Efficient Convolutions for Real-Time Semantic Segmentation of 3D Point Clouds

Chris Zhang [2, 3] Wenjie Luo [1, 3] Raquel Urtasun [1, 3]
[1] University of Toronto, [2] University of Waterloo, [3] Uber Advanced Technologies Group
{chrisz, wenjie, urtasun}@uber.com

Abstract

In this work, we propose a novel voxel representation which allows for efficient, real-time processing of point clouds with deep neural networks. Our approach takes a 2D representation of a simple occupancy grid and produces fine-grained 3D segmentation. We show that our approach outperforms the state-of-the art while being an order of magnitude faster. We can perform segmentation of large outdoor scenes of size 160m x 80m in as little as 30ms. In indoor scenarios, we can segment full rooms in less than 15ms. This is crucial for robotics applications which require real-time inference for safety critical tasks.

1. Introduction

Semantic scene understanding is one of the fundamental building blocks in a wide range of applications in fields such as graphics, human-computer interaction, image search, autonomous driving and many others. Deriving semantic information from 3D point clouds is rapidly gaining traction, as 3D sensors are readily available in many of these domains. For example, self-driving cars are typically equipped with a roof-mounted LIDAR sensor. In the context of indoor scenes, 3D sensors such as Microsoft Kinect are typically used.

Deep convolutional neural networks have proven to be very powerful tools to perform semantic understanding of images in tasks such as classification [15, 27, 31, 11], detection [9, 24, 7] and semantic segmentation [16, 2, 3, 38]. Several approaches leverage these advances to deal with point clouds. Commonly, point clouds are first quantized in a process known as voxelization, with the resulting voxel grid being used as input to 3D CNNs (*e.g.* [12, 25, 32]). While results have been rather impressive, a 3D representation is inherently cubic, and can quickly become unmanageable as the point cloud grows, even with optimizations. Furthermore, most of the computations are wasted as the 3D grid is very sparse, i.e., most of the volume is empty.

An alternative is to work directly on the unstructured point cloud [20, 22] by generating per-point features and aggregating them to form a global feature representation. However, these approaches also often struggle with processing large-scale point clouds, as capturing local structure within a complex scene can be challenging. Thus, they are often implemented with a small receptive field, and large-scale point clouds must be processed in smaller individual chunks. As a consequence the segmentation process is slow and not very precise as global context is ignored. Recently, [23] proposed to use graph neural networks, where each point is a node in the graph and information is passed along the edges during inference. Impressive results were achieved, but memory consumption is an issue as learning is performed via back-propagation through time. This prevents these architectures from using very deep neural networks.

In contrast, in this paper we propose a novel voxel representation of 3D point clouds that allows for the use of 2D convolutions. We use a simple occupancy grid, and treat the gravitational axis as the feature channel. Our 2D representation is faster and requires less memory than 3D alternatives. We demonstrate the effectiveness of our approach in 3D point cloud segmentation of indoor and outdoor scenes and show state-of-the-art results, with an order of magnitude speed-up during inference.

2. Related Work

In this section, we review existing work related to the task of 3D point cloud semantic segmentation. While there exists much work on hand crafted features for point cloud segmentation (e.g., [35, 36, 10]), we focus on reviewing recently proposed deep learning approaches.

Qi *et al.* [20] propose Pointnet, a framework for direct processing of unordered sets of point clouds. A Multilayer Perception (MLP) extracts per-point features, and a max pooling layer is used as a symmetric aggregation function to form a global feature. This was extended in Pointnet++ [22] to process the points in a hierarchical fashion to better capture local structure. Results are strong on smaller point clouds, but there is little evidence to support their ability to effectively understand large, complex scenes. Experiments performed rely on small per-patch processing (1m^3

for Pointnet, 1.5m^3 for Pointnet++) and an assembling step afterwards. In contrast, our approach takes a much larger point cloud as input (often the entire scene), giving our networks a larger receptive field. Also directly working on point clouds, Wang *et al.* [34] propose deep parametric continuous convolutions, in which a parameterized kernel function that spans the full continuous vector space is learned. Results on the road scene dataset are strong, and inference speed of their best model is comparable to ours (33ms vs. 29ms). However, the KD-Tree neighbour search preprocessing step requires significant overhead, compared to our simple voxelization (28ms vs. 1ms).

Convolutional neural networks have been extensively applied for various tasks such as object detection and segmentation of 3D data [18, 21, 29, 28]. Recently, Riegler *et al.* [25] propose an efficient octree representation for 3D voxel grids. They provide implementations for CNN operations on the octree in Torch. They show runtime improvements when increasing resolution and maintaining the region of interest (*i.e.* increasing the sparsity of the occupancy grid). However, the octree has a much smaller effect when the region of interest is increased while maintaining resolution (*i.e.* increasing the size of the occupancy grid). In contrast, our method converts the 3D voxel grid into a 2D representation, with a simple regular binary occupancy grid, allowing for sparsity-invariant efficient computation with out-of-the-box methods available in all common deep learning frameworks.

There also exists work on 2D representations of 3D data. Su *et al.* [30] propose a framework that uses 2D renderings obtained from multiple different camera viewpoints for 3D object recognition. In the autonomous driving space, Chen *et al.* [4] propose a sensory-fusion framework that uses LIDAR point clouds and RGB images to perform 3D object detection. A 2D bird's eye view (BEV) representation is formed using M maximum height map slices, and a single intensity slice and density slice. Luo *et al.* [17] and Yang *et al.* [37] use a BEV representation formed directly from the occupancy grid rather than relying on a hand-crafted feature in the height domain. [4, 17, 37] primarily focus on detection in flat and sparse outdoor scenes. In contrast our approach focuses on semantic segmentation in both dense indoor scenes as well as sparse outdoor ones. Velas *et al.* [33] also propose a 2D representation of 3D LIDAR point clouds. Their approach encodes the 3D data as a 2D signal in the polar coordinate domain, utilizing ring and horizontal angle information. However, this approach has only been shown applicable in a very specific domain (ground detection for LIDAR point clouds) and is not easily generalizable to point clouds collected using other sensors or methods. In contrast, our method takes a general 2D representation in Euclidean space and thus is easily amenable to a wider range of applications.

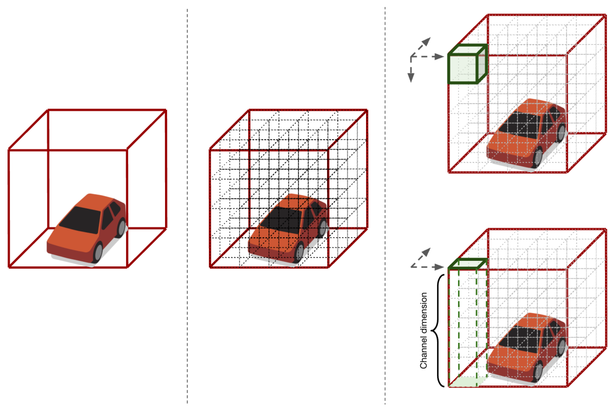

Figure 1. Left: Point cloud. Center: Voxelization step. Right: 3D convolution with a single filter (top) vs. 2D convolution with a single filter (bottom)

Huang *et al.* [12] propose a segmentation technique where the network computes a single prediction for the entire voxel grid and performs coarse segmentation using a sliding window. Dai *et al.* [6] propose a 3D FCNN architecture that produces more fine-grained predictions. Tchapmi *et al.* [32] propose a framework using a 3D FCNN architecture, where a feature vector is computed for each voxel. Voxel predictions are transferred to the points using trilinear interpolation, and results are further improved by enforcing spatial consistency using a conditional random field (CRF). In contrast, our approach computes fine-grained predictions in 3D using a 2D representation. We use a single feature vector computed for all voxels in the same Z extent to make individual predictions for each voxel.

3. Efficient Convolutions for 3D Point Clouds

Our approach can be summarized as follows. We first represent the 3D point cloud using a 2D voxel representation. We use a simple occupancy grid, and treat the gravitational axis as the feature channel. We then use a 2D CNN to ingest the voxel representation and compute per-voxel class scores. We finally project the per-voxel class scores into the 3D point cloud to obtain per-point class scores. Intuitively, this representation has three desirable characteristics: it is *simple*, *memory efficient* and *fast*.

In the following, we first formulate the problem and describe how the 2D representation is obtained. We then describe the CNN architectures and loss function we employed and describe relationships to other closely related models.

3.1. Point Cloud Voxelization

Given a set of observations $\mathbf{O} = \{\mathbf{o}_i\}$ representing a 3D point cloud, we wish to predict a probability distribution across K classes for all $\mathbf{o}_i$. Each $\mathbf{o}_i$ is a vector consisting

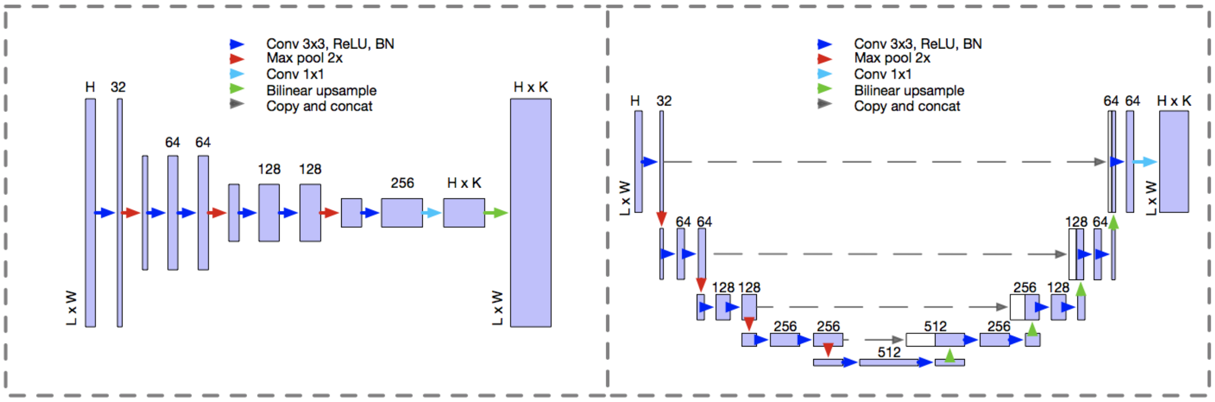

Figure 2. 2D FCNN (left), 2D U-Net (right)

of $\mathbf{p}_i$ representing its (x, y, z) location in Euclidean space, and $\mathbf{m}_i$ representing any other available sensor modalities (*e.g.*, RGB, intensity).

We begin by discretizing $\mathbf{O}$ into a representation that can be ingested by a standard CNN architecture. We define a 3D voxel grid $\mathbf{V} = \{\mathbf{v}_{x,y,z}\}$ in the Euclidean space to encompass all $\mathbf{p}_i$. Each voxel cell $\mathbf{v}_{x,y,z}$ is centred at $(\hat{x}, \hat{y}, \hat{z})$ and has a length, width, and height (l, w, h) respectively. We characterize $\mathbf{v}_{x,y,z}$ with several channels. The first channel is the occupancy channel, which we set to be 1 if there exists a point which lies in $\mathbf{v}_{x,y,z}$, and 0 otherwise. If $\mathbf{v}_{x,y,z}$ is occupied, any additional sensor modalities $\mathbf{m}_i$ are encoded as additional channels (*e.g.*, RGB information would be encoded in three additional channels). Note that if multiple points lie within the same $\mathbf{v}_{x,y,z}$, a simple pooling operation can be performed. In our experience, we see that the specific pooling operation (max, average, or random selection of a representative point) does not have a great effect on performance.

3.2. From 3D to 2D

In standard CNN architectures, the input of a N dimensional convolution layer is a $N+1$ dimensional tensor, with N spatial dimensions and a single feature dimension. For example, a 3D CNN can be used to ingest the voxel grid $\mathbf{V}$, with the input being a tensor of size $L \times W \times H \times C$ with L, W, H representing the dimensions of the voxel grid, and C representing the number of channels in each voxel cell.

In contrast, here we treat the voxel grid $\mathbf{V}$ with voxel cells $\mathbf{v}_{x,y,z}$ as a voxel *image* $\mathbf{V}'$ with voxel *patches* $\mathbf{v'}_{x,y}$. $\mathbf{V}'$ is essentially a bird's eye view of $\mathbf{V}$, with the gravitational axis z as the feature channel. See Figure 1 for a visualization. Sensor information encoded as additional channels in each voxel cell can be flattened, resulting in a matrix of size $L \times W \times (HC)$ as the input to our 2D CNN.

3.3. Model Architecture

We explore two different architectures to process our 2D voxel representation.

2D Fully Convolutional Neural Network: Our first architecture is a modification of VGG [27]. We refer the reader to Figure 2 (left) for an illustration. Our network takes as input the voxel image $\mathbf{V}'$. We then use a set of 2D convolution layers with kernel size 3 and stride 1, interlaced with 2D max-pooling layers with kernel size 2 and stride 2. Each convolution layer is followed with a ReLU activation layer and batch normalization [13]. We double the number of output channels each time $\mathbf{V}'$ is downscaled, keeping the amount of computation in each layer approximately equal.

Towards the final layers, the network learns to compute a feature vector $f_{x,y}$ for each voxel patch $v'_{x,y}$ in the now downscaled $\mathbf{V}'$. Note that because we are using a 2D bird's eye view representation, the feature vector $f_{x,y}$ corresponding to the voxel patch $v'_{x,y}$ must encode information for the voxel cells $v_{x,y,z}$. In other words, a *single* feature vector encodes class information for *multiple* voxel cells along the gravitational axis. To overcome this issue, we decode $f_{x,y}$ using a 1×1 convolution layer, with $H \times K$ number of output channels, which represents a separate probability distribution across K classes for each voxel cell $v_{x,y,z}$.

A softmax is then applied to each voxel cell in the down-scaled voxel grid to obtain class predictions. We bilinearly interpolate the per-voxel predictions in the downscaled voxel grid back to its original size. We then use nearest neighbor interpolation to obtain per-point predictions. This has a similar effect as the strategy used in [32], where per-point predictions are instead directly obtained through trilinear interpolation of the downscaled voxel grid. The advantage of our approach is the ability to train on a per-voxel metric rather than a per-point metric. This is preferred as we found using a per-point metric to be computationally costly without much impact on results.

2D U-Net: Pooling layers achieve increased receptive fields and spatial invariance at the cost of resolution, which can negatively impact semantic segmentation tasks. We choose to adopt the U-Net [26, 5] architecture as shown in Figure 2 (right) as a method to address this issue. The U-Net structure consists of an encoder and decoder network. The encoder network consists of convolution and pooling layers, eventually learning strong, context-aware features, at the cost of lost boundary information during pooling. The decoder network consists of convolution and bilinear upsampling layers, which further process the features learned by the encoder network. Importantly, at each step, the activations from the corresponding layer in the encoder network are copied and concatenated in the channel dimension. Intuitively, this allows the network to see high definition boundary information while being aware of the larger context.

3.4. Loss Function

Severely class imbalanced datasets can hinder performance, especially if the under-represented classes are considered more important. For example, this arises in autonomous driving scenarios where pedestrians and bicyclists are much less common than other categories such as road or background. To address this issue, we train the network using a class-balanced weighted cross-entropy loss similar to [8] defined as,

$$H(y, x) = -\sum_i \alpha_i \, p(y_i) \log\left(p(x_i)\right) \tag{1}$$

$$\alpha_i = median\_freq/f_i \tag{2}$$

where f_i is the number of points of class i in the training set divided by the total number of points in the training set, and $median\_freq$ being the median of frequencies f_i.

As mentioned earlier, we choose to train the network with a loss defined on a per voxel metric rather than a per-point metric because projecting voxel predictions to the point cloud is computationally costly but not necessarily advantageous. Note that all unoccupied voxels are treated as a don't care class, with $\alpha = 0$.

3.5. Relation to other methods

3D convolutions: Notice that our approach can also be described in the framework of 3D convolutions. Lets define a 2D FCNN with convolution layers of kernel size $k \times k$ to ingest our 2D representation of an input voxel grid of size $L \times W \times H \times C$. We can also define a 3D FCNN such that the first convolution layer has a kernel size $k \times k \times H$, and define subsequent convolution layers with kernel size $k \times k \times 1$. These two networks will be identical.

Seperable filters: We can also compare our method to spatially separable filters explored in [19], where given a 3D kernel U, the goal is to find 1D kernels U_x, U_y, U_z such that

$$f * U = f * U_x * U_y * U_z \tag{3}$$

where $*$ denotes the convolution operator, and U_x, U_y, U_z are applied along the x, y, z axis respectively. While the separable filter would approximate a 3D convolution with kernel size $k \times k \times k$ using 1D convolutions with kernel size $k \times 1 \times 1$ along each axis, our approach uses a $k \times k \times 1$ kernel, and collapses the 3rd dimension into the feature channel.

4. Experimental Results

We evaluate our model on various 3D datasets and empirically showcase both the strengths and weaknesses of our approach. First, we evaluate on a dense, indoor benchmark dataset. Following that, we evaluate on a new, sparse, outdoor road scene dataset. By using two contrasting datasets, we show that our 2D representation is generalizable and applicable to a variety of problems.

We train all our models using Adam optimizer [14], with learning rate 0.0001, weight decay 0.0005, and betas 0.9, 0.999. We implement our experiments using PyTorch. Although we train on a per-voxel loss, all metrics subsequently reported in this paper are per-point. The primary metrics we use for evaluation are mean class accuracy (mAcc) and mean class IOU (mIOU). We define mAcc as

$$mAcc = \frac{1}{K} \sum_i^K \frac{tp_i}{tp_i + fn_i} \tag{4}$$

where K is the number of classes, tp is the number of true positives, and fn is the number of false negatives. Similarily, we define mIOU as

$$mIOU = \frac{1}{K} \sum_i^K \frac{tp_i}{tp_i + fn_i + fp_i} \tag{5}$$

where K is the number of classes, tp is the number of true positives, fn is the number of false negatives and fp is the number of false positives.

4.1. Stanford Large-Scale 3D Indoor Spaces Dataset

The Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [1] dataset contains Matterport 3D scans of 6 indoor areas in 3 different buildings, covering a total of 270 rooms and over 6000m^2. Each point contains RGB and global XYZ information, along with a semantic label from 1 of 13 classes (7 structural classes, 5 moveable classes and 1 clutter class for all other elements). To be comparable to [32], we also train on Areas 1, 2, 3, 4, 6 and test on Area 5.

Method	mIOU	mAcc	runtime (ms)
Pointnet [20]	41.09	48.98	-
3D-FCNN-TI [32]	47.46	54.91	435
SEGCloud [32]	48.92	57.35	-
2D-FCNN(ours)	44.13	62.85	7
2D-Unet(ours)	51.27	64.69	13
2D-Unet+RGB(ours)	**51.76**	**68.28**	14

Table 1. S3DIS dataset results, time calculated for inference of 16m x 16m x 5m

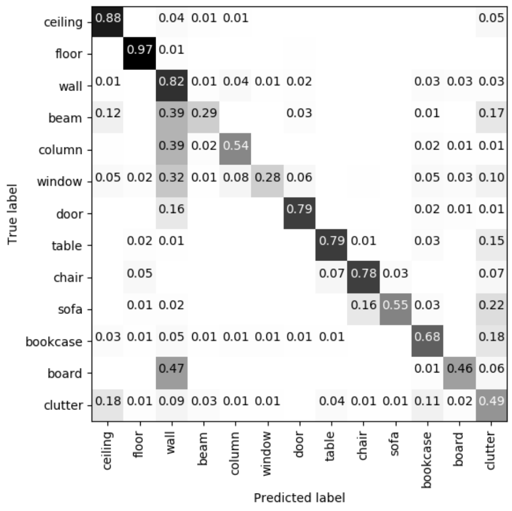

Figure 3. S3DIS dataset results: confusion matrix for 2D-U-Net

We process our training data by dividing each room into blocks of size 16m×16m (X, Y) and keeping the entire Z extent to ensure that the floor and ceiling are present in every block. Rooms larger than 16m×16m are represented with multiple, overlapping blocks. Each block is then divided into a voxel grid in a manner as described in Section 3.1. First, we experiment with a model that only uses voxel occupancy, discarding all RGB information. We divide the voxel grid into 320×320×50 voxel cells with dimensions 5cm×5cm×10cm. Our initial voxel grid only covers 16m×16m×5m, so we include a bucket bin above 5m to include any remaining points. This makes the final voxel grid a 3D tensor of size 320×320×51. To supplement the training data, we perform data augmentation, using random rotation about the gravitational axis and scaling [32]. Next, we experiment with incorporating the RGB information available. Recall that RGB information is represented as additional channels in each voxel cell. To take a 2D representation, we flatten these additional channels into the feature dimension, effectively expanding our input feature channel size by a factor of 4. This makes our input for RGB models a 3D matrix of size 320×320×204.

For evaluation, we process the test data by dividing rooms into blocks of 16m×16m, with no overlapping blocks. Thus the network computes a class prediction once for every point.

Runtime measurements: Runtime measurements reported in Table 1 represent the amount of time required to achieve predictions on a 16m×16m block, ignoring the initial voxelization step and the final step of projecting voxel predictions back to the point cloud. We implement 3D-FCNN-TI without the final trilinear interpolation to obtain timings. We do not implement SEGCloud for timing, as it is simply 3D-FCNN-TI with an additional CRF step. Pointnet reports roughly 1,000,000 points/second, and the S3DIS dataset contains roughly 100,000 points/m^2, implying a runtime of roughly 25s using our block sizes. However, per-point timings are omitted as they are not comparable with per-voxel timings especially on such a dense dataset. Note that for this dense dataset, our voxelization and devoxelization steps add 20ms to the pipeline in total.

Analysis: We see that both our vanilla 2D-FCNN and 2D-U-Net outperforms Pointnet. Pointnet relies on computing a global feature vector, and we argue that it is difficult to find a meaningful global feature for large complex scenes. As a result, they must perform segmentation on small 1m×1m×1m blocks, limiting the receptive field. In contrast, our method is able to leverage CNN architectures to learn hierarchical features with varying contextual scales.

We see that while our 2D-FCNN underperforms, our 2D-U-Net outperforms 3D-FCNN-TI and SEGCloud, with an order of magnitude speed up. Despite the fact that rooms in the S3DIS dataset are relatively cuboid, our 2D representation can still compete. However, our models noticeably struggle with the ceiling class, which other methods seem to perform well with ease. The confusion matrix in Figure 6 shows that our network often confuses ceiling with clutter. After inspecting some example rooms, we see that there exists a few rooms with much higher than average ceilings, and a plethora of clutter underneath. Because our network is not spatially invariant to the gravitational axis, it is difficult to accurately classify the clutter situated at the most common ceiling heights. On the other hand, our network performs very strongly on classes that can be distinguished easily from the bird's eye view, (*e.g.* the door class). We also notice a stronger performance in the board class. We argue that a high resolution is a strong asset when distinguishing board from wall (to notice the small protrusion). Because Pointnet does not discretize the point cloud, they are able to outperform SEGCloud in this class. However, because our network is not invariant in the gravitational axis, we are more easily able to learn the average height of a board, providing a helpful prior during inference. In addition, our network uses more aggressive down-scaling,

Method	ceiling	floor	wall	beam	column	window	door	chair	table	bookcase	sofa	board	clutter
Pointnet [20]	88.80	**97.33**	69.80	0.05	3.92	**46.26**	10.76	52.61	58.93	40.28	5.85	26.38	33.22
3D-FCNN-TI [32]	**90.17**	96.48	70.16	0.00	11.40	33.36	21.12	**76.12**	70.07	57.89	37.46	11.16	**41.61**
SEGCloud [32]	90.06	96.05	69.86	0.00	18.37	38.35	23.12	75.89	70.40	**58.42**	40.88	12.96	41.60
2D-FCNN(ours)	71.07	92.23	51.50	**0.42**	24.78	26.78	35.75	62.91	66.19	47.89	51.81	16.74	25.63
2D-U-Net(ours)	81.66	94.46	**70.41**	0.05	28.12	40.50	47.39	62.75	72.22	50.04	57.74	27.63	33.59
2D-U-Net+RGB(ours)	79.77	93.93	68.99	0.20	**28.26**	38.53	**48.28**	71.09	**73.59**	48.72	**59.20**	**29.27**	33.10

Table 2. S3DIS dataset results, class IOU

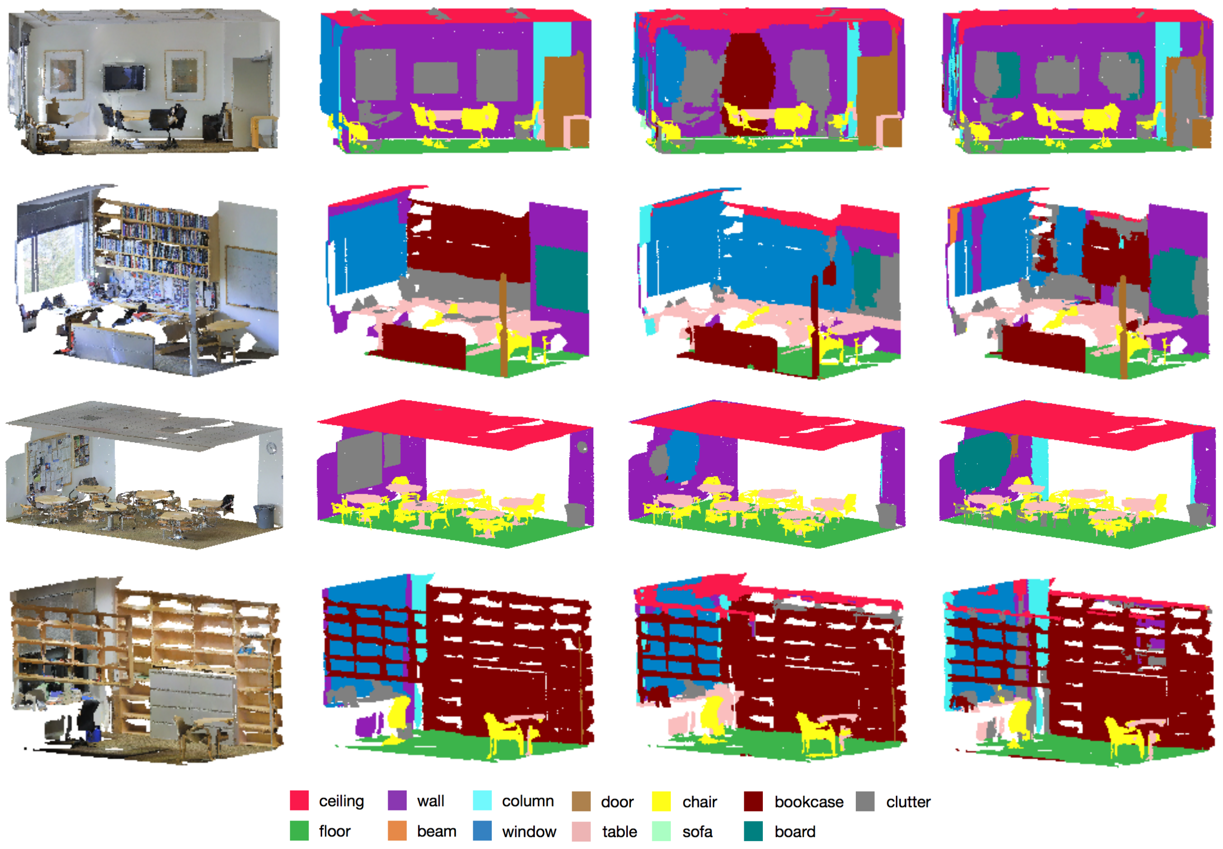

Figure 4. S3DIS qualitative results, from left to right: raw, ground truth, 2D-FCNN, 2D-U-Net

allowing for a larger receptive field. We are able to do this because we supplement our network with the more computationally heavy U-Net architecture. While methods explored in [32] would likely benefit from such a modification as well, our representation's efficiency allows us to do so without increasing memory consumption and inference time to an unreasonable amount.

Through our experiments with the S3DIS dataset, we demonstrate that the 2D representation can (perhaps counter-intuitively) work well for scenes that are not just flat, achieving state-of-the-art results, with speed-ups in the range of an order of magnitude.

Method	mIOU	mAcc	runtime (ms)
Pointnet [20]	38.05	46.97	-
3D-FCNN (ours)	47.35	68.83	137
3D-U-Net(ours)	**58.02**	**81.47**	306
2D-FCNN (ours)	48.98	80.42	12
2D-U-Net (ours)	56.14	79.72	29

Table 3. 3D Road Scene Dataset results, time calculated for inference of 160m x 80m

4.2. 3D Road Scene Dataset

We perform experiments on a new, very large-scale 3D road scene dataset, which contains annotated point

Method	vehicle	bicyclist	pedestrian	motorcycle	animal	road	background
Pointnet [20]	76.73	2.85	6.62	8.02	0.0	91.96	**89.83**
3D-FCNN (ours)	82.67	21.38	33.64	17.97	0.22	90.82	84.73
3D-U-Net(ours)	**91.29**	**43.35**	43.91	**45.01**	**3.24**	92.40	86.96
2D-FCNN (ours)	84.31	15.89	36.88	27.30	1.65	91.86	85.04
2D-U-Net (ours)	91.15	27.41	**51.44**	41.19	1.55	**92.45**	87.82

Table 4. 3D Road Scene Dataset results, class IOU

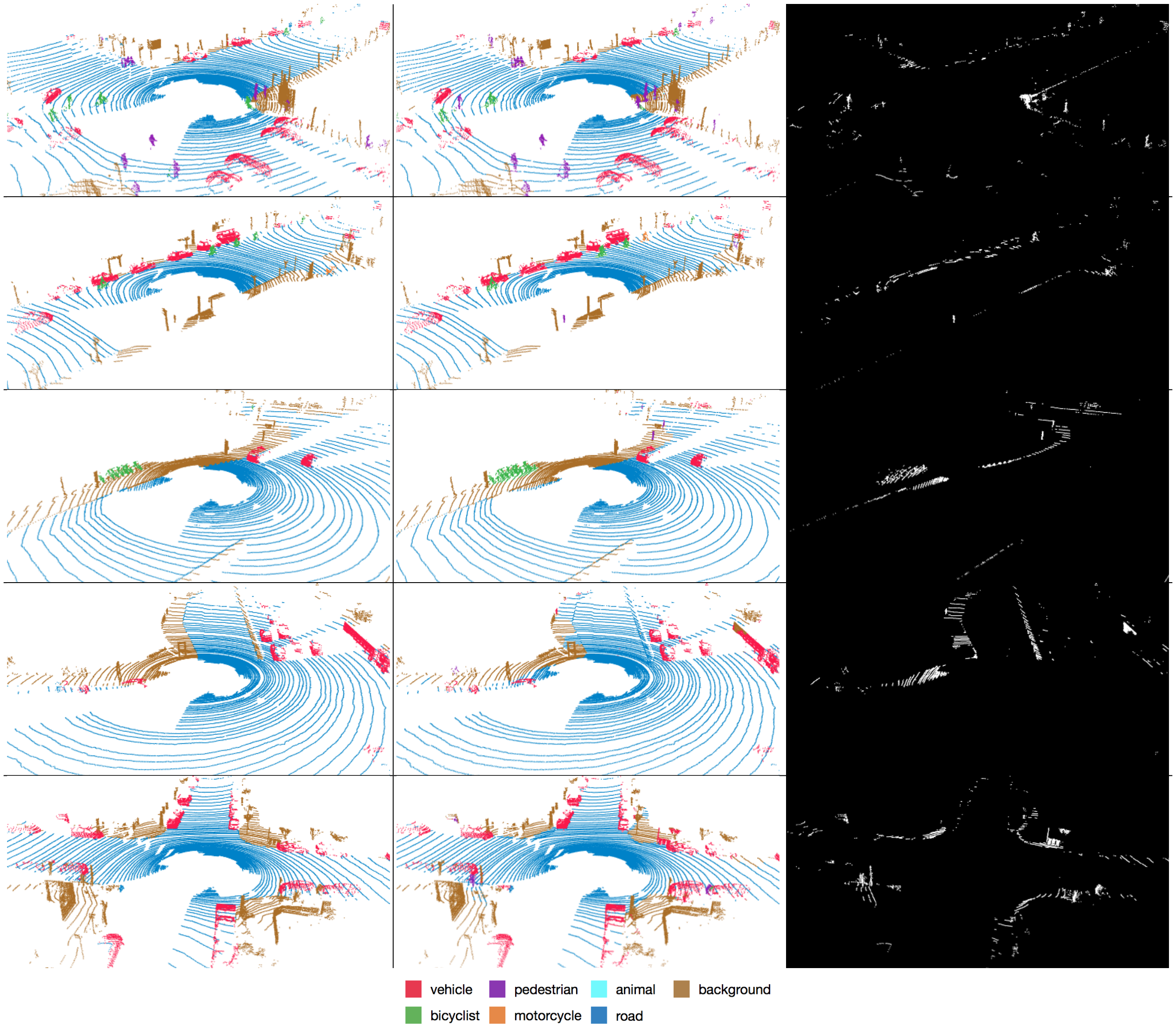

Figure 5. 3D Road scene dataset qualitative results, from left to right: ground truth, 2D-U-Net, Errors

clouds obtained from video snippets captured using a roof-mounted Lidar. This dataset is very large scale and contains more than 200 billion points. Our per point annotations contained 7 classes (vehicle, pedestrian, bicyclist, motorcyclist, animal, road, and background). As expected from a real-world dataset of this nature, class imbalance is present in this dataset, with the background, road and vehicle class making up 61%, 31% and 7% of all points respectively, leaving approximately 1% for the remaining classes. The dataset is composed of snippets, where each snippet con-

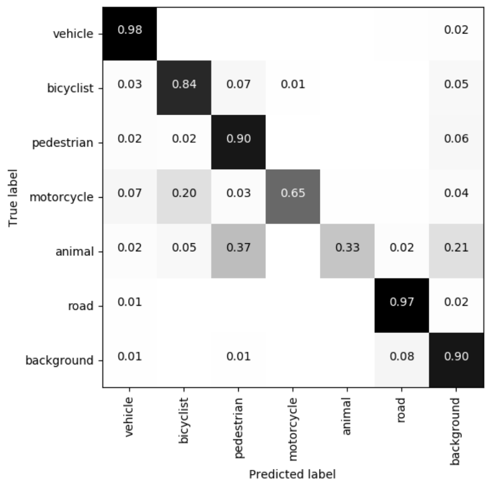

Figure 6. 3D Road Scene dataset results: confusion matrix for 2D-U-Net

tains approximately 250 frames, with each frame containing approximately 100 000 points. We randomly partition the dataset on a snippet level to obtain separate train and test snippets. We test on 16 000 randomly sampled frames from the test snippets.

We use the same networks used in the S3DIS experiment. For a fair comparison, we also implement the 3D version of our 2D networks. To do so, we replace all 2D layers with their 3D counterparts, and further divide the output channel by 2. Note that our resulting 3D FCNN network can be compared to [32], but more lightweight, with more aggressive down-sampling and the CRF post-processing step omitted. All models are trained using the weighted cross entropy loss detailed in Section 3.4. Because of the severe class imbalance, the weighted cross entropy can be very unstable, so we clip all gradients to a magnitude of 1. We train on a region of interest of size 64m$\times$64m$\times$4m, with the ego-car in the center (32m in front/behind the car, 32m to the left/right of the car, and 4m above the ground). We use voxel cells with size 20cm, 20cm, 10cm for our models. Due to the sheer size of the dataset, we do not perform any data augmentation during training. For evaluation, we test on a 160m$\times$80m$\times$4m region of interest, again with the ego-car in the center.

In addition to our own voxel-based approaches, we also implement a Pointnet [20] model for comparison. Unlike experiments in the S3DIS dataset, our implemented Pointnet model takes as input the entire point cloud, rather than 1m blocks. This is done in the interest of inference speed, as segmenting small chunks of the point cloud would be too slow for our requirements. Quantitative results are shown in Tables 3 and 4, and qualitative results in Figure 5.

Runtime measurements: Runtime measurements reported in Table 3 represent the amount of time required to achieve predictions on a 160m$\times$80m$\times$4m block, ignoring the initial voxelization step, and final step of projecting voxel predictions back to the point cloud. Pointnet takes roughly 100ms per ROI, but per-point timings are omitted as they are not comparable with per-voxel timings. For this sparse dataset, our voxelization and devoxelization steps add 1ms to the pipeline in total.

Analysis: Results show we outperform Pointnet in this dataset as well. We see that when taking in the entire point cloud, the Pointnet architecture has difficulty capturing the detailed local structure in complex road scenes. As a result, Pointnet has a strong performance in large classes (*e.g.*, background, road) but poor performance in small classes (*e.g.*, pedestrian, animal).

We see that the 3D U-Net outperforms the 2D U-Net in this experiment. This somewhat verifies our early prediction that 3D models would equally benefit from enhancements such as the U-Net architecture. However, its slow runtime makes it inapplicable in real-time scenarios. On the other hand, we see that 2D networks are still quite competitive with their 3D counterparts while being much faster.

Our experiments show that by using an efficient 2D representation, we can allocate compute resources on more effective methods to increase performance, such as more expressive network architectures, thereby allowing us to achieve quality segmentation results on large-scale point clouds in real-time.

5. Conclusion

In this work, we proposed a novel 2D voxel representation that allows for real-time semantic segmentation of 3D point clouds. Our approach is simple, efficient, fast and has great applications in many of today's challenges, such as autonomous driving. We developed an intuitive understanding of our model's strengths and weaknesses, and empirically compared our approach with alternative methods. We demonstrated our representation's efficiency and generalizability, and showed state-of-the-art results with significant speed-ups on varying datasets. While we have only experimented with two architectures in this work, further experiments with state-of-the-art 2D semantic segmentation architectures and applying them to 3D point cloud semantic segmentation using our voxel representation is a potential area for improvement. Applying our representation to other 3D tasks such as detection is another interesting avenue of future research.

References

[1] 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 *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016. 4

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015. 1

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016. 1

[4] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE CVPR*, 2017. 2

[5] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. Springer, 2016. 4

[6] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 2

[7] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016. 1

[8] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015. 4

[9] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 1

[10] T. Hackel, J. D. Wegner, and K. Schindler. Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(3), 2016. 1

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[12] J. Huang and S. You. Point cloud labeling using 3d convolutional neural network. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2670–2675, Dec 2016. 1, 2

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 3

[14] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[16] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 1

[17] W. Luo, B. Yang, and R. Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3569–3577, 2018. 2

[18] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *IROS*, 2015. 2

[19] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997. 4

[20] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1, 5, 6, 7, 8

[21] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2016. 2

[22] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. 2017. 1

[23] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. 3d graph neural networks for rgbd semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5199–5208, 2017. 1

[24] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1

[25] G. Riegler, A. Osman Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1, 2

[26] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]). 4

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 3

[28] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng. Convolutional-recursive deep learning for 3d object classification. In *Advances in Neural Information Processing Systems*, pages 656–664, 2012. 2

[29] S. Song and J. Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 808–816, 2016. 2

[30] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015. 2

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1

[32] L. P. Tchapmi, C. B. Choy, I. Armeni, J. Gwak, and S. Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *International Conference on 3D Vision (3DV)*, 2017. 1, 2, 3, 4, 5, 6, 8

[33] M. Velas, M. Spanel, M. Hradis, and A. Herout. CNN for very fast ground segmentation in velodyne lidar data. *CoRR*, abs/1709.02128, 2017. 2

[34] S. Wang, S. Suo, W.-C. M. A. Pokrovsky, and R. Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018. 2

[35] M. Weinmann, B. Jutzi, and C. Mallet. Feature relevance assessment for the semantic interpretation of 3d point cloud data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 5:W2, 2013. 1

[36] M. Weinmann, S. Urban, S. Hinz, B. Jutzi, and C. Mallet. Distinctive 2d and 3d features for automated large-scale scene analysis in urban areas. *Computers & Graphics*, 49:47–57, 2015. 1

[37] B. Yang, W. Luo, and R. Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018. 2

[38] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *arXiv preprint arXiv:1612.01105*, 2016. 1