OctreeNet: A Novel Sparse 3-D Convolutional Neural Network for Real-Time 3-D Outdoor Scene Analysis

Fei Wang, Yan Zhuang, *Member, IEEE*, Hong Gu, and Huosheng Hu, *Senior Member, IEEE*

***Abstract*— Convolutional neural networks (CNNs) for 3-D data analyses require a large size of memory and fast computation power, making real-time applications difficult. This article proposes a novel OctreeNet (a sparse 3-D CNN) to analyze the sparse 3-D laser scanning data gathered from outdoor environments. It uses a collection of shallow octrees for 3-D scene representation to reduce the memory footprint of 3-D-CNNs and performs point cloud classification on every single octree. Furthermore, the smallest non-trivial and non-overlapped kernel (SNNK) implements convolution directly on the octree structure to reduce dense 3-D convolutions to matrix operations at sparse locations. The proposed neural network implements a depth-first search algorithm for real-time predictions. A conditional random field model is utilized for learning global semantic relationships and refining point cloud classification results. Two public data sets (Semantic3D.net and Oakland) are selected to test the classification performance in outdoor scenes with different spatial sparsity. The experiments and benchmark test results show that the proposed approach can be effectively used in real-time 3-D laser data analyses.**

***Note to Practitioners*—This article was motivated by the limitations of existing deep learning technologies for analyzing 3-D laser scanning data. This technology enables robots to infer what the surroundings are, which is closely linked to semantic mapping and navigation tasks. Previous deep neural networks have seldom been used in robotic systems since they require a large amount of memory and fast computation power to apply dense 3-D operations. This article presents a sparse 3-D-Convolutional neural network (CNN) for real-time point cloud classification by exploiting the sparsity of 3-D data. This framework requires no GPUs. The practicality of the proposed method is verified on data sets gathered from different platforms and sensors. The proposed network can be adopted for other classification tasks with laser sensors.**

Manuscript received December 24, 2017; revised June 24, 2018 and December 14, 2018; accepted September 9, 2019. Date of publication October 11, 2019; date of current version April 7, 2020. This article was recommended for publication by Associate Editor Z. Yin and Editor K. Saitou upon evaluation of the reviewers' comments. This work was supported by the National Natural Science Foundation of China under Grant U1608253 and Grant 61375088. *(Corresponding author: Yan Zhuang.)*

F. Wang and Y. Zhuang are with the School of Control Science and Engineering, Dalian University of Technology, Dalian 116024, China (e-mail: feiwang@mail.dlut.edu.cn; zhuang@dlut.edu.cn).

H. Gu is with the Faculty of Electronic Information and Electrical Engineering, Dalian University of Technology, Dalian 116024, China (e-mail: guhong@dlut.edu.cn).

H. Hu is with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, U.K. (e-mail: hhu@essex.ac.uk).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2019.2942068

***Index Terms*— Deep neural network, octree forest, point clouds, real-time classification, sparse convolution.**

I. Introduction

POINT cloud classification is a challenging task that has attracted great attention in the areas of computer vision and robotics. Given a point cloud captured by a laser scanner, the goal of classification is to assign one of the pre-defined class labels to each point in the cloud. This classification ability enables robots to build a high-level model of the environment for their localization and navigation tasks [1]. With the rapid development of laser sensors, millions of points can be acquired within a second. This has led to increasing demand for data analysis algorithms that can be implemented in real-time.

Convolutional neural networks (CNNs) have been deployed for analyzing 3-D data in various tasks, such as classification [2], object detection, [3], [4] and semantic segmentation [5], [6]. When applying CNNs in 3-D, we have to deal with two main difficulties. The first one is data representation of an unorganized 3-D point cloud that is a set of scattered points in 3-D space, without a regular structure. The second difficulty comes from the computational complexity that grows cubically with the size of 3-D data. Consequently, existing 3-D networks do not comply with the speed requirement in real-time robotic applications, even with GPU acceleration.

One important property of 3-D point clouds is spatial sparsity, which could be used to reduce the computational and memory requirements. Graham used a sparse representation and corresponding algorithms to dramatically decrease memory usage and increase the speed of 3-D operations if the density of 3-D data is below 10% [7]. Following this idea, the octree, a popular sparse representation of 3-D data, is introduced to reduce the memory footprint of 3-D-CNNs and an efficient data accessing algorithm is presented to speed up 3-D operations [5]. However, to our knowledge, limited research has been conducted in order to leverage the sparsity of 3-D data to improve the efficiency of 3-D-CNNs.

To fully exploit the sparsity of 3-D data, we propose a novel OctreeNet for real-time point cloud classification, which is a sparse CNN based on the octree. The octree forest is used to represent 3-D point clouds and OctreeNet is adopted to perform classification on every single octree. This leads to an efficient parallel implementation framework of inference. A conditional random field (CRF) model is used to impose spatial consistency on the coarse classification results of our OctreeNet.

The key component of our OctreeNet is the smallest nontrivial and non-overlapped kernel (SNNK), which enables sparse convolutions to be performed directly on the octree structure. Moreover, an inference algorithm is presented to reduce the dense 3-D operations into small matrix operations through a depth-first search. The efficiency of our OctreeNet is demonstrated on two public outdoor data sets with different point density. Qualitative and quantitative evaluations against state-of-the-art methods are conducted in terms of both accuracy and efficiency.

The rest of this article is organized as follows. Section II presents related studies on point cloud classification. Section III describes our OctreeNet and corresponding algorithms that could speed up the computations of CNNs for 3-D data analyses. Experimental results are given in Section IV to verify the performance of the proposed method in analyzing the sparse 3-D laser scanning data. Finally, a brief conclusion and future study are given in Section V.

II. Related Work

A common paradigm in most previous studies for point-wise labeling problem includes: 1) handcrafted feature extraction; 2) a discriminative classifier; and 3) a graphical model for label refinement [9], [10], [11]. Under this paradigm, it is critical to design appropriate feature representations to capture the geometric properties at each point of the cloud.

Several features of outdoor scenes have been presented for different purposes. In [9], the spectral features were computed to capture the scatter-ness, linear-ness, and surface-ness of the local geometry. To identify crease edges and occlusion boundaries, Hackel *et al.* [8] augmented the spectral feature set with the first and second-order moments of the point neighborhood around the eigenvectors. Fast point feature histograms (FPFH) was used in [11] to describe the geometric relationships between a point and its neighbors in terms of distance and normal vector orientations. Himmelsbach *et al.* [12] combined point level features (similar to the spectral features) and object-level features (like object volume) to classify the segmented objects in the cloud. However, it is a non-trivial task to find the optimal feature combination for a specific problem.

For real-time 3-D data analyses, the 3-D nearest-neighbor searching operation is the main bottleneck of feature-based methods. A scrolling grid representation and the scanning algorithm were proposed to speed up the range search operation by reusing previously computed data [13]. Hu *et al.* [10] used a collection of pillars, blocks, and voxels as 3-D data representation to improve the speed of long-range queries. The point cloud was down-sampled to a multi-scale pyramid and an approximate neighborhood was constructed at different scales to speed up the feature computation [8].

Besides, the inference may be significantly slowed down when global contextual information is considered. Considering that the time cost of a graphical model is relative to the number of nodes in the graph, Munoz *et al.* [9] speeded up the inference by using segments as the nodes instead of the raw 3-D points. To avoid the use of expensive graphical models, Hu *et al.* [10] proposed an iterative inference procedure over a hierarchical segmentation of the point cloud.

Recently, deep learning technology has been applied for 3-D data analyses. A hierarchical feature learning framework was proposed in [15] for a point-wise labeling problem, which can directly deploy raw 3-D points to capture local structures. Tchapmi *et al.* [6] combined a fully CNN (FCN) with a CRF model via Trilinear Interpolation to output fine-grained point-level predictions. The SqueezeSegV2 model was proposed in [16] for road-object segmentation from 3-D LiDAR point clouds.

As 3-D-CNNs require fast computers and a large scale of memory, they have not widely used for robotic systems. To avoid exhaustive dense convolution in 3-D space, several researchers projected 3-D point clouds into 2-D images and then leveraging the highly engineered CNN structures to perform image classification [17], [18]. During the generation of 2-D images, only a few properties of 3-D points (such as depth and normal) were used and most of the 3-D geometric information was lost. Consequently, these approaches often required additional color information for classification.

Motivated by the idea of sparse matrices, Graham presented a sparse representation of 3-D data and corresponding algorithms to convert 3-D convolutions to matrix operations at sparse feature locations [7]. Unfortunately, the sparsity decreased after each convolutional layer and the number of required operations for deeper layers kept growing. To deal with this problem, an L1-regularization was used on the filter activations in [4] and the submanifold convolution was proposed in [19] restricting outputs only to the set of active inputs. Another way to speed up convolution on sparse data is presented in [20], where convolution is performed in the permutohedral lattice space. The kernels were viewed as a high dimensional linear filter and only convolved at sparsely populated lattice points. Ren *et al.* [21] proposed a tiling-based sparse convolution algorithm for fast inference. They reduced computation in the high-resolution main network by using masks from *a priori* problem knowledge or a low-cost network.

To improve the computational efficiency, Wang *et al.* [22] designed an octree data structure to store the octant information and CNN features into the graphics memory and executed an entire network on the GPU. In [23], the voxel block octree was proposed as the data structure for surface prediction to facilitate high-resolution outputs. Similarly, Tatarchenko *et al.* [24] proposed a convolutional decoder based on octrees to represent high-resolution outputs with a limited memory budget. Riegler *et al.* [5] proposed a hybrid grid-octree data structure to reduce the memory requirement of 3-D-CNNs and presented efficient data accessing algorithms to improve the speed of dense 3-D operations. These studies share the idea of using the octree data structure to save memories. In contrast, our method is focused on exploiting the sparsity of 3-D data to speed up the 3-D network operations.

III. OctreeNet

Unlike 2-D images, a 3-D point cloud is a set of scattered points in space, which is not a suitable input to CNNs. Simple voxelization strategy may provide regular 3-D grids, but the memory usage would increase significantly. Thus, we restrict our attention to a popular space partitioning structure, octree, for 3-D data representation to reduce the memory footprint of 3-D-CNNs.

As we know, performing CNN operations directly on an octree structure is not efficient. Convolutions often require frequent access to the neighboring elements. Accessing an arbitrary element in the standard implementation of the octree requires a traversal from the root to the desired node, which imposes an increased cost for deep octrees. Besides, 3-D-CNNs are time-consuming in nature. Applying a single convolutional layer with stride s on an $N \times N \times N$ grid, we would need to apply it (N/s) more times than in the 2-D case. To extend a whole model in 3-D would be trillions of operations, which is unacceptable for robotic systems with limited computing capacity and memory.

To tackle these challenges, we introduce the octree forest as 3-D data representation to form a novel sparse CNN, namely OctreeNet. Our OctreeNet converts dense convolutions to matrix operations at sparse locations and the classification results can be carried out through a depth-first search algorithm.

A. Octree Forest

As mentioned above, the data-accessing problem in the octree data structure is worsening as the tree depth increases. Therefore, instead of representing the entire 3-D point cloud with a single deep octree, we adopt the octree forest, a collection of shallow octrees, for data representation as shown in Fig. 1(c). More specifically, we coarsely partition the entire 3-D space into cube-shaped local regions. Only the occupied local regions are stored. Each of them is represented by a shallow octree with a small fixed depth (we use depth = 5 in our experiments). These shallow octrees are indexed based on their global locations and stored in a hash table. We can loop over these octrees and retrieve the voxels within them.

With the octree forest structure in Fig. 1, we could use 3-D-CNNs to learn global features for classification if we feed an entire scene into the network. This makes the computational complexity unacceptable as we have to perform all operations on a large-scale 3-D space. To reduce time costs, we first utilize 3-D-CNNs to extract local semantic information within a cube-shaped region (represented by a shallow octree in the forest) of the scene and classify on each region independently. This provides a coarse prediction since global relationships among different regions are ignored. Then, we introduce a CRF model to learn global semantic information and refine the output. This strategy has the following benefits.

1) Applying CNN operations on a local region is much faster as the input size becomes smaller and all required data are restricted within a single shallow octree.
2) Treating regions separately for a parallel implementation of inference.
3) Sufficient samples are obtained to train a deep network.

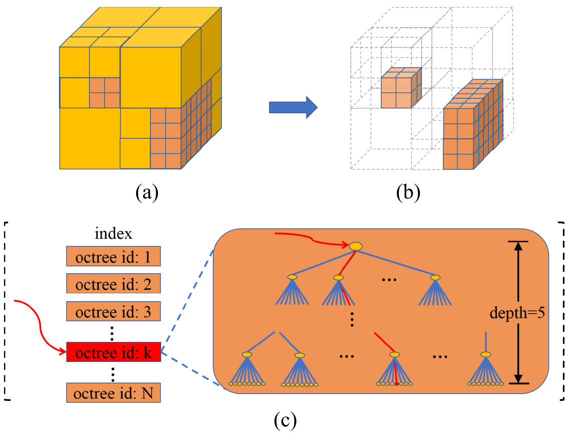


Fig. 1. Illustration of the octree forest representation of a point cloud. The 3-D space is first partitioned into cube-shaped regions and only the occupied regions are stored. An occupied region is represented by a shallow octree and indexed by its global location. To retrieve a voxel in the octree forest, we need to search within the octree where the desired voxel lies. The red lines show a traversal path to the desired voxel. (a) Space partitioning. (b) Occupied regions. (c) Octree forest structure.

B. Smallest Non-Trivial and Non-Overlapped Kernel

Convolution is the most important operation in modern networks but also the most expensive. Simply extending the widely used convolution with a 3×3 kernel to 3-D would significantly increase the computational cost. Considering that 3-D data is sparse in nature, Graham [7] proposed to use the smallest non-trivial kernel (kernel size = 2) to perform sparse 3-D convolutions for speedup. Motivated by this article, we present the SNNK (stride = kernel size) to enable convolutions to be computed directly on the octree structure.

Data Accessing: Let $T_{i,j,k}$ denote the value of a 3-D tensor T at the location (i, j, k). Formally, convolving a 3-D tensor T with a 3-D kernel $W \in \mathbb{R}^{L\times M\times N}$ can be written as follows:

$$H_{i,j,k} = (T * W)_{i,j,k} = \sum_{l=0}^{L-1}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} T_{\hat{i},\hat{j},\hat{k}} \cdot W_{l,m,n} \tag{1}$$

where $\hat{i} = i + l - L/2$, $\hat{j} = j + m - M/2$, $\hat{k} = k + n - N/2$.

To compute a single element in H, we need to retrieve $L \times M \times N$ elements in T. For the typical convolution with kernel size = 3 and stride = 1, the number of required elements is $(3)^3$. Unfortunately, those elements are located on different branches of an octree and we have to traverse the octree four times to access them, as shown in Fig. 2(a). This top-down traversal is a time-consuming operation for the octree structure. Since convolution requires frequent access to the neighboring elements, convolving with the typical kernel on the octree structure is less efficient.

To access data quickly, we let kernel size = stride = 2 in SNNK. Hence, convolution with SNNK becomes

$$H_{i,j,k} = \sum_{l=0}^{1}\sum_{m=0}^{1}\sum_{n=0}^{1} T_{2i+l,2j+m,2k+n} \cdot W_{l,m,n} \tag{2}$$

where $H \in \mathbb{R}^{P\times Q\times S}$ and $T \in \mathbb{R}^{2P\times 2Q\times 2S}$. The $(2)^3$ elements required in (2) come from the same parent in an octree.

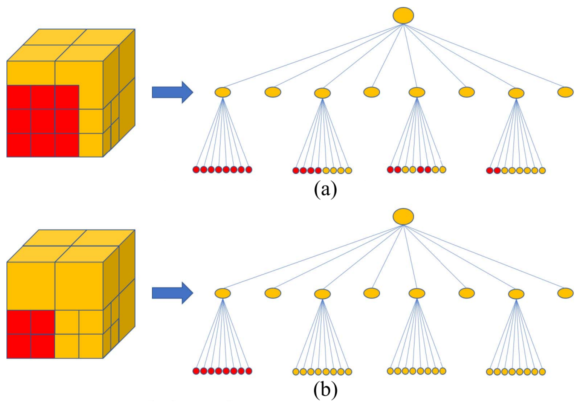


Fig. 2. Comparison of the required data (red color) in the calculation of (a) convolution with a typical kernel (kernel size = 3, stride = 1) and (b) convolution with SNNK. In (a) we have to traverse the octree four times to get the required data. In contrast, all required data lie in the same branch of an octree in (b). To access them, only a single traversal of the octree is needed. This can reduce the computational cost since the convolution operation requires frequent data access.

To access them, only one traversal is needed [see Fig. 2(b)]. This could save us 3/4 of time for data accessing.

Naïve Implementation: The naive implementation of convolution with SNNK is as follows. Let $t_{(i,j,k)}$ denote a vector that contains the eight required elements in (2) and reshape W to a 1-D vector w. The vectorization of (2) is

$$H_{i,j,k} = t_{(i,j,k)}^{T} \cdot w. \tag{3}$$

As kernel size = stride = 2, each $t_{(i,j,k)}$ does not overlap and all $t_{(i,j,k)}$ cover every single element in T. So, we define a function *ten2vec* to map from a 3-D tensor $T \in \mathbb{R}^{N\times N\times N}$ to a matrix $\tilde{T}$

$$ten2vec : \tilde{T}_p = t_{(i,j,k)}^{T} = \underset{(\bar{i},\bar{j},\bar{k})\in\Omega[i\cdot s,j\cdot s,k\cdot s]}{sequence} (T_{\bar{i},\bar{j},\bar{k}}) \tag{4}$$

where $p = i \cdot (N/s)^2 + j \cdot (N/s) + k$, (s^3) is the size of each small input block. *sequence*is a function that converts its input into a vector and $\Omega[i, j, k]$ denotes a set of locations of elements in an input block. Similarly, the reverse mapping is given by

$$vec2ten : \; T_{\bar{i},\bar{j},\bar{k}} = desequence(\tilde{T}_p) \tag{5}$$

where *desequence* is the reverse function of the *sequence*.

With the two functions defined above, convolution with SNNK can be written as follows:

$$H = vec2ten(ten2vec(T) \cdot w) \tag{6}$$

where $s = 2$ in *ten2vec* and $s = 1$ in *vec2ten.*

Sparse implementation: The above implementation is time-consuming since $\tilde{T}$ is very large. Following the definition of an active vector in [7], we define a vector to be active: 1) if it is a non-zero vector for the input layer or 2) if any vector in the layer below from which it receives input is active. As $\tilde{T}$ has a few active vectors, we divide $\tilde{T}$ into active and non-active parts

$$\tilde{T} = \{\tilde{T}_{\text{act}}, \tilde{T}_{\text{non}}\}. \tag{7}$$

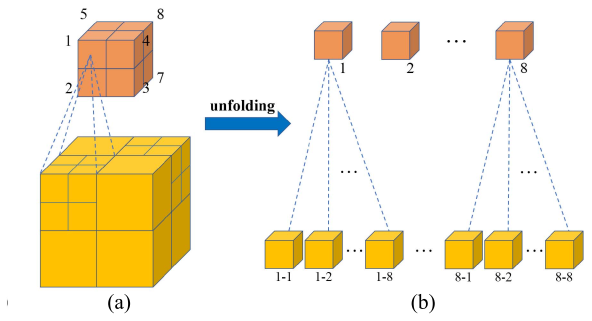


Fig. 3. Illustration of unfolding convolution with SNNK. If the eight elements in a grid are labeled as (a), then convolution with SNNK can be unfolded as (b). In the structure of the unfolded convolution, one node is linked to eight child nodes, which is consistent with the structure of an octree as shown in Fig. 1. There exists a one-to-one correspondence between each node in (b) and each node in an octree, indicating that the convolution operation with SNNK could be performed directly on the octree structure.

All non-active vector in $\tilde{T}$ are the same and the shared output can be pre-computed. As outputs corresponding to active vectors need to be computed at runtime, operations in (6) are only applied on the active part of $\tilde{T}$. This can reduce the computational cost of 3-D convolution since the size of $\tilde{T}_{\text{act}}$ is relatively small.

Furthermore, it is straightforward to find active vectors in the octree structure. Fig. 3 shows the unfolded version of convolution with SNNK. The structure of a vector t with eight elements is consistent with the structure of a node comprising eight children in an octree. Under our definitions, an active vector indicates that the corresponding space is occupied while a non-active vector stands for the unoccupied area. Hence, each active vector can be interpreted as a non-leaf node in the octree structure and the eight elements in the vector correspond to the eight children of the node. The vector is active if and only if the corresponding non-leaf node in the octree does exist. Suppose that the depth of an octree is D_{tree}. To compute the output of the ith convolutional layer, we only have to accomplish.

1) To initialize H with the pre-computed value.
2) To search for all non-leaf nodes with depth $(D_{\text{tree}} - i)$ in the octree and construct $\tilde{T}_{\text{act}}$.
3) To update the active elements in H using $\tilde{T}_{\text{act}} \cdot w$.

C. Overall Architecture

Our OctreeNet structure is summarized in Fig. 4, which has two parts: convolution network and deconvolution network. More specifically, a convolution network is a feature extractor that transforms the input grid into different-level feature representations. High-level features capture global shape information in a point cloud, whereas low-level features correspond to details, such as edges and corners. Our deconvolution network is a hierarchical classifier that handles classification results according to different levels of features learned by the convolution network. Its final output is a dense score map with the same size as the input grid. The score map represents the probability of each voxel that assigned to one of the predefined classes.

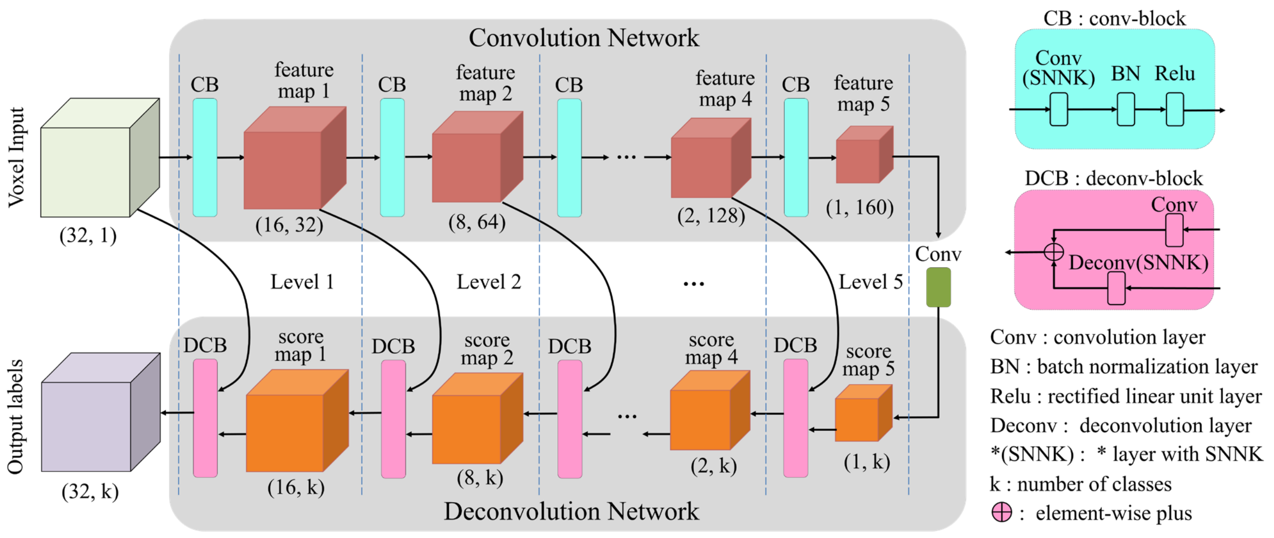


Fig. 4. Overall architecture of OctreeNet. Our network is composed of two parts: a convolution network for different-level feature extraction and a deconvolution network as a hierarchical inference procedure that fuses score maps from high level and detailed feature maps from low level. (*size*, *chn*) denotes the spatial size and the number of channels of each feature map and score map. The downsampling/upsampling is achieved by convolving/deconvolving with SNNK (kernel size = 2, stride = 2).

Our convolution network involves a series of conv-blocks to learn features from different levels. The ith conv-block computes feature maps f_i from the input data through a convolution layer, a batch normalization layer and a rectified linear unit layer. The convolution layer is applied with SNNK. Since the spatial size has been reduced after convolving with SNNK, no pooling layer is used in our OctreeNet. The number of channels starts with 32 in the conv-block furthest from the output and increases by 32 at each subsequent block.

Motivated by [25], we introduce the deconvolution network as a hierarchical classifier for label prediction. The output of the top convolution layer is a score map that indicates which object is contained in the grid. Our deconvolution network leverages this high-level score map and features from lower levels to form a hierarchical inference procedure. Assuming that f_i, s_i are feature maps and score maps at level i. The mathematic operation of a deconv-block can be written as follows:

$$s_i = \mathrm{DC}(s_{i+1}) + C(f_i) \tag{8}$$

where dc denotes a deconvolution operation that enlarges a previous score map to a finer one, and C is a convolution operation that transforms a feature map into a class-specific score map. Deconvolution operation is also applied with SNNK while the convolution operation is performed with a $1 \times 1 \times 1$ kernel.

D. Inference Algorithm Based on Octree Searching

One key contribution of this article is to show that the inference of OctreeNet can be carried out through a depth-first search algorithm. Dense 3-D operations in a neural network are simplified to matrix operations and directly performed on the octree structure in a depth-first search of the octree, improving the computational efficiency when the input data is sparse.

The mathematic operation of a conv-block can be written by

$$f_i = relu(\mathrm{BN}(C(T))) \tag{9}$$

where C denotes the convolution operation with SNNK, BN is the batch normalization operation and *relu* is the rectified linear unit operation. First, convolutions with SNNK can be sparsely computed through an octree searching approach, as suggested in Section III-B. According to [26], BN(T) is computed by

$$\begin{aligned} Y &= \mathrm{BN}(T) = \gamma \cdot \hat{T} + \beta \\ \hat{T} &= (T - \mu)/\sqrt{\sigma^2 + \varepsilon} \end{aligned} \tag{10}$$

where μ and σ^2 denote the expectation and variance of T, γ, and β are learned parameters and ε is an error term.

Because all operations in (10) are element-wise, BN(T) can be applied for every element separately. Since *relu* is also an element-wise operation, the output of ith conv-block can be computed as follows.

1) Initialize f_i with the pre-computed values.
2) Search for all non-leaf nodes with depth $(D_{\text{tree}} - i)$ in the octree and construct $\tilde{T}_{\text{act}}$.
3) Compute T_{temp} by $relu(\gamma\,(\tilde{T}_{\text{act}} \cdot w - \mu)/(\sigma^2 + \varepsilon)^{1/2} + \beta)$.
4) Update the active parts of f_i with T_{temp}.

As our convolutional network is a stack of conv-blocks, the above approach can be computed several times for the final output. However, it involves several non-leaf nodes' searching which is time-consuming. The vector comprising its output $H_{i,j,k}$ in the next conv-block is an active vector if t is an active vector in the current conv-block. This suggests that the output of a conv-block can be used to initialize the input of the next conv-block.

As mentioned above, there is a one-to-one correspondence between the structure of convolution with SNNK and the

Algorithm 1 Octree Searching For CNN

```
Input: an octree node $N$ and its depth $d$
 1:  $N$.feature $\leftarrow$ getPrecomputedFeature($d$)
 2:  $w \leftarrow$ getConvolutionParameterOfCB($d$)
 3:  $\mu, \sigma, \gamma, \beta \leftarrow$ getBNParameterOfCB ($d$)
 4:  if $N$ is NULL then
 5:    return pre-computed feature map $N$.feature
 6:  end if
 7:  for $i = 0$ to 8 do
 8:    $t[i] \leftarrow$ OctreeSearchingForCNN($N$.children[$i$], $d$+1)
 9:  end for
10:  $C \leftarrow t^T \cdot w$
11:  $\hat{C} \leftarrow (C - \mu)/\sqrt{\sigma^2 + \varepsilon}$
12:  $BN \leftarrow \gamma \cdot \hat{C} + \beta$
13:  $N$.feature $\leftarrow relu(BN)$
14:  return current feature map $N$.feature
```

Algorithm 2 Octree Searching For DeCNN

```
Input: an octree node $N$ and its depth $d$
 1: $w_d \leftarrow$ getDeconvolutionParameterOfDCB ($d$)
 2: $w_d, b \leftarrow$ getConvolutionParameterOfDCB ($d$)
 3: $S_{\text{parent}} \leftarrow N$.score_map$\cdot w_d^T$
 4: for $i = 0$ to 8 do
 5:   if $N$.children[$i$] is not NULL then
 6:     $S_{\text{child}} \leftarrow (N$.children[$i$].feature$)^T \cdot w_c + b$
 7:     $N$.children[$i$]. score_map $\leftarrow S_{\text{parent}}[i] + S_{\text{child}}$
 8:     OctreeSearchingForDeCNN($N$.children[$i$], $d$+1)
 9:   end if
10: end for
```

structure of an octree. Then, in the octree structure, if a node is used for the calculation of the ith conv-block, its parent is just the non-leaf node we are searching for the next conv-block (see Fig. 5). Thus, we can get active vectors for all conv-blocks by searching the octree once. Finally, our convolutional network can be implemented using Algorithm 1.

Similarly, our deconvolution network can be computed through another depth-first search of an octree, that is

$$H = vec2ten(ten2vec(T) \cdot w^T) \tag{11}$$

with $s = 1$ and $s = 2$ for *ten2vec* and *vec2ten,* respectively.

This is a transposed version of convolution and can be computed in the same way. Besides, the $1 \times 1 \times 1$ convolution in our deconv-block is treated as element-wise multiplication operation. Therefore, our deconvolution network can be implemented by using Algorithm 2.

IV. Experiments

To demonstrate the effectiveness of our OctreeNet, we test it on two public data sets of outdoor scenes with different point densities. Our approach is evaluated against several existing state-of-the-art methods in terms of both accuracy and efficiency. We compare time costs and memory usages of our network implemented under GPU and CPU environments. A qualitative comparison among different methods is also presented in this section.

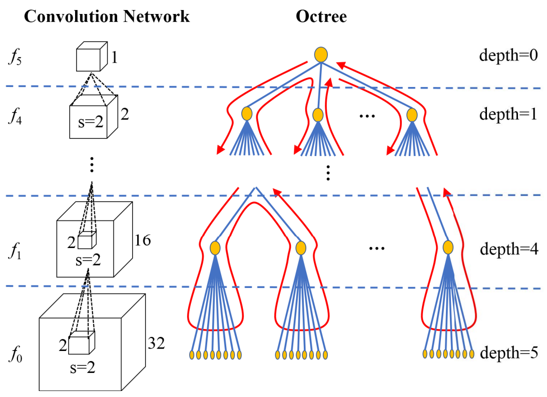


Fig. 5. Illustration of the similar hierarchical structures of convolution network and the octree. f_0 is the 3-D voxel input and s denotes the stride used in convolution. Red arrows show a deep-first search of the octree. There is a one-to-one correspondence between each element in f_i and each node with depth $(5-i)$ in the octree. The calculation of the convolution network is a bottom-up approach, so the output can be computed through a deep-first search of the octree. This can reduce both computational and memory costs according to the sparsity of each f_i.

A. Training

Our network is trained in two phases. First, we train the convolution network to learn robust feature representations of 3-D shapes. Especially, a sigmoid layer is added after the top convolution layer to predict which object is contained in the local region. This is a multi-label problem since different objects may occur in the same local region. The cross-entropy loss function is given by

$$L = -\frac{1}{N}\sum_{n=1}^{N}(y_n \log \hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)) \tag{12}$$

where y_n and $\hat{y}_n$ denote the ground-truth and predictions for the nth training sample.

Then, we fix parameters in the convolution network and train the deconvolution network in a similar way. Fig. 6 shows the loss curves of the above two training phases. With batch normalization in each conv-block, our OctreeNet can be efficiently optimized using stochastic gradient descent. We utilize the popular Adam algorithm with a learning rate of 1e−5 for optimization. The weights in each layer are initialized from a zero-mean Gaussian distribution with a standard deviation of 0.001. The batch size is 32. Here, we rotated the entire point cloud 360°/n intervals around the z-axis and translated half of the size of a local region along each dimension to augment the training data. This could avoid overfitting during the training of our network, which is implemented with Tensorflow on a single NVIDIA GTX 1080 to speed up the training process.

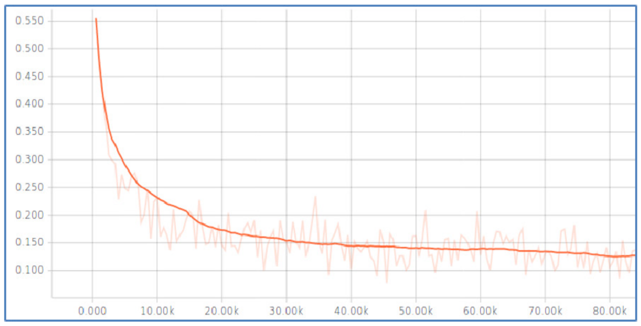
B. Classification Results

We evaluate the performance of OctreeNet on two public data sets, namely the Semantic3D.net data set and the Oakland data set. Classification results on these two data sets are given, respectively. Our approach is compared with state-of-the-art methods in terms of precision, recall, and F1 score.

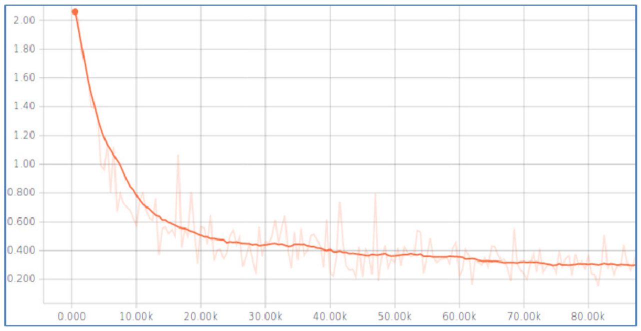
TABLE I

EVALUATIONS OF DIFFERENT METHODS ON THE SEMANTIC3D.NET DATA SET

Evaluation	**Method**	man-made terrain	natural terrain	high vegetation	low vegetation	buildings	hard scape	scanning artefacts	cars
Precision	TMLC-MSR[8]	**0.93**	0.93	0.57	0.41	**0.94**	**0.55**	**0.48**	0.55
	DeePr3SS[17]	0.88	0.92	0.78	0.62	**0.94**	0.46	0.35	0.80
	SnapNet[18]	0.82	0.92	0.81	**0.74**	**0.94**	0.47	0.42	**0.91**
	SEGCloud[6]	0.86	**0.94**	**0.88**	0.51	**0.94**	0.49	0.44	0.82
	OctreeNet(Ours)	0.86	0.89	0.81	0.56	0.93	0.27	0.38	0.63
	OctreeNet+CRF(Ours)	**0.93**	0.88	0.83	0.69	0.93	0.37	0.38	0.75
Recall	TMLC-MSR[8]	0.95	0.78	0.88	0.43	0.93	0.22	0.59	0.69
	DeePr3SS[17]	0.96	0.89	0.93	0.40	0.95	0.23	0.45	0.69
	SnapNet[18]	**0.99**	0.82	0.97	0.24	**0.96**	0.23	**0.74**	0.68
	SEGCloud[6]	0.96	0.68	0.97	**0.65**	0.95	**0.45**	0.41	**0.74**
	OctreeNet(Ours)	0.98	0.78	0.94	0.40	0.94	0.13	0.54	0.61
	OctreeNet+CRF(Ours)	0.96	**0.92**	**0.98**	0.47	0.95	0.13	0.62	0.54
F1-score	TMLC-MSR[8]	0.94	0.85	0.69	0.42	0.94	0.31	0.53	0.61
	DeePr3SS[17]	0.92	**0.90**	0.85	0.48	0.94	0.31	0.40	0.74
	SnapNet[18]	0.90	0.87	0.88	0.37	**0.95**	0.31	**0.54**	**0.78**
	SEGCloud[6]	0.91	0.79	**0.92**	**0.57**	**0.95**	**0.47**	0.43	**0.78**
	OctreeNet(Ours)	0.92	0.83	0.87	0.47	0.93	0.18	0.45	0.62
	OctreeNet+CRF(Ours)	**0.95**	**0.90**	0.90	0.56	0.94	0.19	0.47	0.63



(a)



(b)

Fig. 6. Visualization of losses during the two training phases of OctreeNet. (a) Losses during the training of our convolution network. (b) Losses during the training of our deconvolution network.

Pre-Processing and Post-Processing: In pre-processing, the raw 3-D points are first voxelized to voxels. Any voxel with a point inside is assigned a scalar value 1 (otherwise 0) as [3]. Then, we partition the space into cube-shaped local regions with a resolution of 32 × 32 × 32. Finally, shallow octrees are constructed from each local region as inputs to the OctreeNet. This pre-processing procedure is applied on the two data sets with different voxel size (8 cm for the Semantic3D.net data set and 16 cm for the Oakland data set), which is chosen according to the point density of the data set, as suggested in [10].

The CRF model is adopted as post-processing to learn the relationships between local regions and refine classification results. The model employs the energy function

$$E(x) = \sum_{i} \phi_u(x_i) + \sum_{i<j} \phi_p(x_i, x_j). \tag{13}$$

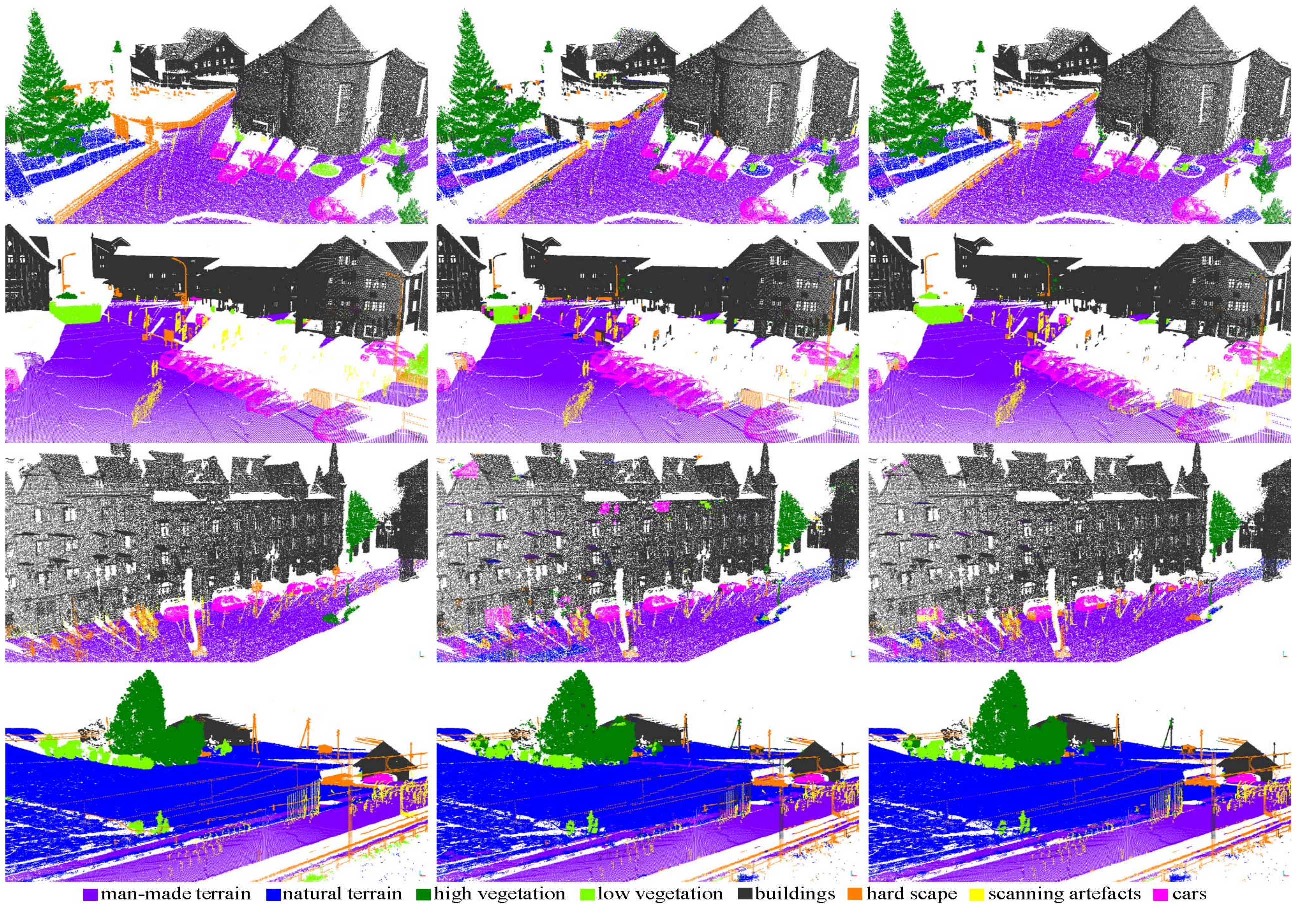
ϕ_u is the unary energy defined as $\phi_u(x_i) = -\log(p(x_i))$, where $p(x_i)$ is the label probability produced by the OctreeNet. ϕ_p denotes the pairwise energy that incorporates the neighbor information

$$\phi_p(x_i, x_j) = \mu(x_i, x_j) W_\theta(\|x_i - x_j\|) \tag{14}$$

where W_θ is the Gaussian function with a standard deviation θ. The label compatibility function $\mu(x_i, x_j)$ and hyperparameter θ are learned with the algorithm provided by [14].

Semantic3D.netData Set: This data set contains more than 3 billion points and covers a variety of urban scenes, such as streets, squares, and churches. Points are extremely dense and the resolution is smaller than 0.01 m. This increases the time costs for both training and testing. The points are labeled into eight categories, i.e., high/low vegetation, buildings, and cars.

Fig. 7 shows the classification results of four different scenes. In the third example, some parts of buildings are incorrectly labeled as cars or vegetation by our OctreeNet. There are also some misclassifications between hard scape and buildings in the first and last cases. This is because only the geometric information within a local region is used by our OctreeNet and global relationships among different regions are ignored. From a local view, some parts of buildings, cars, and fences may share a similar 3-D shape and difficult to



🟪 man-made terrain 🟦 natural terrain 🟩 high vegetation 🟩 low vegetation ⬛ buildings 🟧 hard scape 🟨 scanning artefacts 🟪 cars

Fig. 7. Classification results of four different scenes of the Semantic3D.net data set. From left to right—ground truth annotation, our OctreeNet and OctreeNet + CRF.

TABLE II
PROPERTIES OF DIFFERENT METHODS

Method	**Input Data**		**Feature**		**Context**	
	Point	Color	Manual	Learned	Local	global
TMLC-MSR[8]	✓		✓		✓	
DeePr3SS[17]	✓	✓		✓	✓	✓
SnapNet[18]	✓	✓		✓	✓	✓
SEGCloud[6]	✓	✓		✓	✓	✓
OctreeNet	✓			✓	✓	
OctreeNet+CRF	✓			✓	✓	✓

be distinguished completely. However, the global contextual information lost in OctreeNet can be learned by a CRF model. As post-processing, the CRF can remove some noises in the output and achieve about 3% improvement in average.

The performance of different methods on this benchmark is reported in Table I. To give a better comparison among different methods, we summarize the data requirement, feature extraction method, and contextual information usage of them in Table II. More benchmark test results can be found at http://www.semantic3d.net/view_results.php?chl = 2.

TMLC-MSR [8] is a handcrafted-feature-based method that introduces a multi-scale pyramid to the model local context at different scales. DeePr3SS [17], SnapNet [18], and SEGCloud [6] are deep-learning-based methods. The data was augmented by rotating $360°/n$ intervals around a fix vertical axis in [17]. In [6], randomly rotating 360° along the z-axis and scaling were used for data augmentation. In [18], 3-D points were projected to 2-D images first and data augmentation is done by randomly generating the camera positions and orientations.

As shown in Table II, existing deep learning methods require additional color information for classification. However, our approach does not require color information as it is not available with a standard laser scanner equipped on an outdoor robotic platform. Even without color information, our approach shows only 4% F1-score decrease in average when compared against the state-of-the-art methods on this benchmark.

Oakland Data Set: This data set was collected by an unmanned ground vehicle equipped with two SICK laser measurement sensor (LMS) laser scanners vertically facing sideways. The data covers typical urban scenes and is mainly labeled into seven outdoor objects, i.e., leaves, buildings, ground, pole, tree trunk, wire, and vehicle. Since pole and tree trunk share similar local geometric structure, we merge them into one category referred to as pole.

Fig. 8 shows some typical experimental results of our approach to this benchmark. There are many noises in the

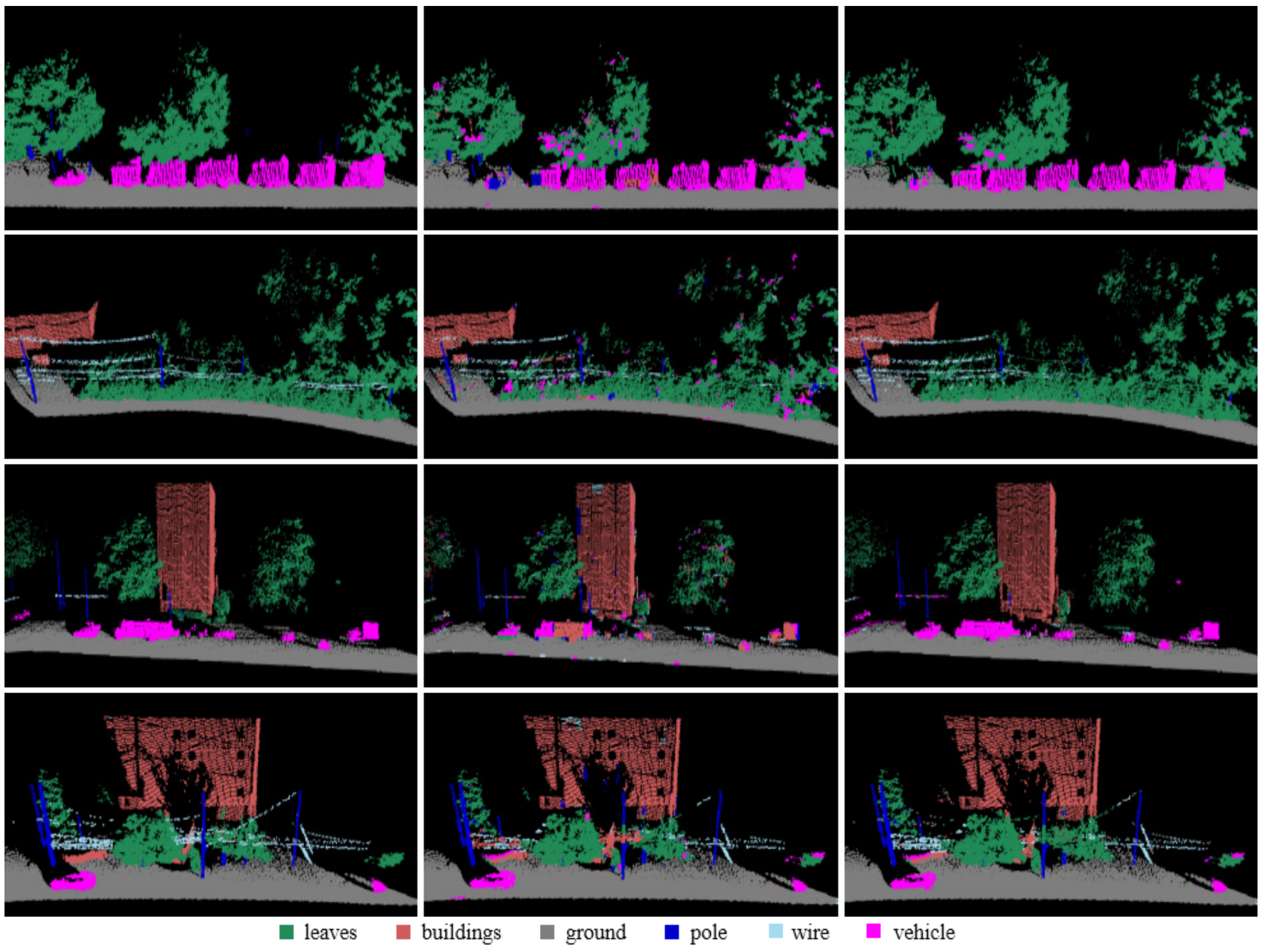


Fig. 8. Classification results of four different scenes in the Oakland data set. From left to right—ground truth annotation, our OctreeNet and OctreeNet + CRF.

classification results of our OctreeNet. This is mainly because the distribution of points in 3-D space is very imbalanced. The point density varies with both: 1) the distance between objects and the laser sensor and 2) the speed of the moving platform. For example, the measurement points of buildings far away from the sensor seem to be several horizontal lines rather than a plane. Hence, our OctreeNet classified them as wires incorrectly. Similarly, when the platform moves fast, the distance between the two adjacent scan lines increases. Consequently, some points on the surface of a building or a vehicle may look like a vertical line and are misclassified as poles. As most of these misclassifications can be corrected by the CRF model, this post-processing improves 9% F1-score in average.

Table III presents the evaluation results among our approach and some state-of-the-art methods on this benchmark. All three methods used for comparison leverage the global semantic information in the point cloud during classification and none of them require the additional color information. Our approach achieves the good performance on this data set.

C. Efficiency

The goal of this article is to achieve the real-time performance of point cloud classification, which is fundamental for robotic systems. To demonstrate this, we analyze the memory usage and time cost of our OctreeNet on 3-D point clouds with different sparsity and different size.

Sparsity Analysis: Since the sparsity of 3-D data has a great impact on the efficiency of our OctreeNet, we provide sparsity analyses on the two data sets first. The sparsity is measured in two levels: 1) the sparsity of an entire scene is captured by the number of occupied local regions and 2) we define the sparsity of a local region as follows:

$$s_l = \frac{\#\text{ free voxel}}{\#\text{ total voxel}}. \tag{15}$$

Low sparsity of local regions means more voxels need to be stored and more time it takes for classification.

Fig. 9 shows the proportion of local regions with different sparsity in the two data sets. All local regions in the Semantic3D.net data set have the sparsity over 90%, and the sparsity of more than 90% of local regions is over 96%. The point density of the Oakland data set is even sparse. Almost 90% of local regions have 99% above sparsity. Fig. 10 shows the variation of the number of occupied local regions along with 3-D scene size. These two figures indicate that although the number of measurement points could be millions or billions, the distribution of points in 3-D space is extremely sparse

TABLE III
EVALUATIONS OF DIFFERENT METHODS ON THE OAKLAND DATA SET

Evaluation	**Method**	leaves	buildings	ground	pole	wire	vehicle
Precision	S3DP[27]	**0.96**	0.83	**0.99**	0.51	0.73	0.79
	Hu et al[10]	**0.96**	0.92	0.97	**0.72**	**0.86**	**0.85**
	NAHO[11]	0.95	0.91	**0.99**	0.7	0.66	0.75
	OctreeNet(Ours)	0.94	0.87	0.98	0.6	0.54	0.64
	OctreeNet+CRF(Ours)	0.95	**0.95**	**0.99**	0.68	0.78	0.82
Recall	S3DP[27]	0.93	0.93	0.98	0.67	0.75	0.74
	Hu et al[10]	0.95	**0.97**	0.98	0.62	0.61	0.72
	NAHO[11]	0.94	0.94	**0.99**	0.56	**0.89**	**0.87**
	OctreeNet(Ours)	0.93	0.9	0.98	0.62	0.44	0.59
	OctreeNet+CRF(Ours)	**0.97**	**0.97**	**0.99**	**0.77**	0.53	0.7
F1-score	S3DP[27]	0.94	0.88	0.98	0.58	0.74	0.76
	Hu et al[10]	**0.96**	0.94	0.98	0.67	0.72	0.78
	NAHO[11]	0.94	0.92	**0.99**	0.62	**0.76**	**0.81**
	OctreeNet(Ours)	0.94	0.88	0.98	0.61	0.48	0.61
	OctreeNet+CRF(Ours)	**0.96**	**0.96**	**0.99**	**0.72**	0.63	0.76

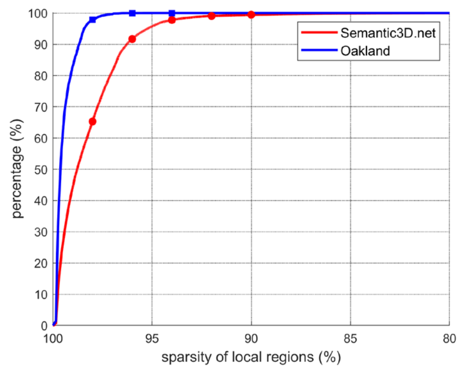


Fig. 9. Proportion of local regions with different sparsity in the two data sets.

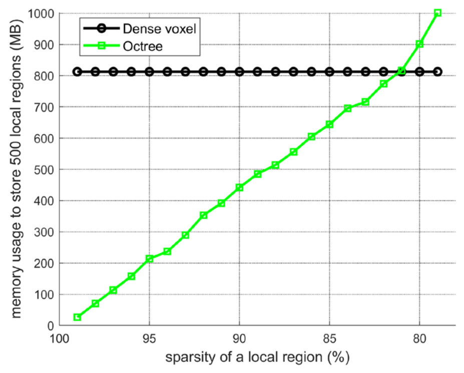


Fig. 11. Memory usages of two data representations: dense voxel and the octree.

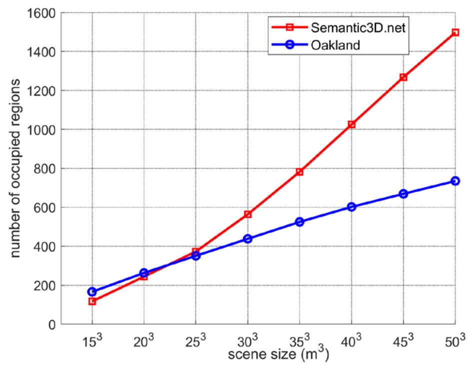


Fig. 10. Variation of the number of occupied regions in a natural scene along with the scene size.

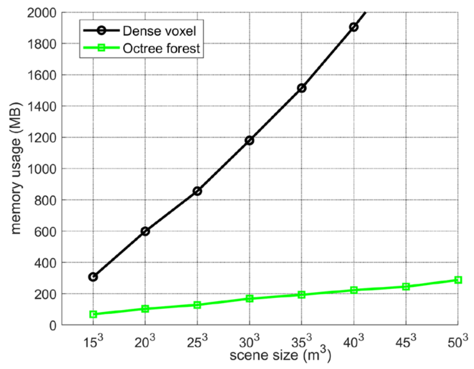


Fig. 12. Memory usages of the two data representations with different scene sizes.

Memory: Fig. 11 shows the memory usages of representing a local region as dense voxels or an octree. Most of the modern networks require a regular input which is dense voxels in the 3-D case. This representation requires a large amount of memory. In contrast, OctreeNet leverages the octree to reduce memory cost. When the sparsity of a local region is over 90%, using the octree representation can save at least 50% memory usage. With the increase of sparsity, up to 95% memory can be saved. Fig. 12 compares the memory usages of two data representations with different scene sizes. As before, using the octree forest to represent point clouds can reduce memory requirements.

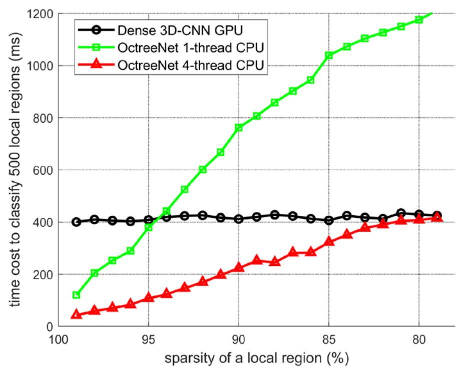


Fig. 13. Time costs of different implementations of our network for classifying local regions with different sparsity.

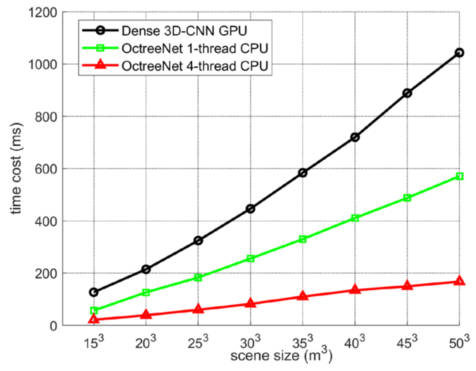


Fig. 14. Time costs of different implementations of our network for classifying a natural scene with different sizes.

Time: In this experiment, the time cost of the inference algorithm proposed in Section III-D is compared with the time cost of a standard implementation of deep network based on Tensorflow. Our inference algorithm is created in C++ with Eigen library and tested on an Intel Core i5-4590 CPU with 8G memory. The time cost of the standard implementation is counted using an NVIDIA GTX 1080 GPU.

Fig. 13 shows the time costs of different implementations for classifying local regions of different sparsity. We observed that when the sparsity is above 94%, a single thread CPU version of our inference algorithm is faster than the standard implementation with GPUs. With multi-thread programming, our algorithm can speed up three times. The multi-thread CPU version shows superior performance than the standard implementation with GPUs until the sparsity is less than 80%.

Fig. 14 shows the variation of time costs of different implementations along with the scene size. An interesting thing is that the single thread CPU version of our algorithm achieves about 2$\times$ speedup over the standard implementation with GPUs. Furthermore, the multi-thread CPU version is even faster and can speed up five times. The time cost of the multi-thread version for classifying a natural scene is less than 200 ms.

Overall: Fig. 15 reports the overall memory and time costs of processing different scenes from the two data sets. In the Semantic3D.net data set, the number of voxels in each scene varies from 0.8 to 3.1 million. In the Oakland data set, each scene contains roughly 35 thousand voxels. Since the unary energy and features in the CRF share the same variables with OctreeNet, the memory cost of the CRF is very limited and not shown explicitly. The time cost of the CRF compared to that of the OctreeNet differs much on these two data sets because the former depends upon the number of voxels while the latter relates to the number of octrees. The ratio between the number of voxels and octrees varies a lot between these two data sets since they have different point densities.

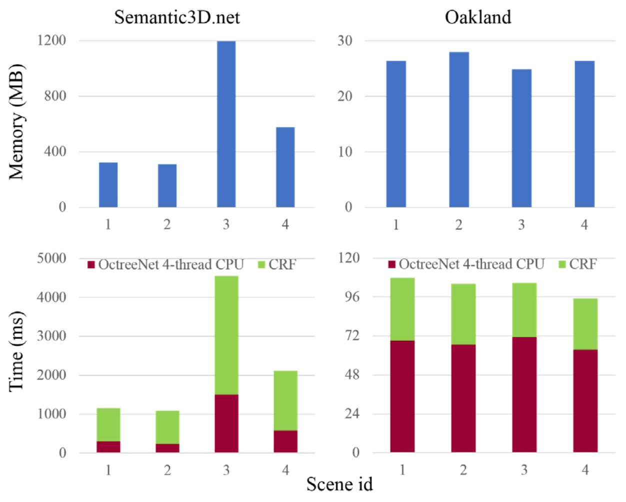


Fig. 15. Overall memory and time costs of classifying different scenes from the two data sets.

TABLE IV
QUALITATIVE COMPARISON OF DIFFERENT METHODS

Method	**Accuracy**	**Memory**	**Time**	**Impleme ntation**	**Real-time**
TMLC-MSR[8]	★★	★★★★	★★	★★★	×
S3DP[27]	★★★	★★★	★★	★	×
Hu et al[10]	★★★	★★	★★★	★	✓
NAHO[11]	★★★	★★★	★	★★★	×
DeePr3SS[17]	★★★★	★★	★★GPU	★★	×
SnapNet[18]	★★★★	★★	★★GPU	★★	×
SEGCloud[6]	★★★★	★	★ GPU	★★	×
OctreeNet	★★	★★★	★★★★	★★★★	✓
OctreeNet+CRF	★★★	★★★	★★★	★★★★	✓

D. Synthesis

An overall comparison among different approaches is given in Table IV. The feature-based method with a local classifier [8] provides reasonable results with limited memory requirements. Since most popular feature extraction methods can be found in the Point Cloud Library, their implementation is easy. When the global contextual information is involved during inference, the overall accuracy can be improved; while the efficiency is decreased with respect to memory usage or computational cost [10], [11], [27]. Some deep-learning-based approaches present dominant performance in accuracy with the requirement of additional color information [6], [17], [18].

Often, a GPU is required to speed up both the training and testing processes. In contrast, our OctreeNet is comparatively efficient by leveraging the sparsity of 3-D data and demands no

GPUs. When a CRF is used for label refinement, our approach presents fine-grained results comparative to the state-of-the-art deep-learning-based methods. Note that our approach only involves coordinates of 3-D points which are direct outputs of a standard laser scanner and no additional color information is required. In the above approaches, only [10] and ours can be applied in real-time applications. However, our OctreeNet is a general model that can be adopted for many classification tasks.

V. Conclusion

This article is focused on the problem of 3-D point cloud classification toward real-time applications of robotic systems. We first introduced the octree forest for 3-D data representation to reduce the memory footprint of 3-D-CNNs. Then, a novel sparse 3-D CNN, OctreeNet, was proposed. Our OctreeNet speeds up convolutions on sparse 3-D data by utilizing the SNNK, which reduces the dense 3-D convolution to small matrix operations and enables computations to be performed directly on the octree structure.

Furthermore, we presented a parallel inference algorithm based on octree searching. A CRF model was used to impose spatial consistency and refine the output. We experimented on two public data sets in outdoor environments with different point densities. Experimental results show the validity and practicality of the proposed approach. Our future study will be focused on extensive testing of our OctreeNet on diverse real-time outdoor navigation tasks.

References

[1] A. Maligo and S. Lacroix, "Classification of outdoor 3D LIDAR data based on unsupervised Gaussian mixture models," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 1, pp. 5–16, Jan. 2017.

[2] Z. Wu *et al.*, " 3D shapenets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, Jun. 2015, pp. 1912–1920.

[3] D. Maturana and S. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Hamburg, Germany, Sep. 2015, pp. 922–928.

[4] M. Engelcke, D. Rao, D. Wang, C. Tong, and I. Posner, "Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Singapore, May/Jun. 2017, pp. 1355–1361.

[5] G. Riegler, A. Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 3577–3586.

[6] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese, "SEGCloud: Semantic segmentation of 3D point clouds," in *Proc. Int. Conf. 3D Vis. (3DV)*, Qingdao, China, Oct. 2017, pp. 537–547.

[7] B. Graham, "Sparse 3D convolutional neural networks," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, London, U.K., Sep. 2016, pp. 1–11.

[8] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3D point clouds with strongly varying density," *ISPRS Ann. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. 3, no. 3, pp. 177–184, Jul. 2016.

[9] D. Munoz, N. Vandapel, and M. Hebert, "Onboard contextual classification of 3-D point clouds with learned high-order Markov random fields," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Kobe, Japan, May 2009, pp. 2009–2016.

[10] H. Hu, D. Munoz, J. Bagnell, and M. Hebert, "Efficient 3-D scene analysis from streaming data," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Karlsruhe, Germany, May 2013, pp. 2297–2304.

[11] M. Najafi, S. T. Namin, M. Salzmann, and L. Petersson, "Non-associative higher-order Markov networks for point cloud classification," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Zürich, Switzerland, Sep. 2014, pp. 500–515.

[12] M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Real-time object classification in 3D point clouds using point feature histograms," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, St. Louis, MO, USA, Oct. 2009, pp. 994–1000.

[13] J. Lalonde, N. Vandapel, and M. Hebert, "Data structures for efficient dynamic processing in 3-D," *Int. J. Robot. Res.*, vol. 26, no. 8, pp. 777–796, Aug. 2007.

[14] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected CRFs with Gaussian edge potentials," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Granada, Spain, Dec. 2011, pp. 109–117.

[15] C. Qi, H. Su, K. Mo, and L. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 77–85.

[16] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer, "SqueezeSegV2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Montreal, QC, Canada, May 2019, pp. 4376–4382.

[17] F. Lawin, M. Danelljan, P. Tosteberg, G. Bhat, F. Khan, and M. Felsberg, "Deep projective 3D semantic segmentation," in *Proc. Int. Conf. Comput. Anal. Images Patterns*, Ystad, Sweden, Aug. 2017, pp. 95–107.

[18] A. Boulch, B. Saux, and N. Audebert, "Unstructured point cloud semantic labeling using deep segmentation networks," in *Proc. Eurograph. Workshop 3D Object Retr.*, Geneva, Switzerland, Apr. 2017, pp. 1–7.

[19] B. Graham, M. Engelcke, and L. van der Maaten, "3D semantic segmentation with submanifold sparse convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 9224–9232.

[20] V. Jampani, M. Kiefel, and P. Gehler, "Learning sparse high dimensional filters: Image filtering, dense CRFS and bilateral neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 4452–4461.

[21] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "SBNet: Sparse blocks network for fast inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 8711–8720.

[22] P. S. Wang, Y. Liu, Y. X. Guo, C. Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3D shape analysis," *ACM Trans. Graph.*, vol. 36, no. 4, p. 72, Jul. 2017.

[23] C. Häne, S. Tulsiani, and J. Malik, "Hierarchical surface prediction for 3D object reconstruction," in *Proc. Int. Conf. 3D Vis. (3DV)*, Qingdao, China, Oct. 2017, pp. 412–420.

[24] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 2088–2096.

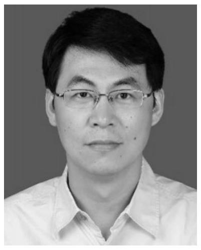
[25] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Lille, France, Feb. 2015, pp. 448–456.

[27] X. Xiong, D, Munoz, J. A. Bagnell, and M. Hebert, "3-D scene analysis via sequenced predictions over points and regions," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Shanghai, China, May 2011, pp. 2609–2616.

Fei Wang received the B.Sc. and M.Sc. degrees in computer science and technology from Dalian Maritime University, Dalian, China, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with the School of Control Science and Engineering, Dalian University of Technology, Dalian.

His research interests are in robotics, deep learning, 3-D data processing, and semantic scene understanding.



Yan Zhuang (M'11) received the B.Sc. and M.Sc. degrees from Northeastern University, Shenyang, China, in 1997 and 2000, respectively, and the Ph.D. degree from the Dalian University of Technology, Dalian, China, in 2004, all in control theory and engineering.

He joined the Dalian University of Technology as a Lecturer in 2005 and became an Associate Professor in 2007, where he is currently a Professor with the School of Control Science and Engineering. His research interests include mobile robot 3-D mapping, outdoor scene understanding, 3-D-laser-based object recognition, and 3-D scene recognition and reconstruction.

Hong Gu received the B.Sc. and M.Sc. degrees from the Shenyang University of Technology, Shenyang, China, in 1982 and 1984, respectively, and the Ph.D. degree from Zhejiang University, Hangzhou, China, in 1990.

Since 1996, he has been a Full Professor with the Dalian University of Technology, Dalian, China. His current research interests include machine learning, big data, and bioinformatics.

Huosheng Hu (M'94–SM'01) received the M.Sc. degree in industrial automation from Central South University, Changsha, China, in 1982, and the Ph.D. degree in robotics from the University of Oxford, Oxford, U.K., in 1993.

He is currently a Professor with the School of Computer Science and Electronic Engineering, University of Essex, Colchester, U.K., where he is leading the Robotics Research Group. His research interests include behavior-based robotics, human-robot interaction, service robots, embedded systems, data fusion, learning algorithms, mechatronics, and pervasive computing. He has published around 450 articles in journals, books, and conferences in these areas.

Prof. Hu is a founding member of the IEEE Robotics and Automation Society Technical committee on Networked Robots, a fellow of Institution of Engineering and Technology (IET) and the Institute of Measurement & Control (InstMC), London, U.K., and a Senior Member of Association for Computing Machinery (ACM). He received a number of best article awards. He has been a Program Chair or a member of Advisory/Organizing Committee for many international conferences such as IEEE International Conference on Robotics and Automation (ICRA), IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE International Conference on Mechatronics and Automation (ICMA), IEEE International Conference on Robotics and Biomimetics (ROBIO), IEEE International Conference on Information and Automation (ICIA), IEEE International Conference on Automation and Logistics (ICAL), and International Association of Science and Technology for Development (IASTED) International Conference on Robotics Applications (RA), IASTED International Conference on Control and Applications (CA), and IASTED International Conference on Computational Intelligence (CI) conferences. He currently serves as the Editor-in-Chief for the *International Journal of Automation and Computing*, Editor-in-Chief for the online *Robotics Journal*, and the Executive Editor for the *International Journal of Mechatronics and Automation*.