Interpolation-Aware Padding for 3D Sparse Convolutional Neural Networks

Yu-Qi Yang[1,2] Peng-Shuai Wang[2] Yang Liu[2]
[1]Tsinghua University [2]Microsoft Research Asia
yangyq18@mails.tsinghua.edu.cn {penwan,yangliu}@microsoft.com

Abstract

Sparse voxel-based 3D convolutional neural networks (CNNs) are widely used for various 3D vision tasks. Sparse voxel-based 3D CNNs create sparse non-empty voxels from the 3D input and perform 3D convolution operations on them only. We propose a simple yet effective padding scheme — interpolation-aware padding *to pad a few empty voxels adjacent to the non-empty voxels and involve them in the 3D CNN computation so that all neighboring voxels exist when computing point-wise features via the trilinear interpolation. For fine-grained 3D vision tasks where point-wise features are essential, like semantic segmentation and 3D detection, our network achieves higher prediction accuracy than the existing networks using the nearest neighbor interpolation or the normalized trilinear interpolation with the zero-padding or the octree-padding scheme. Through extensive comparisons on various 3D segmentation and detection tasks, we demonstrate the superiority of 3D sparse CNNs with our padding scheme in conjunction with feature interpolation.*

1. Introduction

Effective 3D representations for 3D deep learning like voxels [2, 30], point sets [19, 12], and polygonal meshes [25, 11], have been actively studied in recent years. Among them, voxel-based representations are natural extensions from 2D pixels and are compatible with regular-grid-based convolutional operations and suitable for fast GPU processing. However, due to the high cost of memory storage and CNN computation on dense 3D grids, dense-voxel-based 3D CNNs are limited to coarse resolution inputs like 32^3 grids, and cannot handle and generate high-resolution 3D contents. To overcome this limitation, sparse-voxel-based CNNs [26, 21, 9, 3] are proved to be a computational and memory-efficient solution, where only voxels around 3D shapes are created for storing feature channels. With regard to the prediction accuracy on 3D tasks, spare-voxel-based CNNs dominate several large-scale benchmarks, including ScanNet segmentation [6], KITTI segmentation [1], and ScanNet detection [18].

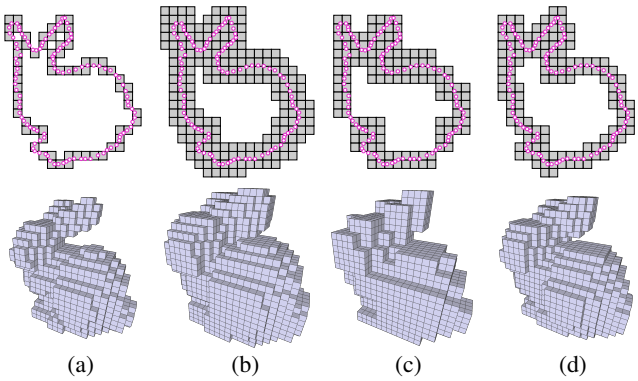


Figure 1: Various padding schemes for a 2D point set (upper) and a 3D point set (bottom). (a): sparse pixels/voxels without padding, *i.e.*, zero padding; (b): with 1-ring padding; (c): with quadtree-padding or octree-padding; (d): with our interpolation-aware padding. Red circles present 2D points sampled from a 2D bunny shape, 3D points are omitted here as they are hidden inside the voxels.

Sparse-voxel-based CNNs can be used naturally for extracting features for each discrete voxel. However, point-wise features are essential for fine-grained tasks, like 3D segmentation and detection. A common practice in sparse-voxel-based CNNs is to assign the feature of the nearest voxel to the point, *i.e.*, using *the nearest neighbor interpolation* [26, 9]. To extract distinguishable point features within the same voxel, *voxel-based trilinear interpolation* can be adopted. The trilinear interpolation requires querying features from the nearby eight voxels, whereas some nearby voxels may not exist due to the sparse-voxel representation. Specifically, octree-based 3D CNNs [26, 27] perform the CNN computation on 8 sibling octants of non-empty octants (see Fig. 1-(c)); the recent sparse-voxel-based CNNs [9, 3, 22] use Hash tables to index the non-empty voxels, and perform the CNN computation only in the non-empty voxels (see Fig. 1-(a)). To do the interpolation, existing works either assign zero features to those non-existing voxels [3], or use normalized interpolation weights to compromising the missing voxels [28, 23].

We observe that current interpolation schemes are

not optimal, and even yield worse performance than the nearest-neighbor interpolation in some experiments. Our key idea is to let the network learn features for the empty voxels, and use these learned features to do the interpolation, instead of using zero features. To this end, a naïve solution is to pad 1-ring neighbors of all original non-empty voxels when building the sparse voxel grids, denoted as *1-ring padding* (see Fig. 1-(b)). Then these augmented voxels can be used as the input of CNNs, and all voxel features required by the interpolation can be computed by CNNs. However, this may incur great computation and memory costs. Considering that what we need are the point features, we can only pad those voxels required by the interpolation of each point (see Fig. 1-(d)). We term our padding scheme as *interpolation-aware padding*. With our padding scheme, the interpolation is well-defined for each point, and the network performance can be greatly improved compared with the previous interpolation schemes. Compared with the 1-ring padding, the memory cost of interpolation-aware padding is also much less.

Our improved interpolation also facilitates the network design. Previous sparse-voxel-based CNNs for segmentation and detection output the features of discrete voxels [26, 3], the resolution of the output voxels has to be high enough to differentiate different points, otherwise the performance may decrease dramatically. With well-defined interpolation, our network instead outputs *coarse* resolution voxel features to extract expressive point-wise features, and further reduces the computation and memory cost. Apart from the benefits of the interpolation, we also compare the four different padding schemes in Fig. 1. Interestingly, our experiments reveal that padding itself also boosts the network prediction accuracy, even without the trilinear interpolation.

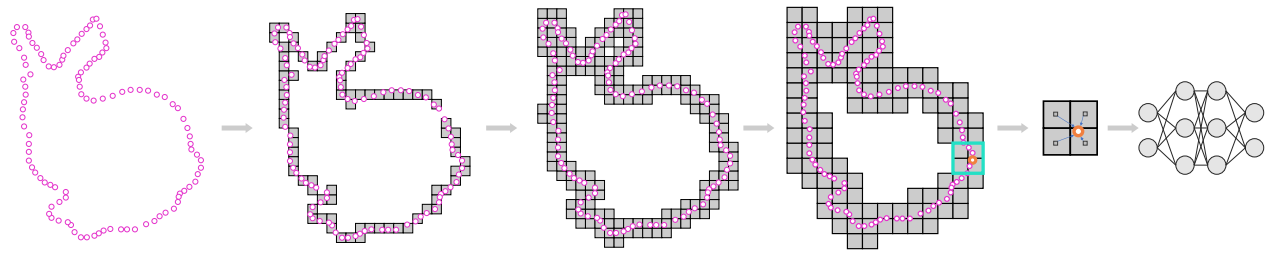
Our contributions include the improved interpolation for sparse-voxel CNNs by the interpolation-aware padding scheme, and network architectures for 3D segmentation and detection which can produce point-wise features without using very high-resolution voxels. To validate the efficacy and superiority of our method, we performed a series of experiments and comparisons on several typical 3D semantic segmentation and 3D detection tasks. Compared with the network using zero-padding and without interpolation, our network with the same number of trainable parameters improves the mean Intersection over Union (mIoU) of segmentation on PartNet [17], ScanNet [6] and KITTI [1] by 2.2, 2.0, and 2.4, respectively, and improves the mAP of detection on ScanNet [6] by 2.0. Compared with the 1-ring padding, our interpolation-aware padding scheme is more practical with less memory consumption. In the supplemental material, we provide our code for reproducing all the results easily. We believe our effective interpolation and sparse padding scheme will be a powerful plug-in for sparse 3D CNNs and benefit more broad applications.

2. Related Work

Dense voxel-based 3D CNNs Grid-based voxelization is a popular 3D discretization method in computer graphics and computer vision. The dense voxels are natural extensions of 2D pixels and suitable for building 3D convolutional neural networks from them. For 3D closed shapes, early works [16, 31] represent them as indicator functions or distance fields on dense voxels and apply 3D CNNs for recognizing 3D objects. Brock *et al*. [2] create a voxel-based 3D variational autoencoder for synthesizing 3D shapes. Choy *et al*. [5] bring the recurrent neural network to voxel-based 3D decoders for inferring 3D shapes from multiview images. Voxel-based 3D generative adversarial networks (GAN) further enhance the generated shape quality [30]. However, in 3D, dense voxels occupy $\mathcal{O}(n^3)$ storage spaces and lead to the costly CNN computation. In practice, dense voxel-based 3D CNNs work on low-resolution grids only.

Sparse voxel-based 3D CNNs The sparsity of 3D data can be utilized to improve the efficiency of 3D CNNs. Observing that most of the voxels have duplicated features, Riegler *et al*. [21] use a hybrid grid-octree to build 3D CNNs to support 3D learning at high resolutions. As a 3D surface is usually a 2D manifold or the collection of 2D manifold patches, it only occupies limited spaces in 3D. By discretizing the shape surface into voxels only, the surface can be represented by a set of sparse voxels. Wang *et al*. [26] propose octree-based CNNs (O-CNN), where the CNN computation only takes place in the non-empty octants and their sibling octants in different octree levels. Submanifold sparse convolutional networks [9] and MinkowskiNet [3] further restrict the storage and CNN computation on the non-empty voxels only, and result in less memory occupation and computation costs. The spatial-hashing-based CNN [22] improves the efficiency of the hashing table by avoiding hash collision and reducing memory overhead. Instead of using as minimal as possible voxels in CNN, our work shows that padding empty voxels properly can improve the performance of sparse 3D CNNs, without enlarging network weight parameter sizes.

Sparse convolution The 3D sparse convolution widely used for 3D signals [26, 9, 3] is essentially a restricted version of a dense convolution, where the input feature maps on the non-existing voxels are set to zero during the convolution computation. The convolution operation can be efficiently implemented by the GPU-version sparse matrix multiplication. The convolution results can be further normalized by the number of non-zero features to achieve better performance for depth completion and image inpainting tasks [24, 14, 13, 10]. In our work, we simply use the unnormalized sparse convolution.



(a) Input point cloud (b) Voxelization (c) Interpolation-aware padding (d) CNN features (e) Interpolation (f) MLP

Figure 2: Overview of our interpolation-aware padding scheme (2D illustration). The input point cloud (a) is first voxelized (b) based on a given spatial resolution, and after voxelization, each of the voxel contains at least 1 point. Then the sparse non-empty voxels are padded so that all neighboring voxels used for interpolating each point exist (c). The sparse voxels are processed via deep sparse-voxel-based CNNs, and CNN features (d) are produced. Note that the spatial resolution of output CNN features (d) may be coarser than the input (b). For the orange point highlighted by a green box in (d), its neighboring voxels are retrieved, and the point feature is computed via interpolation (e). Finally, the point-wise features are used as the input of a shared MLP (f) for segmentation and detection tasks.

Trilinear feature interpolation on sparse voxels For tasks requiring point-wise features, the voxel features produced by sparse-voxel-based CNNs can be scattered to each point via trilinear interpolation. Trilinear interpolation on sparse voxels is not thoroughly evaluated and two different implementations spread in literature and the code repositories. Since not all neighboring voxels exist, Choy *et al.* [3] propose to use the zero feature directly if the corresponding voxel does not exist. Wang *et al.* [28] and Tang *et al.* [23] use the normalized interpolation weights to satisfy the partition of unity property in their implementation. With our interpolation-aware padding, we can directly use the original interpolation formula. Mao *et al.* [15] propose a new point-based convolution operation based on trilinear interpolation. Here we focus on point-wise feature extraction for sparse-voxel-based CNNs and omit the comparison with [15].

3. Method

Given an input point cloud, we train a sparse-voxel-based CNN to extract point-wise features for the downstream tasks like 3D segmentation and detection. The overall pipeline is shown in Fig. 2. The input point cloud is first quantized and rounded to sparse voxels with a user-specified spatial resolution. Instead of directly performing sparse convolutions on these non-empty voxels [9, 3, 22], we pad a set of empty voxels required by the interpolation for each point and set the initial input features of these empty voxels as zero. After a set of CNN operations, each of the voxels contains its extracted features. For a query 3D point, its eight nearest neighboring voxels are retrieved, and the voxel features are interpolated to produce its point-wise feature. Then the point-wise feature is processed via two fully-connected layers and used for each specific task.

In the following sections, we first introduce our novel interpolation-aware sparse padding and the trilinear interpolation in Sec. 3.1, network designs in Sec. 3.2.

3.1. Interpolation-aware Sparse Padding

The padding for sparse-voxel-based CNNs here is fundamentally different from the conventional 2D image padding. We pad extra voxels around the empty voxels as the input. Initially, the input features of the padded voxels are set as zero, and then all voxel features are dynamically produced by the CNN operations. For the 2D convolution on images, *virtually*-padded pixels are needed for pixels close to the image boundary due to the fixed shape of convolution kernels. And the features of the padded pixels are directly set to zeros (*i.e.*, zero padding) or the feature maps at the reflection pixels on the image boundary (*i.e.*, reflection padding).

Our goal is to pad extra voxels around the non-empty voxels so that all the eight nearest neighboring voxels exist when doing the trilinear interpolation for each point. The trilinear interpolation inside a 3D regular grid [29] can be written as:

$$\mathbf{f}(x, y, z) := \frac{\sum I_{ijk} \cdot \mathrm{vol}_{ijk} \cdot \mathbf{f}_{ijk}}{\sum I_{ijk} \cdot \mathrm{vol}_{ijk}}, \tag{1}$$

where $i, j, k \in \{0, 1\}$ are the indices of the eight grid corners, vol_{ijk} is the partial volume surrounded the by the query point and the corner which is diagonally opposite to the corner with index ijk, $\mathbf{f}_{ijk}$ is the feature vector associated on corner ijk, and $I_{ijk} \in \{0, 1\}$ is indicates whether the corresponding voxel exists. In the following, we examine two baseline padding schemes and propose our interpolation-aware padding scheme for balancing the network performance and the runtime memory cost.

Octree padding The work of O-CNN [26] explicitly builds sparse voxels using octrees. The octree padding is an outcome of the octree data structure since each octree node has 8 children nodes. Fig. 1-(c) illustrates an octree padding and its 2D counterpart. However, the eight nearest neighboring voxels used for interpolation of each input point are still

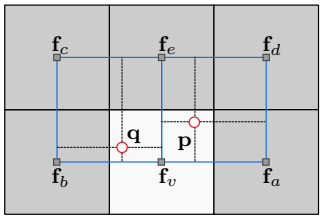


Figure 3: 2D illustration of interpolation-aware padding. Two points $\mathbf{p}$ and $\mathbf{q}$ are inside the light gray pixel. Because the bilinear interpolation at these two points queries the CNN features $\mathbf{f}_a, \mathbf{f}_b, \mathbf{f}_c, \mathbf{f}_d, \mathbf{f}_e, \mathbf{f}_v$, dark gray pixels are explicitly added into the sparse pixel set.

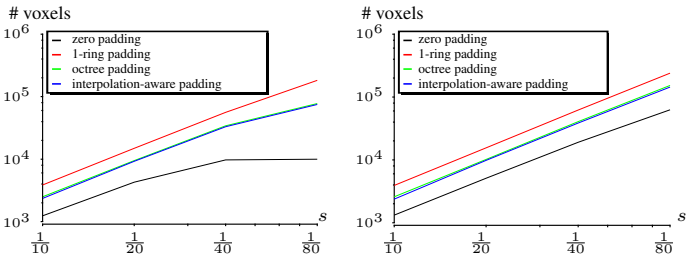


Figure 4: Statistics of the number of occupied voxels by different padding schemes under different voxel sizes. The voxel size s is selected from $\{\frac{1}{10}, \frac{1}{20}, \frac{1}{40}, \frac{1}{80}\}$. The bounding box size of the input Bunny model is 2. The numbers of the input points are 10000 (left) and 100000 (right), respectively.

not guaranteed to exist. We regard this padding scheme as one of the baselines for comparison.

N-ring padding The most direct way of padding voxels for interpolation is to add all the empty voxels whose Manhattan Distance from their centers to the center of one of the non-empty voxels is not greater than N, where N is a positive integer. In this way, all the voxels used in the Eq. 1 exist. Fig. 1-(b) illustrates 1-ring padding in 2D and 3D.

Interpolation-aware padding To interpolate CNN features on a 3D point $\mathbf{p} = (x, y, z)$ which is inside a non-empty voxel $\mathbf{v}$ using Eq. (1), we pad all the empty voxels involved in the interpolation into the CNN computation. We assume the 3D bounding box of the input is specified and the corner with the minimal x-, y-, z- coordinates is denoted by $\mathbf{p}_o = (x_o, y_o, z_o)$. The voxel indices of all the eight voxels involved by the interpolation are computed as follows:

$$I_x := \lfloor \tfrac{x-x_o}{s} + o_x \rfloor,\ I_y := \lfloor \tfrac{y-y_o}{s} + o_y \rfloor,\ I_z := \lfloor \tfrac{z-z_o}{s} + o_z \rfloor,$$

where $\lfloor \cdot \rfloor$ is the floor function, s is the voxel size (*i.e.*, voxel edge length), and $o_x, o_y, o_z \in \{0.5, -0.5\}$. The center locations of these voxels are $\mathbf{p}_o + s(I_x + 0.5, I_y + 0.5, I_z + 0.5)$. Fig. 3 shows a 2D illustration of interpolation-aware padding where two 2D points appear inside pixel v. It can be seen that the number of padded pixels/voxels depend on the point location, which can vary from 0 to 26 in 3D. Due to the construction of interpolation-aware padding, the set of padded voxels is a subset of voxels of 1-ring padding.

Complexity statistics of sparse padding We denote M as the non-empty voxel number, the worst-case memory and computational complexity of octree padding, 1-ring padding, and our interpolation-aware padding are $\mathcal{O}(8 \cdot M)$, $\mathcal{O}(27 \cdot M)$, and $\mathcal{O}(27 \cdot M)$, respectively. The worst case happens only when the average point spacing of the input point cloud is larger than $2s$, *i.e.*, all the non-empty voxels are disjointed with each other. For interpolation-aware padding, the worst-case further requires that eight corner regions of every non-empty voxel contain points, and the padding appears in all directions. The former extreme situation may exist in using high-resolution sparse voxels for the point cloud from LiDAR scans, and the latter case for interpolation-aware padding does not happen on real inputs. For point sets from 3D objects and indoor scenes, the commonly-used sparse voxel size is bigger, less memory cost and computation efforts can be achieved.

In Fig. 4, we calculate the total voxel number after applying different padding schemes on an example, whose input points are uniformly sampled from a 3D Bunny model. The number of padded voxels by interpolation-aware padding is quite similar to the one by octree padding, which is about 6.8 times the number of non-empty voxels when the voxels are highly sparse (10000 points with $s = 1/80$); while it is just 1.4 times when the sparsity is moderate.

3.2. Network design

We use the U-Net structure with five levels of domain resolution as shown in the left panel of Fig. 5, which consists of multiple sparse-convolution-based residual blocks and skip connections. The point feature is interpolated from the output voxel features at the finest level. Two additional FC layers are appended after the interpolated feature.

The network can also interpolate point-wise features from coarse-resolution voxel features. An example is shown in the right panel of Fig. 5. In the decoder of U-Net, the ResNet blocks operating on the two highest resolutions are dropped, which may further reduce the computation and memory cost while maintaining similar performances. Compared with U-Net network with the nearest neighbor interpolation, the output voxel feature has to be at the finest resolution, otherwise, multiple points in the same voxel can not be distinguished, which may harm the performance.

The proposed interpolation-aware padding can be added to all resolution levels of the sparse voxel grids, which achieves the best performance according to our experiments. However, according to the complexity analysis in the previous section, the padding on the finest resolution consumes the most resources. So we add the padding to the level cor-

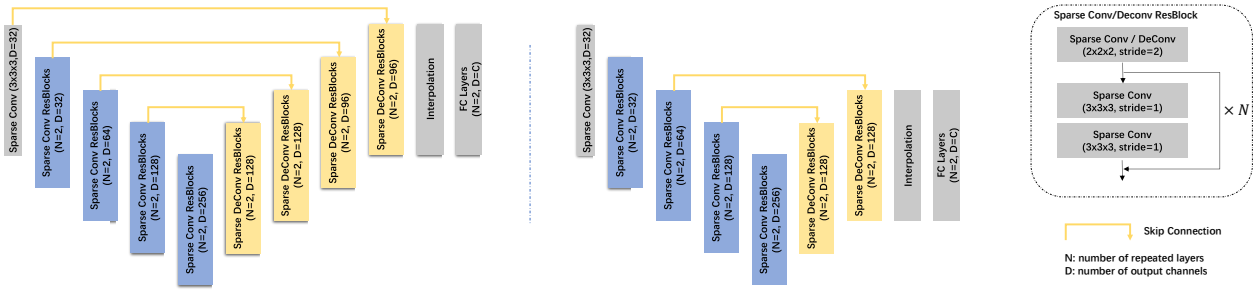


Figure 5: Sparse-voxel based U-Net structure. On the left panel, the resolution of output voxels is the same as the input; on the right panel, the resolution of output voxels is coarser.

responding to the output voxel resolution only for reducing computation and memory cost.

4. Experimental Analysis

In this section, we choose the fine-grained part segmentation task on the PartNet dataset to evaluate our interpolation and padding schemes, and the network structures, with highlights on our contributions. By default, all experiments were tested on a Linux server with Intel Core I7-6850K CPU (3.6 GHz) and a GeForce GTX 2080 Ti GPU (11 GB memory). The implementation is built upon the sparse voxel CNNs provided by [23].

Dataset We pick four categories (Chair, Lamp, Storage furniture, Table) of PartNet as the benchmark, each of which has at least 1000 shapes and contains three levels of semantic labels, and we follow the original data splitting for training and test. The input data (10000 points) is normalized to fit inside a unit box, and the finest grid resolution in the network is set to 64^3. The network outputs the finest-grained semantic label (level-3) at points.

Experiment setting We train the U-Net in Fig. 5 with an output resolution 64^3 to evaluate different interpolation and padding schemes. We name the U-Net integrated with one of padding schemes: zero padding, 1-ring padding, octree padding, and interpolation-aware padding by ZERO, RING, OCTREE, and INTERP, respectively. And we denote the nearest neighbor and trilinear interpolation as NEAR and LINEAR. We train the U-Net with one of the above settings for each shape category and evaluate the segmentation quality on the test data with the part mean IOU metric [17]. For a fair comparison, each network was trained three times and all the networks are initialized with the same set of parameters each time. We use the SGD optimizer with a learning rate of 0.1 and decay 0.1 at 1/2 and 3/4 of the max epoch. The batch size is set to 24. We report the averaged metric with mean deviation in Tab. 1. The IoU metrics on individual categories are reported in the supplemental material.

Group	Pad.	Interp.	S_{out}	mIoU	Time	Mem.
(1)	ZERO	NEAR	64^3	40.5 ± 0.2	382	1471
	OCTREE	NEAR	64^3	40.0 ± 0.2	568	2560
	RING	NEAR	64^3	$\mathbf{40.9} \pm 0.0$	929	3790
	INTERP	NEAR	64^3	40.6 ± 0.1	623	2707
(2)	ZERO	LINEAR	64^3	41.5 ± 0.0	398	1622
	OCTREE	LINEAR	64^3	41.4 ± 0.0	591	2610
	RING	LINEAR	64^3	$\mathbf{42.7} \pm 0.3$	965	3744
	INTERP	LINEAR	64^3	42.3 ± 0.3	651	2758
(3)	ZERO	NEAR	32^3	38.1 ± 0.2	248	902
	INTERP	LINEAR	32^3	$\mathbf{40.1} \pm 0.1$	366	1646

Table 1: Quality statistics of fine-grained segmentation on four PartNet categories under different settings. *Pad.* is the sparse padding type, *Int.* is the sparse interpolation type, *mIoU* is the average part IoU, *Time* is the average time in milliseconds of a single forward and backward propagation on a batch (16 objects), and *Mem.* is the average GPU memory (in Megabyte) occupied by a batch. The experiments are grouped for analysis.

Performance improvement by the interpolation By comparing the results between Group (1) and (2) in Tab. 1, we can see that the performances with the trilinear interpolation (LINEAR) with different padding schemes are consistently better than the nearest neighbor interpolation (NEAR), which is understandable since different points inside one voxel can be distinguished via the trilinear interpolation.

By comparing the padding schemes within Group (2), we can see that the U-Net with 1-ring padding (RING) achieves the best performance, however, it consumes the largest GPU memory in the runtime and requires a longer execution time. The octree padding is no better than zero padding. The U-Net with our interpolation-aware padding (INTERP) and trilinear interpolation (LINEAR) achieves a good balance of accuracy gain and memory cost. The interpolation-aware padding and 1-ring padding enable a well-defined trilinear interpolation and the padded voxels contain meaningful features after training, whereas zero features are used for empty voxels for octree padding and zero padding. To verify that the learned features on padded voxels are different from zero or some constant, we visualize the output features of the U-Net with our interpolation-aware padding. We extract the output

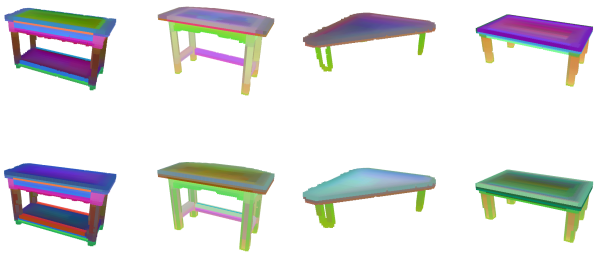


Figure 6: Visualization of learned features on non-empty voxels (top row) and padded voxels (bottom row).

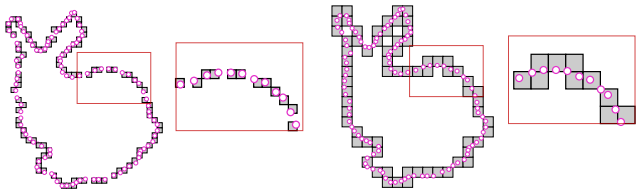


Figure 7: Non-empty pixels of a bunny shape under a high resolution (left) and a coarse resolution (right). The sparse voxels are disjointed when the resolution is high.

features on all the voxels and map them to the RGB domain via dimension reduction (T-SNE). Fig. 6-upper row shows the color map on the non-empty voxels, while Fig. 6-lower row shows the color map on the padded voxels. The features of the padded voxels are neither identical in a single object nor identical across different shapes.

The contributions of sparse padding By comparing the results within Group (1) in Tab. 1, we can see that the padding itself can also help to improve the performance, even without the trilinear interpolation. We suspect that the reason is that padding around the non-empty voxels helps information propagation, especially for very sparse inputs. Fig. 7-left illustrate 2D sparse non-empty pixels of a shape under a high resolution. For the disjointed non-empty pixels whose locations are close to each other (see the pixels around the gap region), the feature maps associated with them cannot be propagated to each other via any 3×3 convolution. Effective information propagation either requires a large-kernel-size convolution or occurs in coarse-version sparse pixels in which non-empty pixels have more neighbors as shown in Fig. 7-right. Although the worst complexity of interpolation-aware padding is $\mathcal{O}(27 \cdot M)$, in real scenarios like the PartNet experiments with the resolution 64^3 the average complexity is $\mathcal{O}(1.95M)$ only, much lower than the worst case.

The improved network design We did experiments to validate the network which outputs coarse resolution features as shown on the right panel in Fig. 5. The results with an output resolution of 32^3 are reported in Group (3)

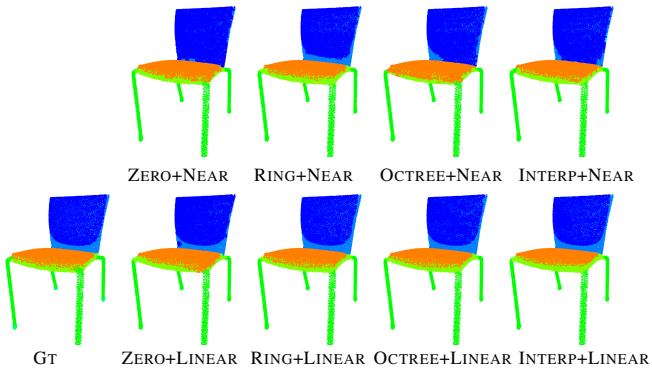


Figure 8: Visualization comparison of segmentation results under different padding and interpolation schemes. Different colors indicate different part labels.

in Tab. 1: compared with previous methods [26, 9, 3] with the zero-padding and the nearest neighbor interpolation, our performance is much better. Compared with the network with an output resolution 64^3 in Group (2), the memory and computation coast reduced by 40%. However, we notice that the overall mIoU is worse than Group (2), which is mainly caused by the Storage with an IoU decrease of 4.6 (more details can be found in the supplemental materials). In Sec. 5, we also tested the U-Net outputting coarse resolution features in ScanNet segmentation (see Sec. 5.1) and KITTI segmentation (see Sec. 5.3), we find that in these experiments the mIoU is comparable or even better than the U-Net outputting fine resolution features.

Result visualization We visualize the segmentation results of a chair in the test dataset under different padding and interpolation schemes in Fig. 8. By comparing the results, we can see that the results with the trilinear interpolation (2nd row) are smoother and more faithful to the ground-truth than the results with the nearest interpolation (the first row). The results of RING and INTERP are always better than ZERO and OCTREE. Zero padding has the worst segmentation results.

Comparison with state-of-the-art In Tab. 2, we report the comparison with other state-of-the-art methods, including PointNet++ [20], PointCNN [12], and O-CNN [28]. It can be seen that our results are much better than others in all four shape categories. This is mostly caused by the finer segmentation results at boundaries and fine-grained regions, as we can see from Fig. 8. Our padding scheme improves the representation power of the interpolated point features and helps to distinguish segmentation region boundaries. Here we should mention that the employed O-CNN [28] can be regarded as sparse-voxel-based CNNs with the trilinear interpolation and the octree padding in *all* voxel resolutions.

Method	mIoU	Chair	Lamp	Stora.	Table
PointNet++ [20]	34.7	39.2	25.3	40.5	33.9
PointCNN [12]	33.7	43.9	20.1	49.4	21.3
O-CNN [28]	41.7	46.8	28.5	53.8	37.7
Ours	**42.7**	**47.6**	**29.4**	**54.8**	**38.9**

Table 2: Comparison with state-of-the-art methods on PartNet.

For a fair comparison, we also use our interpolation-aware padding in all voxel resolutions. Note that the mIoU of octree padding and our interpolation-aware padding increases from 41.4 to 41.7, and 42.3 to 42.7 respectively, compared with the network with a single resolution padding as shown in Group (2) of Tab. 1.

5. Comparisons

We further evaluate and compare the performance of our method in other 3D segmentation and detection tasks with the state-of-the-art methods. Here we use the interpolation-aware padding for comparison, and the 1-ring padding is not tested as the network with 1-ring padding for on ScanNet and KITTI datasets runs out of memory even on a V100 GPU with 32 GB memory.

5.1. Semantic segmentation on ScanNet

Dataset The ScanNet dataset [6] contains 1.5k indoor scenes. We follow [3] to conduct the same data split and augmentation and feed the whole scene to the network without cropping.

Network structure We use the same U-Net with five levels of domain resolution as employed by MinkNet [3]. The network structure is same to Fig. 5, but with more residual blocks (the numbers of repeated resblocks are [2,3,4,6,2,2,2,2]). The only difference is that MinkNet uses the zero-padding and the nearest interpolation, while our network uses our interpolation-aware padding and trilinear interpolation.

Experiment setting Similar to the setup of MinkNet [3], the voxel size at the finest level in the encoder is set to 2 cm and the batch size is 9. The input signal at each voxel is a 3-channel RGB color with 1 additional channel indicating whether the voxel is created by padding or not (0 for the padded voxels and 1 for the original non-empty voxels). The voxel size used for point feature interpolation in the decoder is denoted by S_{out}, and we experiment different S_{out} from 2 cm to 8 cm for evaluating the effect of feature interpolation. For our network with feature interpolation, we only pad voxels at the level of S_{out}. The training scheme is the same as the one used in [4]: the optimizer is SGD and the learning rate is adjusted from 0.1 with the polynomial learning rate policy. We train all models for 600 epochs. All the results are evaluated on the validation set of ScanNet, and are presented in Tab. 3.

Network	Pad.	Interp.	S_{out}	mIOU	Mem.
MinkNet [3]	ZERO	NEAR	2 cm	72.2 ± 0.3	3514
Ours	INTERP	LINEAR	2 cm	$\mathbf{72.8} \pm 0.2$	6829
MinkNet [3]	ZERO	NEAR	8 cm	70.4 ± 0.1	2184
Ours	INTERP	LINEAR	8 cm	$\mathbf{72.4} \pm 0.2$	2986

Table 3: Quality statistics of semantic segmentation on ScanNet val set. *mIoU* is the average IOU of all the classes.

Result analysis The comparison in Tab 3 further confirms the observations in Sec. 4: our network with feature interpolation achieves higher accuracy than MinkNet [3]. We also find that our network with feature interpolation at a coarser resolution (8 cm) is comparable to MinkNet with high resolution (2 cm) output: 72.4 vs. 72.2, while the runtime memory consumption is smaller: 2.9M vs. 3.5M.

5.2. 3D Object detection on ScanNet

Dataset The ScanNet dataset contains instance segmentation labels of indoor scenes. With these labels, the bounding box of each object instance can be calculated. We follow the data preparation in the work of VoteNet [18], and use mAP@0.25 and mAP@0.5 as the evaluation metric.

Network structure We choose the original VoteNet as the baseline which uses PointNet++ [20] as the backbone to extract seed point features. We also replace PointNet++ with the 5-level U-Net mentioned in Sec. 5.1 and keep all other structures unchanged. The seed features are extracted from the third level of the decoder in the U-Net to mimic the features extracted from PointNet++'s SA2 layer which is used in the original VoteNet.

Experiment setting For U-Net, we set the voxel size of the finest level in the encoder to 2 cm, and the voxel size of the third level of the decoder S_{out} is 8 cm. The batch size for all the networks is set to 8. Point color and height are the input signal, and we evaluate the detection results on the validation set. The optimizer used in this task is Adam with a 0.001 initial learning rate. We train each model for 200 epochs. The learning rate decays 0.3 at 80, 120, 160 epochs.

Result analysis With the stronger backbone — sparse U-Net, the mAP@0.5 of object detection on ScanNet is increased by 3.2. And with the interpolated point feature and the interpolation-aware sparse padding, the performance

Network	Pad.	Int	mAP@0.25	mAP@0.5
VoteNet [18]	-	-	57.8 ± 0.6	34.7 ± 0.4
MinkNet [3]	ZERO	NEAR	58.7 ± 0.5	37.9 ± 0.6
Ours	INTERP	LINEAR	$\mathbf{60.7} \pm 0.8$	$\mathbf{41.4} \pm 0.6$

Table 4: Quality statistics of instance detection on ScanNet validation set. VoteNet [18] is based on PointNet++, we replace it with MinkNet [3] and report the results in the second row. The results of combining our interpolation-aware padding and interpolation scheme with MinkNet are shown in the last row.

(mAP@0.5) is increased by 6.7 from the baseline. Our network with the interpolation-aware sparse padding and the trilinear interpolation is also better than the zero-padding and the nearest neighbor interpolation, consistent with the previous segmentation experiments. Note that the sparse padding can be used as a plug-in in any method based on sparse voxel-based 3D convolution, we believe it can benefit other stronger baselines like 3D-MPA [7].

5.3. Semantic segmentation on KITTI Dataset

Dataset Semantic KITTI [1] contains large-scale outdoor scenes annotated with semantic labels based on the 22 sequences [8]. Each sequence contains thousands of point clouds acquired by LiDAR sensors, in which the point density is quite non-uniform even in a single scan. We follow the standard train-validation split and report the results on the validation set.

Network structure We plug our interpolation-aware padding and trilinear interpolation into two networks on this task: the U-Net structure in Fig. 5, which is the same as the one used for MinkNet [3], and the SPVCNN [23], which is composed of a low-resolution sparse-voxel-based U-Net and a high-resolution point-based branch. Our padding and interpolation scheme is combined with the voxel-based U-Net part of SPVCNN.

Experiment setting We use the same setting of [23]: the voxel size of the finest level in the encoder is set to $5\,\mathrm{cm}$ and the batch size is set to 2. The input signal contains a 3-channel point coordinate and a 1-channel LiDAR signal. Similar to the experiments in Sec. 5.1, we experiment our padded-version networks with interpolated features from $5\,\mathrm{cm}$ and $20\,\mathrm{cm}$. The training scheme is the same as the one used in [3]: the optimizer is SGD and the learning rate starts from 0.24 and is adjusted by the cosine scheduler with warm-up. We train all models for 15 epochs.

Result analysis By using our interpolation-aware padding and trilinear interpolation as a plugin of MinkNet [5] and SPVCNN [23], we can see the performance increases consistently as shown in Tab. 5. It is also interesting to find that our network with $S_{out} = 20\,\mathrm{cm}$ performs better than the settings with $S_{out} = 5\,\mathrm{cm}$. We speculate that the interpolation on coarser voxels may be suitable to handle extremely non-uniform distributed points. We also observe that under the zero padding setting, MinkNet with trilinear interpolation is not superior to the one with the nearest-neighbor interpolation, as shown in the first two rows of Tab. 5. The phenomenon shows that a proper padding scheme like our interpolation-aware padding is crucial for the trilinear interpolation.

Network	Pad	Int	$\mathbf{S_{out}}$	mIOU
MinkNet [3]	ZERO	NEAR	$5\,\mathrm{cm}$	61.9 ± 0.3
MinkNet [3]	ZERO	LINEAR	$5\,\mathrm{cm}$	61.4 ± 0.1
Ours	INTERP	LINEAR	$5\,\mathrm{cm}$	$\mathbf{63.5} \pm 0.3$
MinkNet [3]	ZERO	NEAR	$20\,\mathrm{cm}$	61.5 ± 0.3
Ours	INTERP	LINEAR	$20\,\mathrm{cm}$	$\mathbf{63.9} \pm 0.4$
SPVCNN [23]	ZERO	LINEAR	$5\,\mathrm{cm}$	62.9 ± 0.7
Ours	INTERP	LINEAR	$5\,\mathrm{cm}$	$\mathbf{63.2} \pm 0.3$
SPVCNN [23]	ZERO	LINEAR	$20\,\mathrm{cm}$	62.8 ± 0.2
Ours	INTERP	LINEAR	$20\,\mathrm{cm}$	$\mathbf{63.7} \pm 0.1$

Table 5: Quality statistics of semantic segmentation on KITTI dataset. In each panel between horizontal lines, *Ours* means using the same network as upper lines, while adopting our interpolation-aware padding INTERP and trilinear interpolation LINEAR.

6. Conclusion

In this work, we present an interpolation-aware padding that enables well-defined interpolation for sparse-voxel-based CNNs. The efficacy of our padding schemes and improved networks is well demonstrated on 3D segmentation and 3D detection tasks. In the future, we would like to explore sparse padding and related interpolation operations in the following directions.

N-dimensional sparse data Currently, our study is mainly performed in 3D, while all the sparse padding schemes are generalizable to any dimension. However, the increased runtime memory would be a severe side effect in high dimensions.

Non-regular interpolation scheme As the increased feature maps from the padded voxels are the main memory bottleneck, it would be interesting to employ non-regular interpolation schemes like RBF (radial basis function interpolation) to avoid pad empty voxels while enjoying the flexibility brought by feature interpolation. The parameters of RBF are possibly learned during the training.

References

[1] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *ICCV*, 2019. 1, 2, 8

[2] Andrew Brock, Theodore Lim, J.M. Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. In *3D deep learning workshop (NIPS)*, 2016. 1, 2

[3] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D spatio-temporal ConvNets: Minkowski convolutional neural networks. In *CVPR*, 2019. 1, 2, 3, 6, 7, 8

[4] Christopher Choy, Junha Lee, Rene Ranftl, Jaesik Park, and Vladlen Koltun. High-dimensional convolutional networks for geometric pattern recognition. In *CVPR*, 2020. 7

[5] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *ECCV*, 2016. 2, 8

[6] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *CVPR*, 2017. 1, 2, 7

[7] Francis Engelmann, Martin Bokeloh, Alireza Fathi, Bastian Leibe, and Matthias Nießner. 3D-MPA: Multi Proposal Aggregation for 3D Semantic Instance Segmentation. In *CVPR*, 2020. 8

[8] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *CVPR*, pages 3354–3361, 2012. 8

[9] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *CVPR*, 2018. 1, 2, 3, 6

[10] Zixuan Huang, Junming Fan, Shenggan Cheng, Shuai Yi, Xiaogang Wang, and Hongsheng Li. HMS-Net: Hierarchical multi-scale sparsity-invariant network for sparse depth completion. *IEEE Trans. Image Process.*, 29, 2020. 2

[11] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D mesh renderer. In *CVPR*, 2018. 1

[12] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on X-transformed points. In *NeurIPS*, 2018. 1, 6, 7

[13] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *ECCV*, 2018. 2

[14] Guilin Liu, Kevin J. Shih, Ting-Chun Wang, Fitsum A. Reda, Karan Sapra, Zhiding Yu, Andrew Tao, and Bryan Catanzaro. Partial convolution based padding. Technical report, NVIDIA Corporation, 2018. 2

[15] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3D point cloud understanding. In *ICCV*, 2019. 3

[16] D. Maturana and S. Scherer. VoxNet: A 3D convolutional neural network for real-time object recognition. In *International Conference on Intelligent Robots and Systems (IROS)*, 2015. 2

[17] Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *CVPR*, 2019. 2, 5

[18] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep Hough Voting for 3D Object Detection in Point Clouds. In *ICCV*, 2019. 1, 7, 8

[19] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*, 2017. 1

[20] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017. 6, 7

[21] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. OctNet: Learning deep 3D representations at high resolutions. In *CVPR*, 2017. 1, 2

[22] Tianjia Shao, Yin Yang, Yanlin Weng, Qiming Hou, and Kun Zhou. H-CNN: Spatial hashing based CNN for 3D shape analysis. *IEEE Trans. Vis. Comput. Graphics*, 26(7), 2020. 1, 2, 3

[23] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3D architectures with sparse point-voxel convolution. In *ECCV*, 2020. 1, 3, 5, 8

[24] Jonas Uhrig, Nick Schneider, Lukas Schneider, Thomas Brox, and Andreas Geiger. Sparsity invariant CNNs. In *Proc. Int. Conf. on 3D Vision (3DV)*, 2017. 2

[25] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2Mesh: Generating 3D mesh models from single RGB images. In *CVPR*, 2018. 1

[26] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Trans. Graph. (SIGGRAPH)*, 36(4), 2017. 1, 2, 3, 6

[27] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive O-CNN: A patch-based deep representation of 3D shapes. *ACM Trans. Graph. (SIGGRAPH ASIA)*, 2018. 1

[28] Peng-Shuai Wang, Yu-Qi Yang, Qian-Fang Zou, Zhirong Wu, Yang Liu, and Xin Tong. Unsupervised 3D learning for shape analysis via multiresolution instance discrimination. In *AAAI*, 2020. 1, 3, 6, 7

[29] Wiki. Trilinear interpolation — Wikipedia, the free encyclopedia, 2020. [Online; accessed 19-Jan-2021]. 3

[30] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NeurIPS*, 2016. 1, 2

[31] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shape modeling. In *CVPR*, 2015. 2