Tangent Convolutions for Dense Prediction in 3D

Maxim Tatarchenko* University of Freiburg
Jaesik Park* Intel Labs
Vladlen Koltun Intel Labs
Qian-Yi Zhou Intel Labs

Abstract

We present an approach to semantic scene analysis using deep convolutional networks. Our approach is based on tangent convolutions – a new construction for convolutional networks on 3D data. In contrast to volumetric approaches, our method operates directly on surface geometry. Crucially, the construction is applicable to unstructured point clouds and other noisy real-world data. We show that tangent convolutions can be evaluated efficiently on large-scale point clouds with millions of points. Using tangent convolutions, we design a deep fully-convolutional network for semantic segmentation of 3D point clouds, and apply it to challenging real-world datasets of indoor and outdoor 3D environments. Experimental results show that the presented approach outperforms other recent deep network constructions in detailed analysis of large 3D scenes.

1. Introduction

Methods that utilize convolutional networks on 2D images dominate modern computer vision. A key contributing factor to their success is efficient local processing based on the convolution operation. 2D convolution is defined on a regular grid, a domain that supports extremely efficient implementation. This in turn enables using powerful deep architectures for processing large datasets at high resolution.

When it comes to analysis of large-scale 3D scenes, a straightforward extension of this idea is volumetric convolution on a voxel grid [35, 59, 10]. However, voxel-based methods have limitations, including a cubic growth rate of memory consumption and computation time. For this reason, voxel-based ConvNets operate on low-resolution voxel grids that limit their prediction accuracy. The problem can be alleviated by octree-based techniques that define a ConvNet on an octree and enable processing somewhat higher-resolution volumes (e.g., up to 256^3 voxels) [43, 57, 18, 42, 51]. Yet even this may be insufficient for detailed analysis of large-scale scenes.

On a deeper level, both efficient and inefficient voxel-based methods treat 3D data as *volumetric* by exploiting 3D convolutions that integrate over volumes. In reality, data captured by 3D sensors such as RGB-D cameras and LiDAR typically represent *surfaces*: 2D structures embedded in 3D space. (This is in contrast to truly volumetric 3D data, as encountered for example in medical imaging.) Classic features that are used for the analysis of such data are defined in terms that acknowledge the latent surface structure, and do not treat the data as a volume [20, 12, 45].

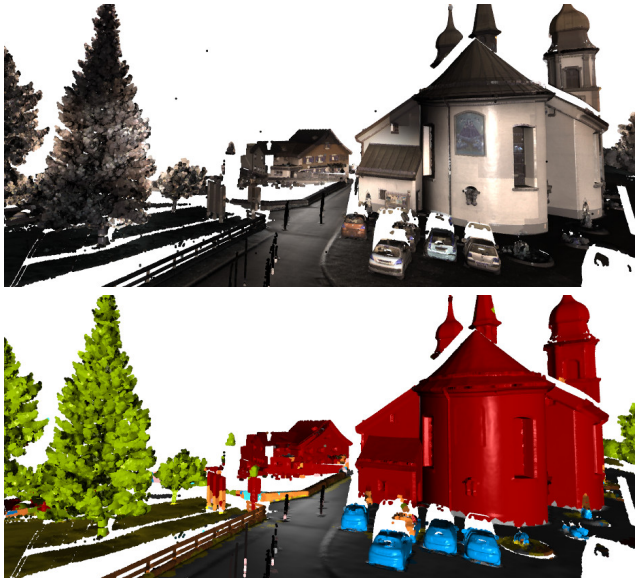


Figure 1. Convolutional networks based on tangent convolutions can be applied to semantic analysis of large-scale scenes, such as urban environments. Top: point cloud from the Semantic3D dataset. Bottom: semantic segmentation produced by the presented approach.

The drawbacks of voxel-based methods are known in the research community. A number of recent works argue that volumetric data structures are not the natural substrate for 3D ConvNets, and propose alternative designs based on unordered point sets [39], graphs [47], and sphere-type surfaces [32]. Unfortunately, these methods come with their own drawbacks, such as limited sensitivity to local structure or restrictive topological assumptions.

We develop an alternative construction for convolutional networks on surfaces, based on the notion of *tangent con-*

*Equal contribution.

volution. This construction assumes that the data is sampled from locally Euclidean surfaces. The latent surfaces need not be known, and the data can be in any form that supports approximate normal vector estimation, including point clouds, meshes, and even polygon soup. (The same assumption concerning normal vector estimation is made by both classic and contemporary geometric feature descriptors [20, 12, 45, 53, 46, 23].) The tangent convolution is based on projecting local surface geometry on a tangent plane around every point. This yields a set of tangent images. Every tangent image is treated as a regular 2D grid that supports planar convolution. The content of all tangent images can be precomputed from the surface geometry, which enables efficient implementation that scales to large datasets, such as urban environments.

Using tangent convolution as the main building block, we design a U-type network for dense semantic segmentation of point clouds. Our proposed architecture is general and can be applied to analysis of large-scale scenes. We demonstrate its performance on three diverse real-world datasets containing indoor and outdoor environments. A semantic segmentation produced by a tangent convolutional network is shown in Figure 1.

2. Related Work

Dense prediction in 3D, including semantic point cloud segmentation, has a long history in computer vision. Pioneering methods work on aerial LiDAR data and are based on hand-crafted features with complex classifiers on top [6, 7, 15]. Such approaches can also be combined with high-level architectural rules [33]. A popular line of work exploits graphical models, including conditional random fields [38, 11, 2, 58, 19, 27, 56]. Related formulations have also been proposed for interactive 3D segmentation [55, 37].

More recently, the deep learning revolution in computer vision has spread to consume 3D data analysis. A variety of methods that tackle 3D data using deep learning techniques have been proposed. They can be considered in terms of the underlying data representation.

A common representation of 3D data for deep learning is a voxel grid. Deep networks that operate on voxelized data have been applied to shape classification [35, 59, 40], semantic segmentation of indoor scenes [10], and biomedical recordings [9, 8]. Due to the cubic complexity of voxel grids, these methods can only operate at low resolution – typically not more than 64^3 – and have limited accuracy. Attempting to overcome this limitation, researchers have proposed representations based on hierarchical spatial data structures such as octrees and kd-trees [43, 57, 18, 42, 13, 25, 51], which are more memory- and computation-efficient, and can therefore handle higher resolutions. An alternative way of increasing the accuracy of voxel-based techniques is to add differentiable post-processing, modeled upon the dense CRF [26, 52].

Other applications of deep networks consider RGB-D images, which can be treated with fully-convolutional networks [16, 29, 36] and graph neural networks [41]. These approaches support the use of powerful pretrained 2D networks, but are not generally applicable to unstructured point clouds with unknown sensor poses. Attempting to address this issue, Boulch et al. [5] train a ConvNet on images rendered from point clouds using randomly placed virtual cameras. In a more controlled setting with fixed camera poses, multi-view methods are successfully used for shape segmentation [21], shape recognition [49, 40], and shape synthesis [14, 30, 22, 50]. Our approach can be viewed as an extreme multi-view approach in which a virtual camera is associated with each point in the point cloud. A critical problem that we address is the efficient and scalable implementation of this approach, which enables its application to dense point clouds of large-scale indoor and outdoor environments.

Qi et al. [39] propose a network for analysing unordered point sets, which is based on independent point processing combined with global context aggregation through max-pooling. Since the communication between the points is quite weak, this approach experiences difficulties when applied to large-scale scenes with complex layouts.

There is a variety of more exotic deep learning formulations for 3D analysis that do not address large-scale semantic segmentation of whole scenes but provide interesting ideas. Yi et al. [60] consider shape segmentation in the spectral domain by synchronizing eigenvectors across models. Masci et al. [34] and Boscaini et al. [4] design ConvNets for Riemannian manifolds and use them to learn shape correspondences. Sinha et al. [48] perform shape analysis on geometry images. Simonovsky et al. [47] extend the convolution operator from regular grids to arbitrary graphs and use it to design shape classification networks. Li et al. [28] introduce Field Probing Neural Networks which respect the underlying sparsity of 3D data and are used for efficient feature extraction. Tulsiani et al. [54] approximate 3D models with volumetric primitives in an end-to-end differentiable framework, and use this representation for solving several tasks. Maron et al. [32] design ConvNets on surfaces for sphere-type shapes.

Overall, most existing 3D deep learning systems either rely on representations that do not support general scene analysis, or have poor scalability. As we will show, deep networks based on tangent convolutions scale to millions of points and are suitable for detailed analysis of large scenes.

3. Tangent Convolution

In this section we formally introduce tangent convolutions. All derivations are provided for point clouds, but they

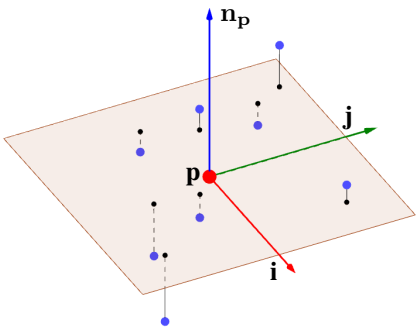


Figure 2. Points $\mathbf{q}$ (blue) from the local neighborhood of a point $\mathbf{p}$ (red) are projected onto the tangent image.

can easily be applied to any type of 3D data that supports surface normal estimation, such as meshes.

Convolution with a continuous kernel. Let $\mathcal{P} = \{\mathbf{p}\}$ be a point cloud, and let $F(\mathbf{p})$ be a discrete scalar function that represents a signal defined over $\mathcal{P}$. $F(\mathbf{p})$ can encode color, geometry, or abstract features from intermediate network layers. In order to convolve F, we need to extend it to a continuous function. Conceptually, we introduce a virtual orthogonal camera for $\mathbf{p}$. It is configured to observe $\mathbf{p}$ along the normal $\mathbf{n_p}$. The image plane of this virtual camera is the tangent plane $\pi_\mathbf{p}$ of $\mathbf{p}$. It parameterizes a virtual image that can be represented as a continuous signal $S(\mathbf{u})$, where $\mathbf{u} \in \mathbb{R}^2$ is a point in $\pi_\mathbf{p}$. We call S a tangent image.

The tangent convolution at $\mathbf{p}$ is defined as

$$X(\mathbf{p}) = \int_{\pi_\mathbf{p}} c(\mathbf{u}) S(\mathbf{u})\, d\mathbf{u}, \tag{1}$$

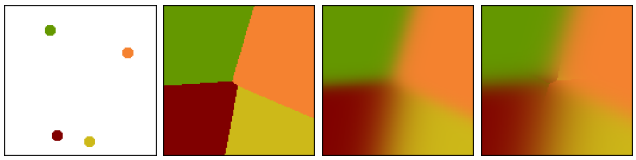
where $c(\mathbf{u})$ is the convolution kernel. We now describe how S is computed from F.

Tangent plane estimation. For each point $\mathbf{p}$ we estimate the orientation of its camera image using local covariance analysis. This is a standard procedure [46] but we summarize it here for completeness. Consider a set of points $\mathbf{q}$ from a spherical neighborhood of $\mathbf{p}$, such that $\|\mathbf{p}-\mathbf{q}\| < R$. The orientation of the tangent plane is determined by the eigenvectors of the covariance matrix $\mathbf{C} = \sum_\mathbf{q} \mathbf{r}\mathbf{r}^\top$, where $\mathbf{r} = \mathbf{q} - \mathbf{p}$. The eigenvector of the smallest eigenvalue defines the estimated surface normal $\mathbf{n_p}$, and the other two eigenvectors $\mathbf{i}$ and $\mathbf{j}$ define the 2D image axes that parameterize the tangent image.

Signal interpolation. Now our goal is to estimate image signals $S(\mathbf{u})$ from point signals $F(\mathbf{q})$. We begin by projecting the neighbors $\mathbf{q}$ of $\mathbf{p}$ onto the tangent image, which yields a set of projected points $\mathbf{v} = (\mathbf{r}^\top\mathbf{i}, \mathbf{r}^\top\mathbf{j})$. This is illustrated in Figure 2. We define

$$S(\mathbf{v}) = F(\mathbf{q}). \tag{2}$$

As shown in Figure 2 and Figure 3(a), points $\mathbf{v}$ are scattered



(a) (b) (c) (d)

Figure 3. Signals from projected points (a) can be interpolated using one of the following schemes: nearest neighbor (b), full Gaussian mixture (c), and Gaussian mixture with top-3 neighbors (d).

on the image plane. We thus need to interpolate their signals in order to estimate the full function $S(\mathbf{u})$ over the tangent image:

$$S(\mathbf{u}) = \sum_\mathbf{v} \big(w(\mathbf{u},\mathbf{v}) \cdot S(\mathbf{v})\big), \tag{3}$$

where $w(\mathbf{u},\mathbf{v})$ is a kernel weight that satisfies $\sum_\mathbf{v} w = 1$. We consider two schemes for signal interpolation: nearest neighbor and Gaussian kernel mixture. These schemes are illustrated in Figure 3. In the nearest neighbor (NN) case,

$$w(\mathbf{u},\mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{v} \text{ is } \mathbf{u}\text{'s NN,} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

In the Gaussian kernel mixture case,

$$w(\mathbf{u},\mathbf{v}) = \frac{1}{A} \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{\sigma^2}\right), \tag{5}$$

where A normalizes the weights such that $\sum_\mathbf{v} w = 1$. More sophisticated signal interpolation schemes can be considered, but we have not observed a significant effect of the interpolation scheme on empirical performance and will mostly use simple nearest-neighbor estimation.

Finally, if we rewrite Equation (1) using the definitions from Equations (2) and (3), we get the formula for the tangent convolution:

$$X(\mathbf{p}) = \int_{\pi_\mathbf{p}} c(\mathbf{u}) \cdot \sum_\mathbf{v} \big(w(\mathbf{u},\mathbf{v}) \cdot F(\mathbf{q})\big)\, d\mathbf{u}. \tag{6}$$

Note that the role of the tangent image is increasingly implicit: it provides the domain for $\mathbf{u}$ and figures in the evaluation of the weights w, but otherwise it need not be explicitly maintained. We will build on this observation in the next section to show that tangent convolutions can be evaluated efficiently at scale, and can support the construction of deep networks on point clouds with millions of points.

4. Efficiency

In this section we describe how the tangent convolution defined in Section 3 can be computed efficiently. In practice, the tangent image is treated as a discrete function on a

regular $l \times l$ grid. Elements $\mathbf{u}$ are pixels in this virtual image. The convolution kernel c is a discrete kernel applied onto this image. Let us first consider the nearest-neighbor signal interpolation scheme introduced in Equation (4). We can rewrite Equation (6) as

$$X(\mathbf{p}) = \sum_{\mathbf{u}} \Big(c(\mathbf{u}) \cdot F\big(g(\mathbf{u})\big) \Big), \tag{7}$$

where $g(\mathbf{u})$ is a selection function that returns a point which projects to the nearest neighbor of $\mathbf{u}$ on the image plane. Note that g only depends on the point cloud geometry and does not depend on the signal F. This allows us to precompute g for all points.

From here on, we employ standard ConvNet terminology and proceed to show how to implement a convolutional layer using tangent convolutions. Our goal is to convolve an input feature map $\mathbf{F}_{in}$ of size $N \times C_{in}$ with a set of weights W to produce an output feature map $\mathbf{F}_{out}$ of size $N \times C_{out}$, where N is the number of points in the point cloud, while C_{in} and C_{out} denote the number of input and output channels respectively. For implementation, we unroll 2D tangent images and convolutional filters of size $l \times l$ into 1D vectors of size $1 \times L$, where $L = l^2$. From then on, we compute 1D convolutions. Note that such representation of a 2D tangent convolution as a 1D convolution is not an approximation: the results of the two operations are identical.

We start by precomputing the function g, which is represented as an $N \times L$ index matrix $\mathbf{I}$. Elements of $\mathbf{I}$ are indices of the corresponding tangent-plane nearest-neighbors in the point cloud. Using $\mathbf{I}$, we gather input signals (features) into an intermediate tensor $\mathbf{M}$ of size $N \times L \times C_{in}$. This tensor is convolved with a flattened set of kernels W of size $1 \times L$, which yields the output feature map $\mathbf{F}_{out}$. This process is illustrated in Figure 4.

Consider now the case of signal interpolation using Gaussian kernel mixtures. For efficiency, we only consider the set of top-k neighbors for each point, denoted NN_k. An example image produced using the Gaussian kernel mixture scheme with top-3 neighbors is shown in Figure 3(d). Equation (5) turns into

$$w(\mathbf{u}, \mathbf{v}) = \begin{cases} \frac{1}{A} \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{\sigma^2} \right) & \text{if } \mathbf{v} \in NN_k \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where A normalizes weights such that $\sum_{\mathbf{v}} w = 1$. With this approximation, each pixel $\mathbf{u}$ has at most k non-zero weights, denoted by $w_{1..k}(\mathbf{u})$. Their corresponding selection functions are denoted by $g_{1..k}(\mathbf{u})$. Both the weights and the selection functions are independent of the signal F,

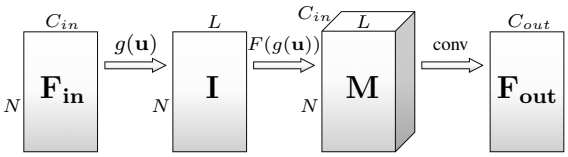


Figure 4. Efficient evaluation of a convolutional layer built on tangent convolutions.

and are thus precomputed. Equation (6) becomes

$$X(\mathbf{p}) = \sum_{\mathbf{u}} \Big(c(\mathbf{u}) \cdot \sum_{i=1}^{k} \big(w_i(\mathbf{u}) \cdot F(g_i(\mathbf{u})) \big) \Big) \tag{9}$$

$$= \sum_{i=1}^{k} \sum_{\mathbf{u}} \Big(w_i(\mathbf{u}) \cdot c(\mathbf{u}) \cdot F(g_i(\mathbf{u})) \Big). \tag{10}$$

As with the nearest-neighbor signal interpolation scheme, we represent the precomputed selection functions g_i as k index matrices $\mathbf{I}_i$ of size $N \times L$. These index matrices are used to assemble k intermediate signal tensors $\mathbf{M}_i$ of size $N \times L \times C_{in}$. Additionally, we collate the precomputed weights into k weight matrices $\mathbf{H}_i$ of size $N \times L$. They are used to compute the weighted sum $\mathbf{M} = \sum_i \mathbf{H}_i \odot \mathbf{M}_i$, which is finally convolved with the kernel W.

We implemented the presented construction in TensorFlow [1]. It consists entirely of differentiable atomic operations, thus backpropagation is done seamlessly using the automatic differentiation functionality of the framework.

5. Additional Ingredients

In this section we introduce additional ingredients that are required to construct a convolutional network for point cloud analysis.

5.1. Multi-scale analysis

Pooling. Convolutional networks commonly use pooling to aggregate signals over larger spatial regions. We implement pooling in our framework via hashing onto a regular 3D grid. Points that are hashed onto the same grid point pool their signals. The spacing of the grid determines the pooling resolution. Consider points $\mathcal{P} = \{\mathbf{p}\}$ and corresponding signal values $\{F(\mathbf{p})\}$. Let $\mathbf{g}$ be a grid point and let $\mathcal{V}_\mathbf{g}$ be the set of points in $\mathcal{P}$ that hash to $\mathbf{g}$. (The hash function can be assumed to be simple quantization onto the grid in each dimension.) Assume that $\mathcal{V}_\mathbf{g}$ is not empty and consider average pooling. All points in $\mathcal{V}_\mathbf{g}$ and their signals are pooled onto a single point:

$$\mathbf{p}'_\mathbf{g} = \frac{1}{|\mathcal{V}_\mathbf{g}|} \sum_{\mathbf{p} \in \mathcal{V}_\mathbf{g}} \mathbf{p} \quad \text{and} \quad F'(\mathbf{p}'_\mathbf{g}) = \frac{1}{|\mathcal{V}_\mathbf{g}|} \sum_{\mathbf{p} \in \mathcal{V}_\mathbf{g}} F(\mathbf{p}). \tag{11}$$

In a convolutional network based on tangent convolutions, we pool using progressively coarser grids. Starting with some initial grid resolution (5cm in each dimension, say), each successive pooling layer increases the step of the grid by a factor of two (to 10cm, then 20cm, etc.). Such hashing also alleviates the problem of non-uniform point density. As a result, we can select the neighborhood radius for the convolution operation globally for the entire dataset.

After each pooling layer, the radius r that is used to estimate the tangent plane and the pixel size of the virtual tangent image are doubled accordingly. Thus the resolution of all tangent images decreases in step with the resolution of the point cloud. Note that the downsampled point clouds produced by pooling layers are independent of the signals defined over them. The downsampled point clouds, the associated tangent planes, and the corresponding index and weight functions can thus all be precomputed for all layers in the convolutional network: they need only be computed once per pooling layer.

The implementation of a pooling layer is similar in spirit to that of a convolutional layer described in Section 4. Consider an input feature map $\mathbf{F}_{in}$ of size $N_{in} \times C$. Using grid hashing, we assemble an index matrix $\mathbf{I}$ of size $N_{out} \times 8$, which contains indices of points that hash to the same grid point. Assuming that we decrease the grid resolution by a factor of 2 in each dimension in each pooling layer, the number of points that hash to the same grid point will be at most 8 in general. (For initialization, we quantize the points to some base resolution.) Using $\mathbf{I}$, we assemble an intermediate tensor of size $N_{out} \times 8 \times C$. We pool this tensor along the second dimension according to the pooling operator (max, average, etc.), and thus obtain an output feature map $\mathbf{F}_{out}$ of size $N_{out} \times C$.

Note that all stages in this process have linear complexity in the number of points. Although points are hashed onto regular grids, the grids themselves are never constructed or represented. Hashing is performed via modular arithmetic on individual point coordinates, and all data structures have linear complexity in the number of points, independent of the extent of the point set or the resolution of the grid.

Unpooling. The unpooling operation has an opposite effect to pooling: it distributes signals from points in a low-resolution feature map $\mathbf{F}_{in}$ onto points in a higher-resolution feature map $\mathbf{F}_{out}$. Unpooling reuses the index matrix from the corresponding pooling operation. We copy features from a single point in a low-resolution point cloud to multiple points from which the information was aggregated during pooling.

5.2. Local distance feature

So far, we have considered signals that could be expressed in terms of a scalar function $F(\mathbf{q})$ with a well-defined value for each point $\mathbf{q}$. This holds for color, intensity, and abstract ConvNet features. There is, however, a signal that cannot be expressed in such terms and needs special treatment. This signal is distance to the tangent plane $\pi_{\mathbf{p}}$. This local signal is calculated by taking the distance from each neighbor $\mathbf{q}$ to the tangent plane of $\mathbf{p}$: $d = (\mathbf{q} - \mathbf{p})^\top \mathbf{n}_{\mathbf{p}}$.

This signal is defined in relation to the point $\mathbf{p}$, therefore it cannot be directly plugged into the pipeline shown in Figure 4. Instead, we precompute the distance images for every point. Scattered signal interpolation is done in the same way as for scalar signals (Equation (3)). After assembling the intermediate tensor $\mathbf{M}$ for the first convolutional layer, we simply concatenate these distance images as an additional channel in $\mathbf{M}$. The first convolutional layer generates a set of abstract features $\mathbf{F}_{out}$ that can be treated as scalar signals from here on.

All precomputations are implemented using Open3D [61].

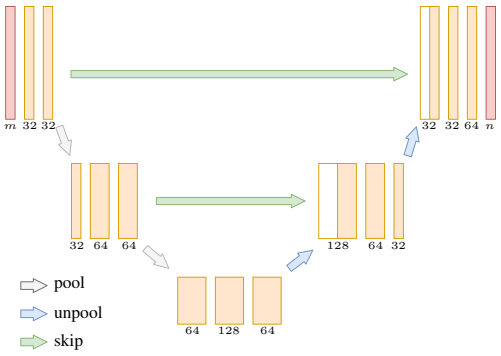


Figure 5. We use a fully-convolutional U-shaped network with skip connections. The network receives m-dimensional features as input and produces prediction scores for n classes.

6. Architecture

Using the ingredients introduced in the previous sections, we design an encoder-decoder network inspired by the U-net [44]. The network architecture is illustrated in Figure 5. It is a fully-convolutional network over a point cloud, where the convolutions are tangent convolutions. The encoder contains two pooling layers. The decoder contains two corresponding unpooling layers. Encoder features are propagated to corresponding decoder blocks via skip-connections. All layers except the last one use 3×3 filters and are followed by Leaky ReLU with negative slope 0.2 [31]. The last layer uses 1×1 convolutions to produce final class predictions. The network is trained by optimizing the cross-entropy objective using the Adam optimizer with initial learning rate 10^{-4} [24].

Receptive field. The receptive field size of one convolutional layer is determined by the pixel size r of the tangent image and the radius R that is used to collect the neighbors

of each point $\mathbf{p}$. We set $R = 2r$, therefore the receptive field size of one layer is R. After each pooling layer, r is doubled. The receptive field of an element in the network can be calculated by tracing the receptive fields of preceding layers. With initial $r = 5$cm, the receptive field size of elements in the final layer of the presented architecture is $4 \cdot 10 + 4 \cdot 20 + 2 \cdot 40 = 200$cm.

7. Experiments

We evaluate the performance of the presented approach on the task of semantic 3D scene segmentation. Our approach is compared to recent deep networks for 3D data on three different datasets.

7.1. Datasets and measures

We conduct experiments on three large-scale datasets that contain real-world 3D scans of indoor and outdoor environments.

Semantic3D [17] is a dataset of scanned outdoor scenes with over 3 billion points. It contains 15 training and 15 test scenes annotated with 8 class labels. Being unable to evaluate the baseline results on the official test server, we use our own train/test split: Bildstein 1-3-5 are used for testing, the rest for training.

Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [3] contains 6 large-scale indoor areas from 3 different buildings, with 13 object classes. We use Area 5 for testing and the rest for training.

ScanNet [10] is a dataset with more than 1,500 scans of indoor scenes with 20 object classes collected using an RGB-D capture system. We follow the standard train/test split provided by the authors.

Measures. We report three measures: mean accuracy over classes (mA), mean intersection over union (mIoU), and overall accuracy (oA). We build a full confusion matrix based on the entire test set, and derive the final scores from it. Measures are evaluated over the original point clouds. For approaches that produce labels over downsampled or voxelized representations, we map these predictions to the original point clouds via nearest-neighbor assignment.

Although we report oA for completeness, it is not a good measure for semantic segmentation. If there are dominant classes in the data (e.g., walls, floor, and ceiling in indoor scenes), making correct predictions for these but poor predictions over the other classes will yield misleadingly high oA scores.

7.2. Baselines

We compare our approach to three recent deep learning methods that operate on different underlying representations. We have chosen reasonably general methods that have the potential to be applied to general scene analysis and have open-source implementations. Our baselines are PointNet [39], which operates on points, ScanNet [10], which operates on low-resolution voxel grids, and OctNet [43], which operates on higher-resolution octrees. We used the source code provided by the authors. Due to the design of these methods, the data preparation routines and the input signals are different for each dataset, and sometimes deviate from the guidelines provided in the papers.

PointNet. For indoor datasets, we used the data sampling strategy suggested in the original paper with global xyz, locally normalized xyz, and RGB as inputs. For Semantic3D, we observed global xyz to be harmful, thus we only use local xyz and color. Training data is generated by randomly sampling $(3\text{m})^3$ cubes from the training scenes. Evaluation is performed by applying a sliding window over the entire scan.

ScanNet. The original network used 2 input channels: occupancy and a visibility mask computed using known camera trajectories. Since scenes in general are not accompanied by known camera trajectories, we only use occupancy in the input signal. Following the original setup, we use $1.5\times1.5\times3$m volumes voxelized into $31\times31\times62$ grids and augmented by 8 rotations. Each such cube yields a prediction for one $1\times1\times62$ column. (I.e., the ScanNet network outputs a prediction for the central column only.) We use random sampling for training, and exhaustive sliding window for testing.

OctNet. We use an architecture that operates on 256^3 octrees. Inputs to the network are color, occupancy, and a height-based feature that assigns each point to the top or bottom part of the scan. Based on correspondence with the authors regarding the best way to set up OctNet on different datasets, we used $(45\text{m})^3$ volumes for Semantic3D and $(11\text{m})^3$ volumes for the indoor datasets.

7.3. Setup of the presented approach

The architecture described in Section 6 is used in all experiments. We evaluate four variants that use different input signals: distance from tangent plane (D), height above ground (H), normals (N), and color (RGB). All input signals are normalized between 0 and 1. The initial resolution r of the tangent image is 5cm for the indoor datasets and 10cm for Semantic3D. It is doubled after each pooling layer. In addition to providing the distance from tangent plane as input to the first convolutional layer, we concatenate the local distance features from all scales of the point cloud to the feature maps of the corresponding resolution produced by pooling layers.

For ScanNet and S3DIS, we used whole rooms as individual training batches. For Semantic3D, each batch was a random sphere with a radius of 6m. For indoor scans,

we augment each scan by 8 rotations around the vertical axis. To correct for imbalance between different classes, we weigh the loss with the negative log of the training data histogram.

7.4. Signal interpolation

We begin by comparing the effectiveness of two different signal interpolation schemes: nearest neighbor and Gaussian mixture. Both networks were trained on S3DIS with D and H as the input signals. The resulting segmentation scores are provided in the supplement. The two networks produce similar results. We conclude that the nearest neighbor signal estimation scheme is sufficient, and use it in all other experiments.

7.5. Main results

Quantitative results for all methods are summarized in Table 2. Overall, our method produces high scores on all datasets and consistently outperforms the baselines. Qualitative comparisons are shown in Figure 6.

Comparing the configurations of our networks that use different input signals, we can see that geometry is much more important than color on the indoor datasets. Adding RGB information only slightly improves the scores on S3DIS and is actually harmful for mean and overall accuracy on the ScanNet dataset. The situation is different for the Semantic3D dataset: the network trained with color significantly outperforms all other configurations. Due to the fact that H is normalized between 0 and 1 for every scan separately, this information turns out to be harmful when the global height of different scans is significantly different. Therefore, the network trained only with the distance signal performs better than the other two geometric configurations.

In setting up and operating the baseline methods, we found that all of them are quite hard to apply across datasets: some non-trivial decisions had to be made for each new dataset during the data preparation stage. None of the baselines showed consistent performance across the different types of scenes.

PointNet reaches high oA scores on both indoor datasets. However, the oA measure is strongly dominated by large classes such as walls, floor, and ceiling. S3DIS has a fairly regular layout because of the global room alignment procedure, which is very beneficial for PointNet and allows it to reach reasonable mA and mIoU scores on this dataset. However, PointNet performs poorly on the ScanNet dataset, which has more classes and noisy data. All but the most prominent classes (i.e., walls and floor) are misclassified. PointNet completely fails to produce meaningful predictions on the even more challenging Semantic3D dataset.

Our configuration of the ScanNet method produces reasonable oA scores on both indoor datasets, but does much worse in the other two measures. For reference, on the ScanNet dataset we additionally report the number from the original paper where a binary visibility-from-camera mask was used as an additional input channel. This number is much higher than our occupancy-only results, which do not assume a known camera trajectory. Due to the fact that the network only outputs predictions for the central column of the voxel grid, evaluation is extremely time-consuming for the large scenes in the Semantic3D dataset. Because of this scalability issue, we did not succeed in evaluating ScanNet on this dataset.

OctNet reaches good performance on the Semantic3D dataset. However, the same network configuration yields bad results when applied to the indoor datasets. A possible explanation for this may be poor generalization due to overfitting to the structure of training octrees.

7.6. Efficiency

We compared the efficiency of different methods on a scan from S3DIS containing 125K points after grid hashing. The results are reported in Table 1. Since ScanNet and PointNet require multiple iterations for labeling a single scan, we report both the time of a single forward pass and the time for processing a full scan. OctNet and our method process a full scan in one forward pass, which also explains their higher memory consumption compared to ScanNet and PointNet. ScanNet does not provide code for data preprocessing, so we report the runtime of our Python implementation needed for generating 38K sliding windows during inference. Our method exhibits the best runtime for both precomputation and inference.

	Prep (s)	FP (s)	Full (s)	Mem (GB)
PointNet	16.5	0.01	0.65	0.39
OctNet	15.5	0.61	0.61	3.33
ScanNet	867.8	0.002	6.34	0.97
Ours	1.59	0.52	0.52	2.35

Table 1. Efficiency of different methods. We report preprocessing time (Prep), time for a single forward pass (FP), time for processing a full scan (Full), and memory consumption (Mem).

8. Conclusion

We have presented tangent convolutions – a new construction for convolutional networks on 3D data. The key idea is to evaluate convolutions on virtual tangent planes at every point. Crucially, tangent planes can be precomputed and deep convolutional networks based on tangent convolutions can be evaluated efficiently on large point clouds. We have applied tangent convolutions to semantic segmentation of large indoor and outdoor scenes. The presented ideas may also be applicable to other problems in analysis, processing, and synthesis of 3D data.

	Semantic3D [17]			ScanNet [10]			S3DIS [3]		
	mIoU	mA	oA	mIoU	mA	oA	mIoU	mA	oA
PointNet [39]	3.76	16.9	16.3	12.2	17.9	68.1	41.3	49.5	78.8
OctNet [43]	50.7	71.3	80.7	18.1	26.4	76.6	26.3	39.0	68.9
ScanNet [10]	n/a	n/a	n/a	13.5	19.2 (*50.8*)	69.4 (*73.0*)	24.6	35.0	64.2
Ours (D)	58.1	78.9	84.8	40.9	52.5	**80.9**	49.8	60.3	80.2
Ours (DH)	58.0	75.8	83.3	40.3	52.2	80.6	50.0	60.0	81.2
Ours (DHN)	52.5	79.3	79.5	40.7	**55.3**	80.3	51.7	61.0	82.2
Ours (DHNRGB)	**66.4**	**80.7**	**89.3**	**40.9**	55.1	80.1	**52.8**	**62.2**	**82.5**

Table 2. Semantic segmentation accuracy for all methods across the three datasets. We report mean intersection over union (mIoU), mean class accuracy (mA), and overall accuracy (oA). Note that oA is a bad measure and we recommend against using it in the future. We tested different configurations of our method by combining four types of input signals: depth (D), height (H), normals (N), and color (RGB).

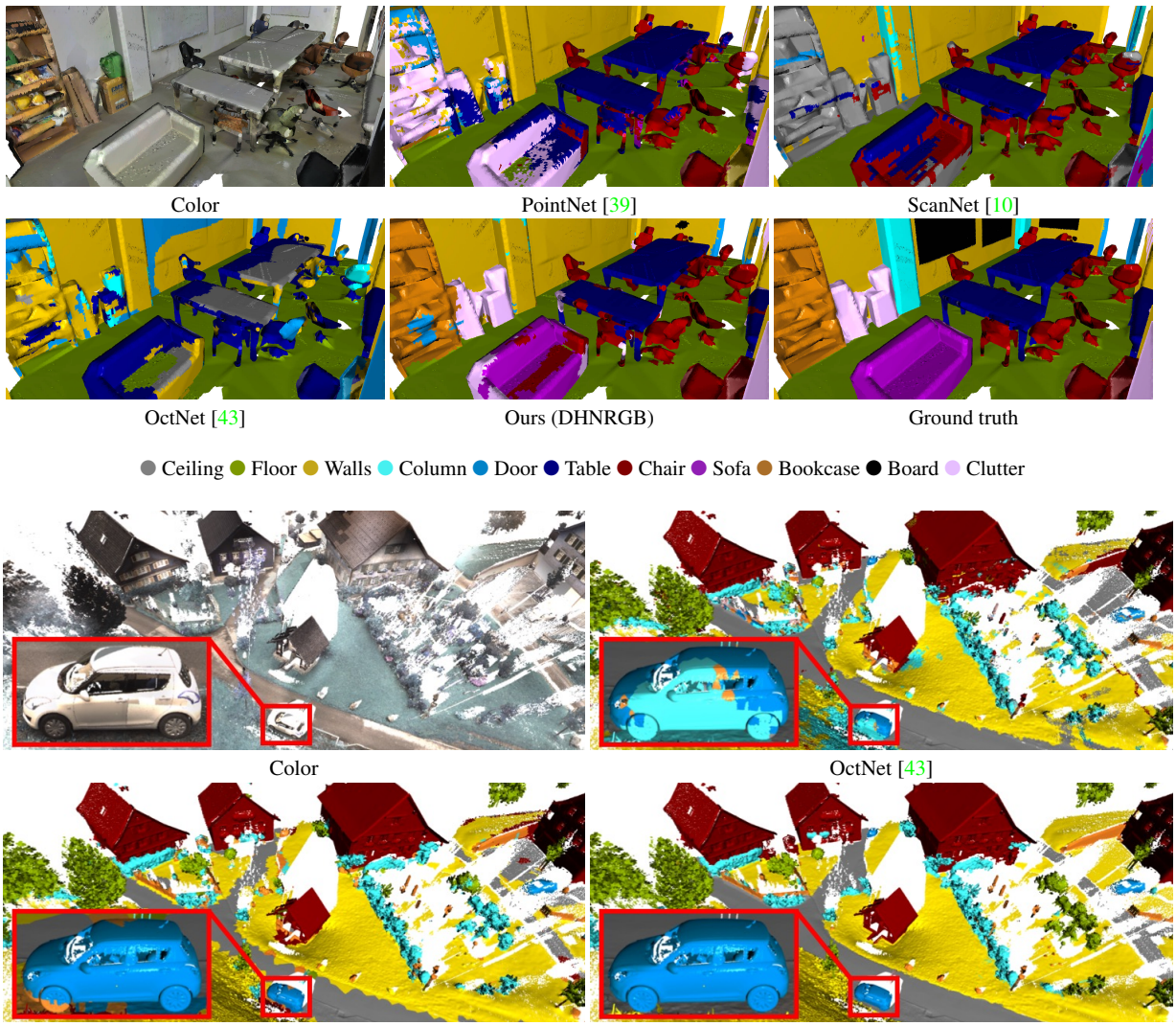


Figure 6. Qualitative comparisons on S3DIS [3] (top) and Semantic3D [17] (bottom). Labels are coded by color.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.

[2] A. Anand, H. S. Koppula, T. Joachims, and A. Saxena. Contextually guided semantic labeling and search for three-dimensional point clouds. *International Journal of Robotics Research*, 32(1), 2013.

[3] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3D semantic parsing of large-scale indoor spaces. In *CVPR*, 2016.

[4] D. Boscaini, J. Masci, E. Rodolà, and M. M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *NIPS*, 2016.

[5] A. Boulch, B. L. Saux, and N. Audebert. Unstructured point cloud semantic labeling using deep segmentation networks. In *Eurographics Workshop on 3D Object Retrieval*, 2017.

[6] A. P. Charaniya, R. Manduchi, and S. K. Lodha. Supervised parametric classification of aerial LiDAR data. In *CVPR Workshops*, 2004.

[7] N. Chehata, L. Guo, and C. Mallet. Contribution of airborne full-waveform lidar and image data for urban scene classification. In *ICIP*, 2009.

[8] H. Chen, Q. Dou, L. Yu, and P. Heng. VoxResNet: Deep voxelwise residual networks for volumetric brain segmentation. *arXiv:1608.05895*, 2016.

[9] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3D U-Net: Learning dense volumetric segmentation from sparse annotation. In *MICCAI*, 2016.

[10] A. Dai, A. X. Chang, M. Savva, M. Halber, T. A. Funkhouser, and M. Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *CVPR*, 2017.

[11] G. Floros and B. Leibe. Joint 2D-3D temporally consistent semantic segmentation of street scenes. In *CVPR*, 2012.

[12] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik. Recognizing objects in range data using regional point descriptors. In *ECCV*, 2004.

[13] M. Gadelha, S. Maji, and R. Wang. 3D shape generation using spatially ordered point clouds. In *BMVC*, 2017.

[14] M. Gadelha, S. Maji, and R. Wang. 3D shape induction from 2D views of multiple objects. In *3DV*, 2017.

[15] A. Golovinskiy, V. G. Kim, and T. A. Funkhouser. Shape-based recognition of 3D point clouds in urban environments. In *ICCV*, 2009.

[16] S. Gupta, P. A. Arbeláez, R. B. Girshick, and J. Malik. Indoor scene understanding with RGB-D images: Bottom-up segmentation, object detection and semantic segmentation. *IJCV*, 112(2), 2015.

[17] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys. Semantic3D.net: A new large-scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017.

[18] C. Häne, S. Tulsiani, and J. Malik. Hierarchical surface prediction for 3D object reconstruction. In *3DV*, 2017.

[19] A. Hermans, G. Floros, and B. Leibe. Dense 3D semantic mapping of indoor scenes from RGB-D images. In *ICRA*, 2014.

[20] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *PAMI*, 21(5), 1999.

[21] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri. 3D shape segmentation with projective convolutional networks. In *CVPR*, 2017.

[22] A. Kar, C. Häne, and J. Malik. Learning a multi-view stereo machine. In *NIPS*, 2017.

[23] M. Khoury, Q.-Y. Zhou, and V. Koltun. Learning compact geometric features. In *ICCV*, 2017.

[24] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[25] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3D point cloud models. In *ICCV*, 2017.

[26] P. Krähenbühl and V. Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *NIPS*, 2011.

[27] A. Kundu, Y. Li, F. Dellaert, F. Li, and J. M. Rehg. Joint semantic segmentation and 3D reconstruction from monocular video. In *ECCV*, 2014.

[28] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. FPNN: Field probing neural networks for 3D data. In *NIPS*, 2016.

[29] Z. Li, Y. Gan, X. Liang, Y. Yu, H. Cheng, and L. Lin. LSTM-CF: Unifying context modeling and fusion with LSTMs for RGB-D scene labeling. In *ECCV*, 2016.

[30] Z. Lun, M. Gadelha, E. Kalogerakis, S. Maji, and R. Wang. 3D shape reconstruction from sketches via multi-view convolutional networks. In *3DV*, 2017.

[31] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshops*, 2013.

[32] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman. Convolutional neural networks on surfaces via seamless toric covers. *ACM Transactions on Graphics*, 36(4), 2017.

[33] A. Martinovic, J. Knopp, H. Riemenschneider, and L. J. Van Gool. 3D all the way: Semantic segmentation of urban scenes from start to end in 3D. In *CVPR*, 2015.

[34] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *ICCV Workshops*, 2015.

[35] D. Maturana and S. Scherer. VoxNet: A 3D convolutional neural network for real-time object recognition. In *IROS*, 2015.

[36] J. McCormac, A. Handa, A. J. Davison, and S. Leutenegger. SemanticFusion: Dense 3D semantic mapping with convolutional neural networks. In *ICRA*, 2017.

[37] O. Miksik, V. Vineet, M. Lidegaard, R. Prasaath, M. Nießner, S. Golodetz, S. L. Hicks, P. Pérez, S. Izadi, and P. H. S. Torr. The semantic paintbrush: Interactive 3D mapping and recognition in large outdoor spaces. In *CHI*, 2015.

[38] D. Munoz, N. Vandapel, and M. Hebert. Onboard contextual classification of 3-D point clouds with learned high-order Markov random fields. In *ICRA*, 2009.

[39] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*, 2017.

[40] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view CNNs for object classification on 3D data. In *CVPR*, 2016.

[41] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3D graph neural networks for RGBD semantic segmentation. In *ICCV*, 2017.

[42] G. Riegler, A. O. Ulusoy, H. Bischof, and A. Geiger. OctNetFusion: Learning depth fusion from data. In *3DV*, 2017.

[43] G. Riegler, A. O. Ulusoy, and A. Geiger. OctNet: Learning deep 3D representations at high resolutions. In *CVPR*, 2017.

[44] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[45] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *ICRA*, 2009.

[46] S. Salti, F. Tombari, and L. di Stefano. SHOT: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125, 2014.

[47] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*, 2017.

[48] A. Sinha, J. Bai, and K. Ramani. Deep learning 3D shape surfaces using geometry images. In *ECCV*, 2016.

[49] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *ICCV*, 2015.

[50] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3D models from single images with a convolutional network. In *ECCV*, 2016.

[51] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs. In *ICCV*, 2017.

[52] L. P. Tchapmi, C. B. Choy, I. Armeni, J. Gwak, and S. Savarese. SEGCloud: Semantic segmentation of 3D point clouds. In *3DV*, 2017.

[53] F. Tombari, S. Salti, and L. Di Stefano. Unique shape context for 3D data description. In *ACM Workshop on 3D Object Retrieval*, 2010.

[54] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik. Learning shape abstractions by assembling volumetric primitives. In *CVPR*, 2017.

[55] J. P. C. Valentin, V. Vineet, M. Cheng, D. Kim, J. Shotton, P. Kohli, M. Nießner, A. Criminisi, S. Izadi, and P. H. S. Torr. SemanticPaint: Interactive 3D labeling and learning at your fingertips. *ACM Transactions on Graphics*, 34(5), 2015.

[56] V. Vineet, O. Miksik, M. Lidegaard, M. Nießner, S. Golodetz, V. A. Prisacariu, O. Kähler, D. W. Murray, S. Izadi, P. Pérez, and P. H. S. Torr. Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction. In *ICRA*, 2015.

[57] P. Wang, Y. Liu, Y. Guo, C. Sun, and X. Tong. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Transactions on Graphics*, 36(4), 2017.

[58] C. Wu, I. Lenz, and A. Saxena. Hierarchical semantic labeling for task-relevant RGB-D perception. In *RSS*, 2014.

[59] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015.

[60] L. Yi, H. Su, X. Guo, and L. J. Guibas. SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation. In *CVPR*, 2017.

[61] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.