. The third spatial axis is hidden for visualization.

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 using the same generalized sparse convolutions homogeneously throughout the network, making the implementation easier and generic. In addition, as the building block of the network is convolutions, it allows us to mimic recent architectural innovations in 2D directly to high-dimensional networks. Thus, we adopt one of the most successful network architectures for our problem directly and create multiple instances of high-dimensional networks that closely resembles the original residual networks [11].

For the first layer, instead of a 7×7 2D convolution, we use $5 \times 5 \times 5 \times 1$ generalized sparse convolution. However, for the rest of the networks, we follow the original design and visualize the final 4D variant of ResNet18 on Fig. 4.

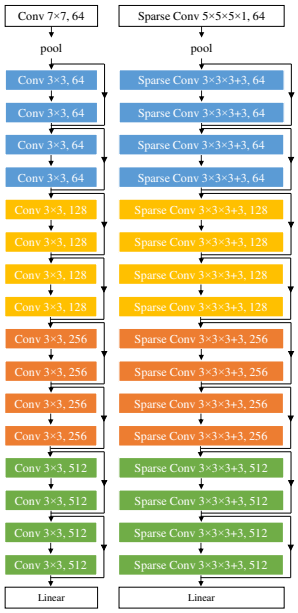

Figure 4: Architecture of ResNet18 (left) and MinkowskiNet18 (right). Note the structural similarity. $\times$ indicates a hypercubic kernel, $+$ indicates a hypercross kernel. (best viewed on display)

For the u-shaped variants, we create multiple strided sparse convolutions and strided sparse transpose convolutions with skip connections connecting the layers with the same stride size (Fig. 5). We use the variants of this architecture for semantic segmentation experiments.

6. Trilateral Stationary-CRF

The predictions from the MinkowskiNet for different time steps are not necessarily consistent throughout the temporal axis. To make such consistency more explicit and to improve predictions, we propose a conditional random field with a stationary kernel defined in a trilateral space. The trilateral space consists of 3D space, 1D time, and 3D chromatic space; it is an extension of a bilaterl space in image processing. The color space allows points with different colors that are spatially adjacent (e.g. on a boundary) to be far apart in the color space. Unlike conventional CRFs with Gaussian edge potentials and dense connections [14, 36], we do not enforce the function family of compatibility function except for the stationarity constraint.

We use the variational inference and approximate the distribtuion with the meanfield approximation [13] and convert the fixed point update similar to [36]. We make use of the arbitrary kernel shape of the generalized sparse convolution and convert the fixed point update into the generalized sparse convolution in the 7D space. During the training, we jointly optimize both a base network that generates unary potentials

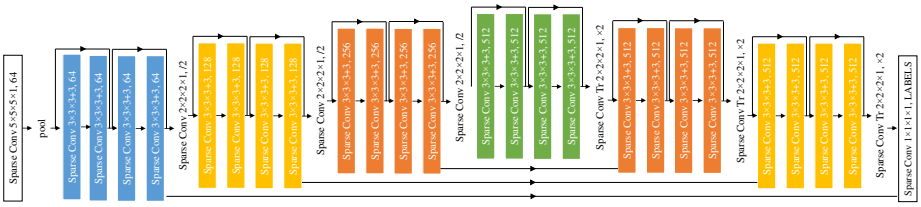

Figure 5: Architecture of MinkowskiUNet32. $\times$ indicates a hypercubic kernel, $+$ indicates a hypercross kernel. (best viewed on display)

and the compatibility function in the CRF end-to-end.

6.1. Definition

Let a CRF node in the 7D (space-time-chroma) space be x_i. We use the camera extrinsics to convert the spatial coordinates of a node x_i to be defined in the world coordinate system so that static points stay at the same coordinate even when the observer moves.

For each node x_i, we define the unary potential as $\phi_u(x_i)$ 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 final 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 $\tau_\mathbf{u}, \tau_\mathbf{v} \in \mathbb{R}^D$.

6.2. Variational Inference

The optimization problem $\arg\max_\mathbf{X} P(X)$ is intractable. So, we use the variational inference to minimize a divergence between the optimal $P(\mathbf{X})$ and an approximated distribution $Q(\mathbf{X})$. Specifically, we use the meanfield approximation, $Q = \prod_i Q_i(x_i)$ as the closed form solution exists. From the Theorem 11.9 in [13], 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 the edges can be defined using $\mathcal{N}^7$. The final algorithm is on 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 Minkowski networks, we first use multiple standard 3D benchmarks for 3D semantic segmentation. Next, we create multiple 4D datasets from 3D datasets with temporal information and perform ablation study.

7.1. Implementation

We implemented the Minkowski Engine using C++/CUDA and wrap it with PyTorch [22]. 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 (Sec. 4).

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 previous works. To convert voxel-level predictions

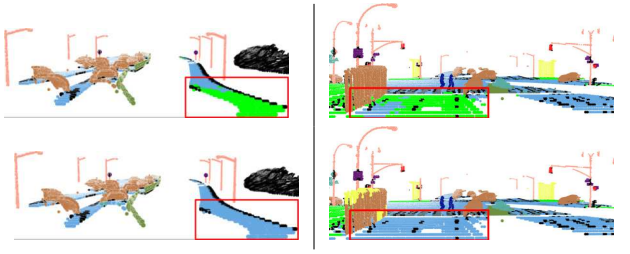

Figure 6: Visualizations of 3D (top), and 4D networks (bottom) on Synthia. A road (blue) far away from the car is often confused as sidewalks (green) with a 3D network, which persists after temporal averaging. However, 4D networks accurately captured it.

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 with 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 also use 5cm voxel and do not use rotation averaging.

RueMonge 2014 (Varcity). The RueMonge 2014 dataset [26] 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 [28] 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 moving 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 test. 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 pointclouds 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

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 to 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

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

Method	mIOU
ScanNet [5]	30.6
SSC-UNet [10]	30.8
PointNet++ [24]	33.9
ScanNet-FTSDF	38.3
SPLATNet [29]	39.3
TangetConv [30]	43.8
SurfaceConv [21]	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.

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.

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

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.

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

Method	mIOU	mAcc
PointNet [23]	41.09	48.98
SparseUNet [9]	41.72	64.62
SegCloud [31]	48.92	57.35
TangentConv [30]	52.8	60.7
3D RNN [33]	53.4	71.3
PointCNN [16]	57.26	63.86
SuperpointGraph [15]	58.04	66.5
MinkowskiNet20	62.60	69.62
MinkowskiNet32	**65.35**	**71.71**

Per class IoU in the supplementary material.

Table 5: RueMonge 2014 dataset (Varcity) TASK3 [26]

Method	mIOU
MV-CRF [27]	42.3
Gradde et al. [7]	54.4
RF+3D CRF [18]	56.4
OctNet (256^3) [25]	59.2
SPLATNet (3D) [29]	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.

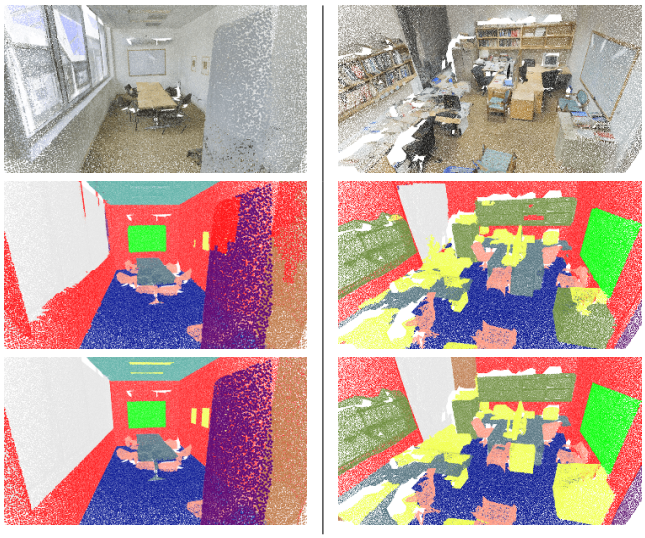

Figure 8: Visualization of Stanford dataset Area 5 test results. From the top, RGB input, prediction, ground truth.

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.

In Tab. 2 and Tab. 3, we can see the effectiveness of 4D networks and the 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, we small increase in computation, we could achive more robust algorithm with higher accuracy. In addition, when we process temporal sequence using the 4D networks, we could even get 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.

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. Using these, we create a 4D convolutional neural network for spatio-temporal perception. Experimentally, we show that 3D convolutional neural networks can outperform 2D networks and 4D perception can be more robust to noise.

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. 8

[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] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 5

[12] 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

[13] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. 5, 6

[14] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in Neural Information Processing Systems 24*, 2011. 5

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

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

[17] 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. 2

[18] 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. 8

[19] 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

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

[21] 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

[22] 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. 6

[23] 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

[24] 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

[25] 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, 8

[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*. Springer, 2014. 2, 7, 8

[27] 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. 8

[28] 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

[29] 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, 8

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

[31] 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

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

[33] 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

[34] 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

[35] 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. 2

[36] 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. 5