

# 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks

Benjamin Graham  
Facebook AI Research  
benjamin@fb.com

Martin Engelcke  
University of Oxford  
martin@robots.ox.ac.uk

Laurens van der Maaten  
Facebook AI Research  
lvdmaaten@fb.com

## Abstract

*Convolutional networks are the de-facto standard for analyzing spatio-temporal data such as images, videos, and 3D shapes. Whilst some of this data is naturally dense (e.g., photos), many other data sources are inherently sparse. Examples include 3D point clouds that were obtained using a LiDAR scanner or RGB-D camera. Standard “dense” implementations of convolutional networks are very inefficient when applied on such sparse data. We introduce new sparse convolutional operations that are designed to process spatially-sparse data more efficiently, and use them to develop spatially-sparse convolutional networks. We demonstrate the strong performance of the resulting models, called submanifold sparse convolutional networks (SS-CNs), on two tasks involving semantic segmentation of 3D point clouds. In particular, our models outperform all prior state-of-the-art on the test set of a recent semantic segmentation competition.*

## 1. Introduction

Convolutional networks (ConvNets) constitute the state-of-the-art method for a wide range of tasks that involve the analysis of data with spatial and/or temporal structure, such as photos, videos, or 3D surface models. While such data frequently comprises a densely populated (2D or 3D) grid, other datasets are naturally sparse. For instance, handwriting is made up of one-dimensional lines in two-dimensional space, pictures made by RGB-D cameras are three-dimensional point clouds, and polygonal mesh models form two-dimensional surfaces in 3D space.

The curse of dimensionality applies, in particular, to data that lives on grids that have three or more dimensions: the number of points on the grid grows exponentially with its dimensionality. In such scenarios, it becomes increasingly important to exploit data sparsity whenever possible in order to reduce the computational resources needed for data processing. Indeed, exploiting sparsity is paramount when

Figure 1: Examples of 3D point clouds of objects from the ShapeNet part-segmentation challenge [23]. The colors of the points represent the part labels.

analyzing, e.g., RGB-D videos which are sparsely populated 4D structures.

Traditional convolutional network implementations are optimized for data that lives on densely populated grids, and cannot process sparse data efficiently. More recently, a number of convolutional network implementations have been presented that are tailored to work efficiently on sparse data [3, 4, 18]. Mathematically, some of these implementations are identical to regular convolutional networks, but they require fewer computational resources in terms of FLOPs and/or memory [3, 4]. Prior work uses a sparse version of the `im2col` operation that restricts computation and storage to “active” sites [4], or uses the voting algorithm from [22] to prune unnecessary multiplications by zeros [3]. OctNets [18] modify the convolution operator to produce “averaged” hidden states in parts of the grid that are outside the region of interest.

One of the downsides of prior sparse implementations of convolutional networks is that they “dilate” the sparse data in every layer by applying “full” convolutions. In this work,

---

Work done while interning at Facebook AI Research

| Method                                         | Average IoU    |
|------------------------------------------------|----------------|
| NN matching with Chamfer distance              | 77.57%         |
| Synchronized Spectral CNN [11]                 | 84.74%         |
| Pd-Network (extension of Kd-Network [10])      | 85.49%         |
| Densely Connected PointNet (extension of [17]) | 84.32%         |
| PointCNN                                       | 82.29%         |
| Submanifold SparseConvNet (Section 6.5)        | <b>85.98 %</b> |

Table 1: Average intersection-over-union (IoU) of six approaches on the test set of a recent part-based segmentation competition on ShapeNet [23]. Higher is better. Our SSCNs outperform all alternative approaches.

we show that it is possible to create convolutional networks that *keep the same level of sparsity throughout the network*. Therefore, it becomes practical to train networks with significantly more layers, *e.g.*, ResNets [7] and DenseNets [9].

To this end, we develop a new implementation for performing *sparse convolutions* (SCs) and introduce a novel convolution operator termed *submanifold sparse convolution* (SSC).<sup>1</sup> We use these operators as the basis for submanifold sparse convolutional networks (SSCNs) that are optimized for efficient semantic segmentation of 3D point clouds, *e.g.*, on the examples shown in Figure 1.

In Table 1, we present the performance of SSCNs on the test set of a recent part-based segmentation competition [23] and compare it to some of the top-performing entries in the competition: SSCNs outperform all of these entries. Source code for our library is publicly available online.<sup>2</sup>

## 2. Related Work

Our work primarily builds upon previous literature on sparse convolutional networks [3, 4], and image segmentation using dense convolutional networks [14, 19, 24]. Examples of applications of dense 3D convolutions on volumetric data include classification [15] and segmentation [2]; these methods suffer from high memory usage and slow inference, limiting the size of models that can be used.

Methods for processing 3D point clouds without voxelization have also been developed [10, 17]. This may seem surprising given the dominance of ConvNets for processing 2D inputs; it is likely due to the computational obstacles involved in using dense 3D convolutional networks.

Prior work on sparse convolutions implements a convolutional operator that increases the number of active sites with each layer [3, 4]. In [4], all sites that have at least one “active” input site are considered as active. In [3], a greater

degree of sparsity is attained *after* the convolution has been calculated by using ReLUs and a special loss function. In contrast, we introduce *submanifold* sparse convolutions that fix the location of active sites so that the sparsity remains unchanged for many layers. We show that this makes it practical to train deep and efficient networks similar to VGG networks [20] or ResNets [7], and that it is well suited for the task of point-wise semantic segmentation.

OctNets [18] are an alternative form of sparse convolution. Sparse voxels are stored in oct-trees: a data structure in which the grid cube is progressively subdivided into  $2^3$  smaller sub-cubes until the sub-cubes are either empty or contain a single active site. OctNet operates on the surfaces of empty regions, so a size-3 OctNet convolution on an empty cube of size  $8 \times 8 \times 8$  requires 23% of the calculation of a dense 3D convolution. Conversely, submanifold convolutions require no calculations in empty regions.

Another approach to segmenting point clouds is to avoid voxelizing the input, which may lead to a loss of information due to the finite resolution. This can be done by either using carefully selected data structures such as Kd-trees [10], or by directly operating on the unordered set of points [17]. Kd-Networks [10] build a Kd-tree by recursively partitioning the space along the axis of largest variation until each leaf of the tree contains one input point. This takes  $O(N \log N)$  time for  $N$  input points. PointNet [17] uses a pooling operation to produce a global feature vector.

Fully convolutional networks (FCNs) were proposed in [14] as a method of 2D image segmentation; FCNs make use of information at multiple scales to preserve low-level information to accurately delineate object boundaries. U-Nets [19] extend FCNs by using convolutions to more accurately merge together the information from the different scales before the final classification stage; see Figure 4.

## 3. Spatial Sparsity for Convolutional Networks

We define a  $d$ -dimensional convolutional network as a network that takes as input a  $(d + 1)$ -dimensional tensor: the input tensor contains  $d$  spatio-temporal dimensions (such as length, width, height, time, *etc.*) and one additional feature-space dimension (*e.g.*, RGB color channels or surface normal vectors). The input corresponds to a  $d$ -dimensional grid of *sites*, each of which is associated with a feature vector. We define a site in the input to be *active* if any element in the feature vector is not in its *ground state*, *e.g.*, if it is non-zero<sup>3</sup>. If the data is not naturally sparse, thresholding may be used to eliminate input sites at which the feature vector is within a small distance from the ground state. Note that even though the input tensor is  $(d + 1)$ -dimensional, activity is a  $d$ -dimensional phenomenon: entire lines along the feature dimension are either active or inactive.

<sup>1</sup>These operators appeared earlier in an unpublished report [5], including experiments on several classification datasets.

<sup>2</sup><https://github.com/facebookresearch/SparseConvNet>

<sup>3</sup>Note that the ground state does not necessarily have to be zero, in particular, when convolutions with a bias term are used.

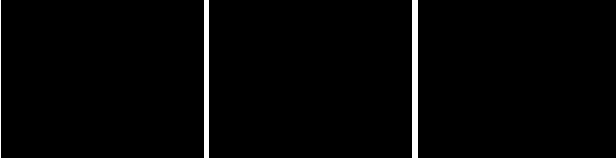


Figure 2: Example of “submanifold” dilation. **Left:** Original curve. **Middle:** Result of applying a regular  $3 \times 3$  convolution with weights  $1/9$ . **Right:** Result of applying the same convolution again. Regular convolutions substantially reduce the feature sparsity with each convolutional layer.

Similarly, the hidden layers of a  $d$ -dimensional convolutional network are represented by  $d$ -dimensional grids of feature-space vectors. When propagating the input data through the network, a site in a hidden layer is active if any of the sites in the layer that it takes as input is active. (Note that when using size-3 convolutions, each site is connected to  $3^d$  sites in the hidden layer below.) Activity in a hidden layer thus follows an inductive definition in which each layer determines the set of active states in the next layer. In each hidden layer, inactive sites all have the same feature vector: the one corresponding to the ground state. The value of the ground state only needs to be calculated once per forward pass at training time, and only once for all forward passes at test time. This allows for substantial savings in computation and memory use.

We argue that the framework described above is unduly restrictive, in particular, because the convolution operation has not been modified to accommodate the sparsity of the input data. If the input data contains a single active site, then after applying a  $3^d$  convolution, there will be  $3^d$  active sites. Applying a second convolution of the same size will yield  $5^d$  active sites, and so on. This rapid growth of the number of active sites is a poor prospect when implementing modern convolutional network architectures that comprise tens or even hundreds of convolutional layers, such as VGG networks, ResNets, or DenseNets [8, 9, 20].

Of course, convolutional networks are not often applied to inputs that only have a single active site, but the aforementioned dilation problems are equally problematic when the input data comprises one-dimensional curves in spaces with two or more dimensions, or two-dimensional surfaces in three or more dimensions. We refer to this problem as the “submanifold dilation problem”. Figure 2 illustrates the problem: even when we apply small  $3 \times 3$  convolutions on this grid, the sparsity of the grid rapidly disappears.

## 4. Submanifold Convolutional Networks

We explore a simple solution to the submanifold dilation problem that restricts the output of the convolution only to the set of active input points. A potential problem of this ap-



Figure 3:  $\text{SSC}(\cdot, \cdot, 3)$  receptive field centered at different active spatial locations. Active locations in the field are shown in green. Red locations are ignored by SSC so the pattern of active locations remains unchanged.

proach is that hidden layers in the network may not receive all the information they require to classify the input data: in particular, two neighboring connected components are treated completely independently. We resolve this problem by using convolutional networks that incorporate pooling or strided convolution operations. Such operations are important in the sparse convolutional networks<sup>4</sup> we study, as they allow information to flow between disconnected components in the input. The closer components are spatially, the fewer strided operations are necessary for components to “communicate” in their intermediate representations.

### 4.1. Sparse Convolutional Operations

We define a sparse convolution  $\text{SC}(m, n, f, s)$  with  $m$  input feature planes,  $n$  output feature planes, a filter size of  $f$ , and stride  $s$ . An SC convolution computes the set of active sites in the same way as a regular convolution: it looks for the presence of any active sites in its receptive field of size  $f^d$ . If the input has size  $(-f + s)/s$ . Unlike regular convolutions or the sparse convolutions of [4], an SC convolution discards the ground state for non-active sites by assuming that the input from those sites is zero. This seemingly small change reduces the computational cost by circa 50%.

**Submanifold sparse convolution.** The main contribution of this paper is the definition of another sparse convolution. Let  $f$  denote an odd number. We define a submanifold sparse convolution  $\text{SSC}(m, n, f)$  as a modified  $\text{SC}(m, n, f, s = 1)$  convolution. First, we pad the input with  $(f - 1)/2$  zeros on each side, so that the output will have the same size as the input. Next, we restrict an output site to be active iff the site at the corresponding site in the input is active (*i.e.*, if the central site in the receptive field is active). Whenever an output site is determined to be active, its output feature vector is computed by the SSC convolution; see Figure 3 for an illustration. Table 2 displays the computational and memory requirements of a regular convolution (C) operation and our SC and SSC convolutions.

<sup>4</sup>Our “sparse convolutional networks” are networks designed to operate on spatially-sparse input data; they do not have sparse parameters [12, 13].

| Active      | Type   | C        | SC    | SSC   |
|-------------|--------|----------|-------|-------|
| Yes         | FLOPs  | $3^d mn$ | $amn$ | $amn$ |
|             | Memory | $n$      | $n$   | $n$   |
| No, $a > 0$ | FLOPs  | $3^d mn$ | $amn$ | 0     |
|             | Memory | $n$      | $n$   | 0     |
| No, $a = 0$ | FLOPs  | $3^d mn$ | 0     | 0     |
|             | Memory | $n$      | 0     | 0     |

Table 2: Computational and memory requirements of three convolutions: regular convolution (C), sparse convolution (SC), and submanifold sparse convolution (SSC). We consider convolutions with size  $f = 3$  and padding  $s = 1$  at a single location in  $d$  dimensions. Herein,  $a$  is the number of active inputs to the spatial location,  $m$  the number of input feature planes, and  $n$  the number of output feature planes.

SSC convolutions are similar to OctNets [18] in that they preserve sparsity structure. However, unlike OctNets, empty space imposes no computational or memory overhead in the implementation of SSC convolutions.

**Other operators.** To construct convolutional networks using SC and SSC, we also need activation functions, batch normalization, and pooling. Activation functions are defined as usual, but are restricted to the set of active sites. Similarly, we define batch normalization in terms of regular batch normalization applied over the set of active sites. Max-pooling  $MP(f, s)$  and average-pooling  $AP(f, s)$  operations are defined as a variant of  $SC(\cdot, \cdot, f, s)$ .  $MP$  takes the maximum of the zero vector and the input feature vectors in the receptive field.  $AP$  calculates  $f^{-d}$  times the sum of the active input vectors. We also define a deconvolution [25] operation  $DC(\cdot, \cdot, f, s)$  as an inverse of the  $SC(\cdot, \cdot, f, s)$  convolution. The set of active output sites from a  $DC$  convolution is exactly the same as the set of input active sites to the corresponding  $SC$  convolution: the connections between input and output sites are simply inverted.

## 4.2. Implementation

To implement (S)SC convolutions efficiently, we store the state of a input/hidden layer in two parts: a hash table<sup>5</sup> and a matrix. The matrix has size  $a \times m$  and contains one row for each of the  $a$  active sites. The hash table contains (location, row) pairs for all active sites: the location is a tuple of integer coordinates, and the row number indicates the corresponding row in the feature matrix. Given a convolution with filter size  $f$ , let  $F = \{0, 1, \dots, f-1\}^d$  denote the spatial size of the convolutional filter. Define a *rule book* to be a collection  $R = (R_i : i \in F)$  of  $f^d$  integer matrices

<sup>5</sup><https://github.com/sparsehash/sparsehash>

each with two columns. To implement an  $SC(m, n, f, s)$  convolution, we:

1. Iterate once through the input hash-table. We build the output hash table and rule book on-the-fly by iterating over points in the input layers, and all the points in the output layer that can see them. When an output site is first visited, a new entry is created in the output hash table. For each active input  $x$  located at point  $i$  in the receptive field of an output  $y$ , add a row (input-hash( $x$ ), output-hash( $y$ )) to rule book element  $R_i$ .
2. Initialize the output matrix to all zeros. For each  $i \in F$ , there is a parameter matrix  $W^i \in \mathbb{R}^{m \times n}$ . For each row  $(j, k)$  in  $R_i$ , multiply the  $j$ -th row of the input feature matrix by  $W^i$  and add it to the  $k$ -th row of the output feature matrix. This can be done efficiently on GPUs because it is a matrix-matrix multiply-add.

To implement an SSC convolution, we re-use the input hash table for the output, and construct an appropriate rule book. Note that because the sparsity pattern does not change, the same rule book can be re-used in the network until a pooling or subsampling layer is encountered.

If there are  $a$  active points in the input layer, the cost of building the input hash-table is  $O(a)$ . For FCN and U-Net networks, assuming the number of active sites reduces by a multiplicative factor with each downsampling operation, the cost of building all the hash-tables and rule-books is also  $O(a)$  regardless of the depth of the network.

The above implementation differs from [4] in that the cost of calculating an output site is proportional to the number of active inputs, rather than to the size of the receptive field. For SC convolutions this is similar to the voting algorithm [22, 3] – the filter weights are never multiplied with inactive input locations – but for SSC convolutions, the implementation is less computationally intensive than voting as there is no interaction between active input locations and inactive neighboring output locations.

## 5. Submanifold FCNs and U-Nets for Semantic Segmentation

Three-dimensional semantic segmentation involves the segmentation of 3D objects or scenes represented as point clouds into their constituent parts; each point in the input cloud must be assigned a part label. As progress has been made in the segmentation of 2D images using convolutional networks [14, 19, 24], interest in the problem of 3D semantic segmentation has grown recently. Interest was fueled, in particular, by a new dataset for the part-based segmentation of 3D objects, and an associated competition [23].

We use a sparse voxelized input representation similar to [3, 4] and a combination of SSC convolutions and strided

SC convolutions to construct sparse variants of the popular FCN [14] and U-Net [2] networks. The resulting networks are illustrated in Figure 4; see the caption for details. We refer to these networks as *submanifold sparse convolutional networks* (SSCNs), because they process low-dimensional data living in a space of higher dimensionality.<sup>6</sup>

The basic building blocks for our networks are “pre-activated”  $\text{SSC}(\cdot, \cdot, 3)$  convolutions. Each convolution is preceded by batch normalization and a ReLU non-linearity. In addition to FCN and U-Nets with standard convolutional layers, we also experiment with variants of these networks that use pre-activated residual blocks [8] that contain two  $\text{SSC}(\cdot, \cdot, 3)$  convolutions. Herein, the residual connections are identity functions: the number of input and output features are equal. Whenever the networks reduce the spatial scale by a factor of two, we use  $\text{SC}(\cdot, \cdot, 2, 2)$  convolutions. Our implementation of FCNs upsamples feature maps to their original resolution rather than performing deconvolutions using residual blocks. This substantially reduces the number of parameters and mult-add operations in the FCN.

## 6. Experiments

In this section, we perform experiments with SSCNs on the ShapeNet competition dataset [23]. We compare SSCNs against three strong baseline models in terms of performance and computational cost: (1) shape contexts [1], (2) dense 3D convolutional networks, and (3) multi-view 2D convolutional networks [21]. Throughout our experimental evaluation, we focus on the trade-off between segmentation accuracy and computational efficiency measured in FLOPs<sup>7</sup>. In a second set of experiments, we also study SSCN performance on the NYU Depth (v2) dataset [16].

### 6.1. Dataset

The ShapeNet segmentation dataset [23] comprises 16 different object categories (plane, chair, hat, *etc.*), each of which is composed of up to 6 different parts. For instance, a “plane” is segmented into “wings”, “engine”, “body”, and “tail”. Across all object categories, the dataset contains a total of 50 different object part classes. Each object is represented as a 3D point cloud that was obtained by sampling points uniformly from the surface of the underlying CAD model. Each point cloud contains between 2,000 and 3,000 points. To increase the size of the validation set, we re-split the training and validation sets using the first bit of the MD5 hash of the point cloud files to obtain a training set with 6,955 examples and a validation set with 7,052 examples. The test set contains 2,874 examples.

<sup>6</sup>We note that this is a slight abuse of the term “submanifold”. We emphasize that the data on which these networks are applied may contain multiple connected components, and even a mixture of 1D and 2D objects embedded in 3D space.

<sup>7</sup>We ignore the FLOPs from the final classification layer.

| View type   | IoU accuracy |
|-------------|--------------|
| Aligned     | 63.5%        |
| Random pose | 47.8%        |

Table 3: Accuracy of segmentation classifiers based on shape-context features on (1) the original ShapeNet dataset and (2) a variant of the dataset in which objects are randomly rotated. The results show that removing the alignment of the ShapeNet objects via random 3D rotations makes the segmentation problem more challenging.

In the original dataset, the objects are axis-aligned: for instance, rockets always point along the z-axis. To make the problem more challenging, we perform a random 3D translation and rotation on each point cloud before classifying it. The results in Table 3 show that removing the alignment, indeed, makes the segmentation task more challenging.

To evaluate the accuracy of our models, we adopt the intersection-over-union (IoU) metric of [23]. The IoU is computed for each part per object category and averaged over parts and examples for the category to produce a “per-category IoU”. This way of averaging the IoU scores rewards models that make accurate predictions even for object-parts that are very small: small parts have the same weight in the accuracy measure as larger parts. The final accuracy measure is obtained by taking a weighted average of the per-category IoUs, using the fraction of training examples per category as weights.

### 6.2. Details of Experimental Setup

In all experiments, the same data pre-processing procedure is used. Specifically, each point cloud is centered and re-scaled to fit into a sphere with diameter  $S$ ; scale  $S$  determines the size of the voxelized representation. We use  $S \in \{16, 32, 48\}$  in our experiments. At scale  $S = 48$ , the voxels are approximately 99% sparse. In experiments with dense convolutional networks, we place the sphere with random translations and rotations in a grid of size  $S$ . For SSCNs, we place the sphere similarly in a grid of size  $4S$ . To voxelize the point cloud, we measure the number of points per voxel and normalize them so that non-empty voxels have an average density of one.

Networks are trained using the same optimization hyperparameters, unless otherwise noted. We use stochastic gradient descent (SGD) with a momentum of 0.9, Nesterov updates, and  $L_2$  weight decay of  $10^{-4}$ . The initial learning rate is set to 0.1, and the learning rate is decayed by a factor of  $e^{-0.04}$  after every epoch. We train all networks for 100 epochs using a batch size of 16. We train a single network on all 16 object categories jointly using a multi-class negative log-likelihood loss function over all 50 part labels.

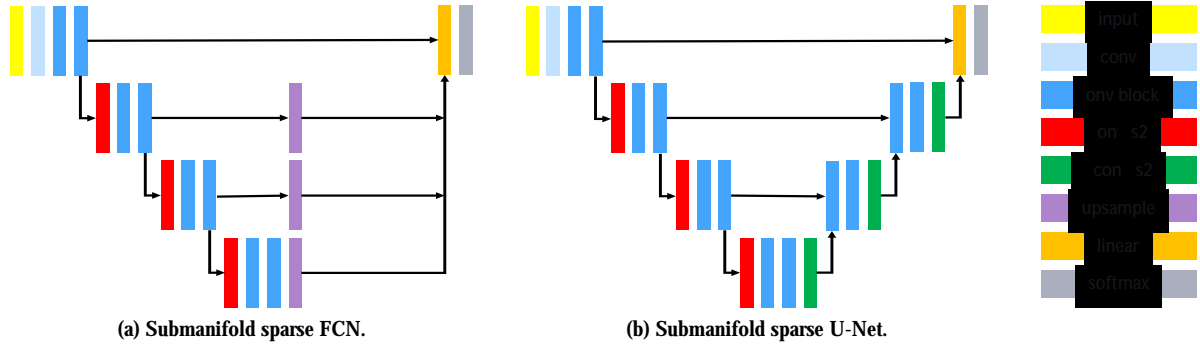


Figure 4: Illustrations of our submanifold sparse FCN (a) and U-Net (b) architectures. Dark blue boxes represents one or more “pre-activated”  $\text{SSC}(\cdot, \cdot, 3)$  convolutions, which may have residual connections. Red boxes represent size-2, stride-2 downsampling convolutions; green deconvolutions “invert” these convolutions. Purple upsampling boxes perform “nearest-neighbor” upsampling. The final linear and softmax layers are applied separately on each active input voxel.

We experiment with two types of SSCN network architectures. The first architecture (C3) operates on a single spatial resolution by stacking  $\text{SSC}(\cdot, \cdot, 3)$  convolutions; we use with 8, 16, 32, or 64 filters per layer, and 2, 4, or 6 layers. The second architecture type comprises FCNs and U-Nets with three layers of downsampling. These networks have 8, 16, 32, or 64 filters in the first layer, and double the number of filters each time the data is downsampled. For the convolutional blocks in these networks, we use stacks of 1, 2, or 3 SSC convolutions, or stacks of 1, 2, or 3 residual blocks.

**Details on testing.** At test time, we only compute softmax probabilities for part labels that actually appear in the object that is being segmented, *i.e.*, we assume the models know the category of the object they are segmenting. Softmax probabilities for irrelevant part classes are set to zero (and the distribution over part labels is re-normalized).

For each of the three network types (C3, FCN, and U-Net), we train a range of models with varying sizes, as described above, and monitor their accuracy on the validation set. For each network type, we select the networks that correspond to local maxima in the accuracy *vs.* FLOPs curve, and report test set accuracies for those networks. Akin to multi-crop testing that is commonly in image classification, we ensemble model predictions over multiple views: we generate  $k$  different views of the object by randomly rotating them, and average the model predictions for each point over the  $k$  different views of the object.

### 6.3. Baselines

In addition to SSCNs, we consider three baseline models in our experiments: (1) shape contexts [1], (2) dense 3D convolutional networks, and (3) multi-view 2D convolutional networks [21]. We describe the details of the four baseline models separately below.

**Shape contexts.** Inspired by [1], we define a voxelized shape context vector. Specifically, we define a ShapeContext layer as a special case of the  $\text{SSC}(1, 27, 3, 1)$  submanifold convolution operator: we set the weight matrix of the operator to be a  $27 \times 27$  identity matrix so that it accumulates the voxel intensities in its  $3 \times 3 \times 3$  neighborhood. We scale the data using average pooling with sizes 2, 4, 8, and 16 to create four additional views. Combined, this produces a 135-dimensional feature vector for each voxel. This feature vector is fed into a non-convolutional multi-layer perceptron (MLP) with two hidden layers, followed by a 50-class softmax classifier. The MLPs have 32, 64, 128, 256, or 512 units per layer. At test time, we use multi-view testing with  $k = 3$ .

**Dense 3D convolutional networks.** For dense 3D convolutional networks, we simply considered dense versions of the SSCN networks. Due to computational constraints, we restricted the FCN and U-Net convolutional blocks to a single C3-layer. We trained some of the models with a reduced learning rate due to numerical instabilities we observed during training. Again, we use  $K = 3$  multi-view testing.

**Convolutional networks on multi-view 2D projections.** This baseline model discards the inherent 3D structure of the data by projecting the point cloud into a two-dimensional view by assuming infinite focal length, applying a 2D convolutional network on this projection, and averaging the predictions over multiple views. An immediate advantage of this approach is that well-studied models from 2D vision can be used out-of-the-box without further adaptations. Moreover, the computational cost scales with the surface area, rather than the volume of the point cloud.

In our implementation of this approach, we first convert the point clouds into a 3D grid of size  $S \times S \times S$  as we

(a) Comparison with baseline methods. (b) Comparison between architectures (see 6.2). (c) SSCN with different scales,  $S$ .

Figure 5: Average interaction-over-union (IoU) on the test set of SSCNs trained for 3D semantic segmentation on the ShapeNet competition data set (higher is better).

did for the previous baseline. We then project to a plane of size  $S \times S$ , *i.e.*, a face of the cube, with two feature channels. One feature channel is the first visible, non-zero voxel along the corresponding column. The second channel is the distance to the visible voxel, normalized to the range  $[0, 2]$ , analogous to the depth channel of an RGB-D image. Our network architectures are two-dimensional versions of the dense 3D convolutional networks described above.

During training, a random projection of the point cloud is passed into the model. Points in the point cloud that fall into the same voxel are given the same prediction. Some voxels are occluded by others—the network receives no information on the occluded voxels. We modify the multi-view testing procedure to take into account the occlusion of voxels. Similar to before, predictions are performed using a weighted sum over  $k$  random projections. We found that 2D networks require more views to obtain high accuracy and use up to  $k = 10$  views. Voxels that are observed in the 2D projection are given a weight of 1. The weight of occluded voxels decays exponentially with the distance to the voxel that occludes them which guarantees a prediction for each point, even if that point is occluded in all views.

## 6.4. Results

In Figure 5, we report the average IoU on the ShapeNet test set of a range of differently sized variants of: (1) the three baseline models and (2) our submanifold C3, FCNs, and U-Nets. The average IoU is shown as a function of the number of multiplication-addition operations (FLOPs) required by the models for computing the predictions. Please note that the results in the figure are not directly comparable with those in [23] because we are testing the models in the more challenging “random-pose” setting.

**SSCNs vs. baselines.** Figure 5(a) compares SSCNs with the three baselines.<sup>8</sup> The results show that shape context features, multi-view 2D ConvNets, and dense 3D ConvNets perform roughly on par in terms of accuracy per FLOP. SSCN networks outperform all baseline models by a substantial margin. For instance, at  $10^8$  FLOPs, the average IoU of SSCNs is 6-8% higher than that of the baselines. Importantly, our results show that restricting information to travel along submanifolds in the data does not hamper the performance of SSCNs, whilst it does lead to considerable computational and memory savings that can be exploited to train larger models with better accuracies.

**Ablation.** In Figure 5(b), we compare the three SSCN architectures presented in Section 6.2. We observe that SSCNs involving downsampling and upsampling operations (FCNs and U-Nets) outperform SSCNs operating on a single spatial resolution and we conjecture that this is due to the increased receptive field obtained by downsampling.

Figure 5(c) shows the performance of SSCNs at three different scales  $S$  (using all three architectures: C3, FCN, and U-Net). We observe that the performance of SSCNs is similar for different values of  $S$ , in particular, for low numbers of FLOPs. At a higher number of FLOPs, the models operating at a larger scale perform slightly better.

## 6.5. Results on Competition Data

To compare SSCNs with the entries to the competition in [23], we also trained an FCN-SSCN on the aligned point clouds. In this experiment, we performed data augmenta-

<sup>8</sup>The number of FLOPs reported for shape contexts may be slightly misleading: the computational costs of calculating shape context features is not reflected in the number of FLOPs, as it involves integer arithmetic.

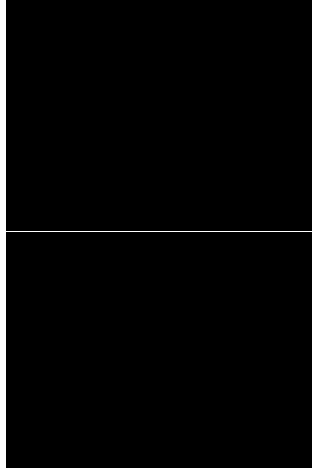


Figure 6: Two examples of RGB-D images from the NYU Depth dataset (v2) [16]. Each example comprises an RGB image (left) and a corresponding depth image (right).

tion using random affine transforms. We set  $S = 24$  and use 64 filters in the input layer, three levels of downsampling, and two residual blocks per spatial resolution. The results of 10-view testing are compared with the competition entries in Table 1. With a test error of 85.98%, our network outperforms other methods by 0.49% IoU.

## 6.6. Semantic Segmentation of Scenes

We also performed experiments on the NYU Depth dataset (v2) [16] for semantic segmentation of scenes rather than objects. The dataset contains 1,449 RGB-D images, which are semantically segmented into 894 different classes. Figure 6 shows two examples from the dataset: each example comprises an RGB image and the associated depth map. Following [6, 14], we crop the images and reduce the number of classes to 40. To assess the performance of our models, we measure their pixel-wise classification accuracy. We compare our models to a 2D FCN [14].

We perform experiments with two differently sized SSCN-FCN networks. Network A has 16 filters in the input layer, and one  $\text{SSC}(\cdot, \cdot, 3)$  convolution per level. Network B has 24 filters in the input layer, and two  $\text{SSC}(\cdot, \cdot, 3)$  convolutions per level. Both networks use eight levels of downsampling. We increase the number of filters in the networks when downsampling: in particular, we add 16 (A) or 24 (B) features every time we reduce the scale.

We use the depth information to convert the RGB-D images into a 3D point cloud. Each point in the cloud has the three (RGB) features that were normalized to the range  $[-1, 1]$ , and a fourth indicator features that is set to 1 for each point in the point cloud. The indicator feature is needed to model the case in which a voxel is active but all three colour channels have a value of zero. During training,

| Network     | k | Accuracy | FLOPs  | Memory |
|-------------|---|----------|--------|--------|
| 2D FCN [14] | 1 | 61.5%    | 28.50G | 135.7M |
| SSCN-FCN A  | 1 | 64.1%    | 1.09G  | 5.2M   |
|             | 4 | 66.9%    | 4.36G  | 20.7M  |
| SSCN-FCN B  | 1 | 66.4%    | 4.50G  | 11.6M  |
|             | 4 | 68.5%    | 17.90G | 46.4M  |

Table 4: Semantic segmentation performance of five different convolutional networks on the NYU Depth test set (v2) on 40 classes. The table displays the pixel-wise classification accuracy, the computational costs (in FLOPs), and the memory requirements (c.f. Table 2) of each of the models.

we perform data augmentation by applying random affine transformations to the point cloud. Before voxelizing the point cloud, we downscale by a factor of two, and place the points into the model’s receptive field. We form voxels by averaging the feature vectors of the points corresponding to the voxel. At test time, we experiment with both single-view and multi-view predictions (*i.e.*, with  $k = 1$  and  $k = 4$ ).

The results of our experiments on the NYU Depth dataset (v2) are presented in Table 4. In line with our previous results, the results in the table show that SSCNs outperform 2D FCN in terms of pixel accuracy by up to 7%. At the same time, SSCNs also substantially reduce the computational requirements of the prediction model.

To verify that SSCN-FCN-A actually uses depth information, we repeat the previous experiment whilst setting all the depth values to zero; this prevents the SSCN from exploiting depth information. We observe: (1) a reduction of FLOPs by 60%, as there are fewer active voxels; and (2) a drop in accuracy from 64.1% to 50.8%, which demonstrates that SSCNs do use 3D structure when performing segmentation. This confirms that SSCN-FCN-A, indeed, uses depth information for making predictions.

## 7. Conclusions

In this paper, we introduced submanifold sparse convolutional networks (SSCNs) for the efficient processing of high-dimensional, sparse input data. We demonstrated the efficacy of SSCNs in a series of experiments on semantic segmentation of three-dimensional point clouds. Specifically, our empirical evaluation of SSCN networks shows that they outperform a range of state-of-the-art approaches for this problem, both when identifying parts within an object and when recognizing objects in a larger scene. Moreover, SSCNs are computationally efficient compared to alternative approaches. Our software toolbox for constructing SSCNs is publicly available at <https://github.com/facebookresearch/SparseConvNet>.

## References

- [1] S. Belongie, J. Malik, and J. Puzicha. Shape Matching and Object Recognition using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.
- [2] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2016.
- [3] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3Deep: Fast Object Detection in 3D Point Clouds using Efficient Convolutional Neural Networks. *IEEE International Conference on Robotics and Automation*, 2017.
- [4] B. Graham. Sparse 3D Convolutional Neural Networks. *British Machine Vision Conference*, 2015.
- [5] B. Graham and L. van der Maaten. Submanifold Sparse Convolutional Networks. 2017. <https://arxiv.org/abs/1706.01307>.
- [6] S. Gupta, P. Arbelaez, and J. Malik. Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013.
- [7] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [8] K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. *European Conference on Computer Vision*, 2016.
- [9] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely Connected Convolutional Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [10] R. Klokov and V. Lempitsky. Escape from Cells: Deep Kd-Networks for The Recognition of 3D Point Cloud Models. *arXiv preprint arXiv:1704.01222*, 2017.
- [11] X. G. L. Yi, H. Su and L. Guibas. SyncSpecCNN: Synchronized Spectral CNN for 3D Shape Segmentation. *arXiv preprint arXiv:1612.00606*, 2016.
- [12] Y. LeCun, J. Denker, and S. Solla. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*, 1990.
- [13] B. Liu, M. Wang, H. Foroosh, M. Tappen, , and M. Penksy. Sparse Convolutional Neural Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
- [14] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
- [15] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. *IEEE International Conference on Intelligent Robots and Systems*, 2015.
- [16] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor Segmentation and Support Inference from RGBD Images. *European Conference on Computer Vision*, 2012.
- [17] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv preprint arXiv:1612.00593*, 2016.
- [18] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning Deep 3D Representations at High Resolutions. *arXiv preprint arXiv:1611.05009*, 2016.
- [19] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015.
- [20] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [21] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-View Convolutional Neural Networks for 3D Shape Recognition. *International Conference on Computer Vision*, 2015.
- [22] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. *Robotics: Science and Systems*, 2015.
- [23] L. Yi, H. Su, L. Shao, M. Savva, et al. Large-Scale 3D Shape Reconstruction and Segmentation from ShapeNet Core55. *arXiv preprint arXiv:1710.06104*, 2017.
- [24] F. Yu and V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv preprint arXiv:1511.07122*, 2015.
- [25] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.