Computer Vision and Image Understanding 224 (2022) 103555

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

RocNet: Recursive octree network for efficient 3D processing

Juncheng Liu *, Steven Mills, Brendan McCane

Department of Computer Science, University of Otago, Dunedin, New Zealand

ARTICLE INFO

Communicated by Nikos Paragios

MSC:
41A05
41A10
65D05
65D17

Keywords:
Recursive
3D shape
Deep representation
Segmentation
Classification

ABSTRACT

We introduce a deep recursive octree network for general-purpose 3D voxel data processing. Our network compresses a voxel grid of any size down to a very small latent space in an autoencoder-like network. We show results for compressing 32^3, 64^3 and 128^3 grids down to just 80 floats in the latent space. We demonstrate the effectiveness and efficiency of our proposed method on several publicly available datasets with four experiments: 3D shape classification, 3D shape reconstruction, shape generation and semantic segmentation. Experimental results show that our algorithm maintains accuracy while consuming less memory with shorter training times compared to existing methods, especially in 3D reconstruction tasks.

1. Introduction

While neural networks achieve excellent performance in various tasks in 2D vision, how to effectively process 3D data by neural networks has recently attracted more attention in both the computer vision and computer graphics communities. Unlike the unified representation of 2D images as pixels, there are many formats widely used for 3D data such as point clouds (Qi et al., 2017a,b, 2018), volumetric voxels (Wu et al., 2016; Shen and Stamos, 2020), surface meshes (Groueix et al., 2018) and implicit surface (Park et al., 2019; Mildenhall et al., 2020). Each of these formats has its advantages and disadvantages. Point clouds represent 3D data by unstructured points without imposing topological constraints. However, they bring difficulties in subsequent applications such as rendering and space navigation. Surface meshes are most widely used in graphics due to efficient rendering and modeling flexibility, yet their inherent graph structure makes it difficult, though not impossible, to process via neural networks. 3D voxel grids are the straightforward generalization of 2D pixels and are a natural representation of 3D space. For certain tasks, such as navigation/exploration for robotics, using explicit 3D occupancy voxel representation makes it easier for subsequent tasks such as path-planning. But voxel grids are very memory intensive and memory requirements grow very fast as the size of the grid increases, making them difficult to apply to fine resolutions or large spaces.

In this paper we present RocNet: a recursive 3D autoencoder network that mimics the structure of an octree. As shown in Fig. 1, a voxel grid is first converted to a standard octree structure until each node is homogeneous or a maximum depth is reached. RocNet recursively merges octants starting from the octree leaves, until the root is processed. In this way the whole octree can be represented by a compact 1D feature vector. Since there are a large number of empty leaves in an octree, our algorithm saves large amount of memory and computational time. Compared to traditional 3D convolutional networks, 3D convolution is only performed within octants in our recursive network hence is less computationally intensive. Memory savings arise from the octree structure which adapts to the geometry of the scene, allowing large empty (or otherwise homogeneous) volumes to be represented compactly.

RocNet is an end-to-end autoencoder that is a recursive, discriminative, and generative network that is primarily aimed at compressing 3D voxel structures for representation, discrimination, and generation. The main contributions of this work are: this is the first method to explore the combination of volumetric octree and recursive networks; near state-of-the-art compression ratios; near state-of-the-art recognition on ModelNet40; state-of-the-art computational and memory consumption efficiency; state-of-the-art reconstruction accuracy on ShapeNetCorev2 database.

This is an extended version of our conference paper (Liu et al., 2020). The main extensions are two-fold: we reinforce the generation ability of RocNet by adding an adversarial training term which further regularizes the latent shape space; With minor modifications

* Corresponding author.
E-mail address: juncheng.liu@otago.ac.nz (J. Liu).

https://doi.org/10.1016/j.cviu.2022.103555
Received 19 November 2021; Received in revised form 26 August 2022; Accepted 30 August 2022
Available online 6 September 2022

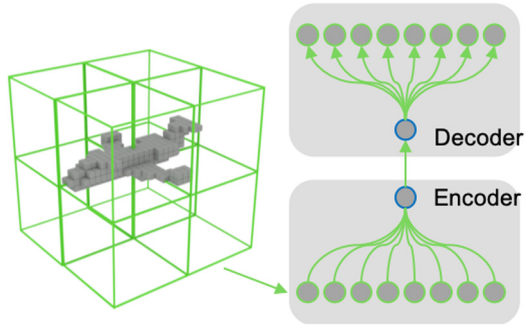


Fig. 1. RocNet autoencoder structure. Octants are merged recursively in a bottom-up manner until the root node is encoded. Decoder recursively decodes nodes until leaves are recovered.

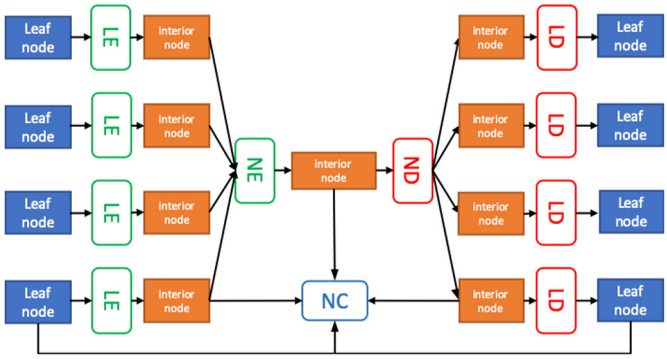


Fig. 2. RocNet autoencoder structure. LE: leaf encoder. NE: node encoder. ND: node decoder. LD: leaf decoder. NC: node classifier.

in the structure, we apply RocNet to a semantic segmentation task and it is proven effective on manually-labeled ShapeNetCore dataset. Compared to the previous version, we further demonstrate that the proposed method is applicable and effective to most of tasks of 3D voxel processing.

2. Related work

The idea of using octrees to efficiently process sparse 3D data has been intensively explored in deep learning in recent years. These methods can be roughly classified into two groups: discriminative and generative networks.

Discriminative octree networks. Discriminative networks are designed to accomplish recognition tasks such as classification and retrieval. O-CNN (Wang et al., 2017) uses an octree-based 3D convolutional network by using fast shuffled key and hash table search. Their method significantly reduces the computational time and storage requirements for 3D convolution by restricting computations to occupied octants only. OctNet (Riegler et al., 2017a) uses an alternative octree-based convolutional network where a more efficient convolution is defined for octants. A major difference between RocNet and the other methods is RocNet performs convolution only within octants. Convolution is not performed across octants hence is more computationally efficient.

Generative octree networks. Generative octree networks are able to generate 3D data in an octree-like manner. Adaptive O-CNN (Wang et al., 2018) implements an advanced version of O-CNN which is able to generate models in adaptive patches. HSP (Häne et al., 2017) predicts high resolution voxel grids from images by using an octree up-convolutional decoder architecture. OctNetFusion (Riegler et al., 2017b) is the generative version of OctNet, which reconstructs dense 3D data from multiple depth images by predicting the partitioning of the 3D space. Similar to our method, OGN (Tatarchenko et al., 2017) also generates octrees, but their method is aimed at up-sampling from a coarse regular voxel grid and thus focuses on decoding only. In contrast, we seek to form an efficient latent space for storing grids of arbitrary size, where the latent space is not restricted to be grid-like.

Recursive networks. Our method is also related to recursive networks which were originally proposed as an effective tool for grammar tree parsing in natural language (Socher et al., 2011). In computer graphics, GRASS (Li et al., 2017) employed recursive networks to implement a generative shape structure autoencoder. Their work focuses specifically on object structures and makes explicit use of object symmetries. RocNet focuses on volumetric representation and can also be used for generic 3D representations such as rooms or buildings. Another successful application of recursive networks is k-d networks (Klokov and Lempitsky, 2017) which use a k-d tree to hierarchically process 3D point clouds, whereas we focus on voxel representations.

3. Recursive octree networks

In this section we will describe the network architecture by which the octrees are encoded and decoded. The overall structure of the proposed network is shown in Fig. 2. The network recursively merges 8 child nodes to form their parent node until the root node is encoded. The insight is that the recursive network easily fits the hierarchical nature of octrees.

Our network consists of 7 components: leaf, node, and tree encoders; leaf, node, and tree decoders; and a node classifier. The octree is encoded from the leaves up. The *leaf encoder* encodes the leaves and produces a multi-channel 3D code for each leaf. For each non-leaf node, the *node encoder* combines codes from each of its children until the root is reached, again producing a multi-channel 3D code for each node. Finally the *tree encoder* converts a multi-channel 3D code into a compact 1D feature vector. To produce an octree from a compact code, the corresponding *decoders* reverse the encoding process. The only difference is that we need to determine if a given node is an internal node or a leaf and the *node classifier* is used to make that determination. We describe the octree representation and each component in more detail in the following sections.

3.1. Octree representation

An octree decomposes a 3D space recursively until it contains homogeneous content (all voxels inside a leaf should be consistently occupied or empty) or a given maximum depth is reached. In the proposed algorithm, we adopt the octree representation as the input format of our neural network.

An example of an octree structure is presented in Fig. 3. For simplicity, we illustrate a quad-tree on a 2D grid instead of an actual octree on a 3D grid. We define 4 types of octants: empty leaf (unoccupied leaf octants), full leaf (fully occupied leaf octants), mixed leaf (partially occupied leaf octants) and interior node (non-leaf octants) as shown in Fig. 3(f). An octree can be therefore represented as two separate parts: tree topology and its mixed leaves. Empty and full leaves do not need storage and can be directly recovered from tree topology.

Tree topology representations. To encode octree topology, we use a post-order Depth-First Search (DFS) ordering. As shown in Fig. 3(d), the octants of a 3 layer octree are serialized by the post-order DFS: it starts at the root node and goes as far as it can down its child nodes from left to right, then visit itself and backtracks.

Mixed leaf representations. We collect all mixed leaf nodes in an octree in the order defined above, forming a 4D binary matrix. In the example shown in Fig. 3(e), the maximum depth is 1 and two mixed leaf nodes are extracted. To clearly demonstrate the advantages of our proposed structure, we simply use binary occupancy encoding across all our experiments. However, other representations such as color values, real-valued occupancy probability, or surface normals could also be easily integrated into our algorithm in this step.

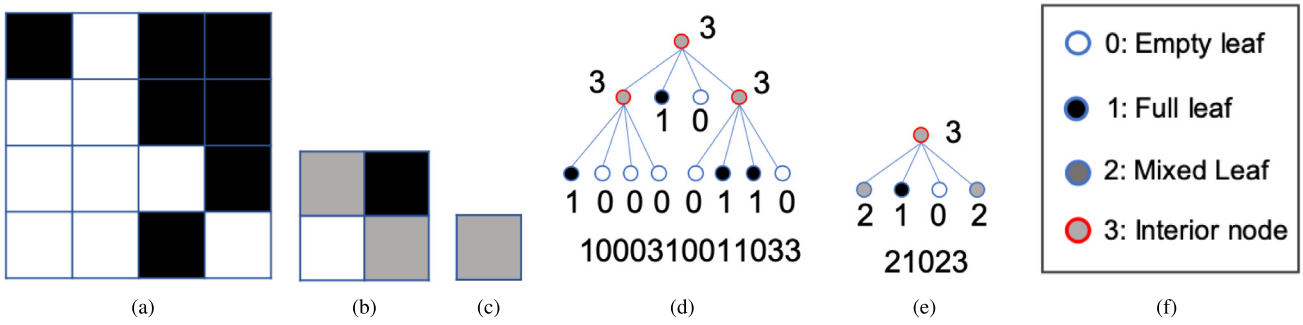


Fig. 3. Example of an octree, although a quad-tree is shown for simplicity. (a)–(c) depth 2 to 0 (root) of an occupancy octree. Black square indicates occupied octants and white square indicates empty ones. Mixed nodes are represented as gray squares. (d) octree structure of maximum depth 2 and its corresponding structure code. (e) octree structure of maximum depth 1 and its corresponding structure code. (f) 4 defined node types.

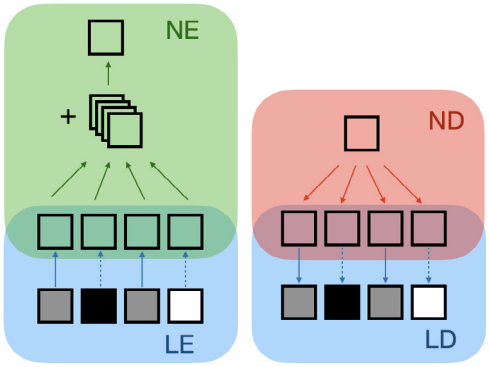


Fig. 4. Node encoder and decoder. Solid arrows are 3D convolution and batch normalization. Dotted arrows are empty/full leaf skips. A quad-tree is shown here for simplicity.

3.2. Recursive octree encoder

Generally speaking, the encoders encode leaf nodes, interior nodes, and eventually the whole octree into features as shown in Fig. 4(a).

Leaf encoder. Explicit leaf nodes need to be converted into recognizable features before being fed into the recursive network. In comparison with a traditional recursive autoencoder (Socher, 2014) (RAE) that encodes nodes into 1D features, our method encodes a leaf node into a 4D feature with fixed dimensions using several 3D convolutional layers of kernel size 4 and stride 2. The number of channels of feature maps is set to 16, 32 and 64. We use the exponential linear unit (Clevert et al., 2015) (ELU) function ($f : x \in \mathcal{R} \mapsto \max(0, x) + \min(0, e^x - 1)$) as the activation function and add a batch normalization (BN) layer after each 3D convolution layer. Our network is free of a pooling layer. Therefore the sequence of a leaf encoder is several layers of "3DConv + BN + ELU". The leaf encoder therefore is a mapping: $\mathcal{B}^{k\times k\times k} \mapsto \mathcal{R}^{64\times 4\times 4\times 4}$, where $\mathcal{B} = \{0, 1\}$, k is the leaf resolution (we set $k = 32$ for voxels with a resolution higher than 32^3 and $k = 16$ for 32^3), and 64 is the number of output channels. Since there are a large number of empty and full leaf nodes, applying 3D convolution to them is unnecessary and time-consuming, hence we pass these leaves directly to the next stage without applying 3D convolutions. We call these "leaf skips" as shown in Figs. 4(a) and 4(b).

Node encoder. We first lift the number of channels of child nodes from 64 to 128 by a $1 \times 1 \times 1$ sized kernel 3D convolutional layer denoted as $\phi : \mathcal{R}^{64\times 4\times 4\times 4} \mapsto \mathcal{R}^{128\times 4\times 4\times 4}$ and then merge the nodes into an interior parent node. The inverse of ϕ, denoted as ψ, is applied to the parent node which maps number of channels back to 64. We use "additive merging" for all encoders of this kind, that is, every child node is first applied with a 3D convolution followed by a batch normalization layer, and then added together to form the parent node:

$$F_p = \psi\left(\sum_{i=1}^{8} \mathrm{ELU}(\phi(F_i))\right), \tag{1}$$

$F_p : \mathcal{R}^{64\times 4\times 4\times 4} \mapsto \mathcal{R}^{64\times 4\times 4\times 4}$. This encoder is applied recursively until the root node is encoded. Note that node encoders for different depth do not share weights.

Tree encoder. After obtaining the feature of the root node, the whole octree is encoded to a single feature map of $64 \times 4 \times 4 \times 4$. The tree encoder is applied to flatten the feature map into a single 1D feature vector. We achieve this by simply employing a 3D convolutional layer with kernel size 4 and stride 1. The number of channels is set to the dimensions of the feature vector. The tree encoder then is a map: $\mathcal{R}^{64\times 4\times 4\times 4} \mapsto \mathcal{R}^{d_{out}}$. In our implementation d_{out} is set to 80 across all experiments.

3.3. Recursive octree decoder

The decoders simply reverse the above process using transposed convolutions instead of convolutions.

Tree decoder. This decoder converts a 1D feature back to a 4D feature by a transposed convolutional layer and a non-linear activation layer ELU. To transform feature vectors back to the above defined dimensions, we use a kernel of size 4 and stride 1.

Node decoder. After decoding the tree we obtain the 4D feature for the root node. We then apply the node decoder to it recursively until leaf nodes are decoded. This decoder consists of a 3D convolutional layer for the node itself and a convolutional layer for each its 8 child nodes followed by a batch normalization layer as shown in Fig. 4(b). Note that node decoders for different depth do not share weights.

Leaf decoder. When a node is recognized as a leaf it will be decoded by a leaf decoder which recovers the features back to explicit binary occupancy voxel grids. Note that leaf skips will also be applied in the leaf decoder for efficiency, i.e. only leaves recognized as mixed need to be decoded. This decoder consists of several stacked transposed 3D convolutional layers of kernel size 4 stride 2 and padding 1. This operation will enlarge the voxel size from 4 to k which is the dimension of explicit representation we used for leaves. All layers are followed by batch normalization layers and ELU activation with the exception of the final layer which is activated by a sigmoid non-linearity without batch normalization.

3.4. Node classifier

A node classifier is a 4-category classifier trained during the decoding process in order to recover the octree topology. Note that the classifier is not involved in encoding, as at that stage the type of each node is known. This classifier labels each node one of the 4

Table 1
Comparison of model size. Numbers of trainable parameters are shown. RocNet is followed by resolution and the leaf size.

	Size		Size
PointNet (Garcia-Garcia et al., 2016)	80M	RocNet-64–32	1.42M
3DShapeNets (Wu et al., 2015)	38M	RocNet-128–32	1.70M
VoxNet (Maturana and Scherer, 2015)	0.92M	RocNet-256–32	1.98M
LightNet (Zhi et al., 2018)	0.3M	RocNet-512–32	2.25M
VRN-ensemble (Brock et al., 2016)	90M	RocNet-1024–32	2.53M
FusionNet (Hegde and Zadeh, 2016)	118M	RocNet-2048–32	2.80M

aforementioned node types. It has the same layers as tree encoder except for an additional fully-connected layer. We use cross-entropy as the loss function. The node classifier is disabled during training and is only used for prediction in the testing stage. The predicted label is used to decide whether to stop decoding (empty/full leaf) or to decode using leaf decoder (mixed leaf) or node decoder (interior node).

3.5. Loss function

The loss function in our algorithm has two separate parts: the node labeling loss, $\mathcal{L}_l$ and the leaf reconstruction loss, $\mathcal{L}_r$:

$$\mathcal{L} = \mathcal{L}_l + \mathcal{L}_r. \tag{2}$$

Node labeling loss. We use cross-entropy loss given by node classifier as the node labeling loss:

$$\mathcal{L}_l = -\sum_{i}^{4} c_i \log(s_i), \tag{3}$$

where c_i and s_i are the ground-truth and node classifier score for 4 node types. The node label loss of an octree is the sum of label losses of all nodes.

Reconstruction loss. Decoded leaves are used to calculate the reconstruction loss and we use a weighted binary cross-entropy as the loss function:

$$\mathcal{L}_r = -\alpha t \log(o) - (1-t)\log(1-o), \tag{4}$$

where $t \in \{0, 1\}$ is the ground-truth occupancy value and $o \in (0, 1)$ is the output of the leaf decoder for each individual voxel. The weight parameter α is employed because the number of empty voxels is usually much larger than occupied ones. In most cases, α should be larger than 1 for elimination of false negatives. In our implementation we use $\alpha = 5$ across all experiments. The reconstruction loss of an octree is the sum of the reconstruction losses across all its mixed leaves.

Algorithm 1 RocNet node encoder and decoder

```
procedure ENCODING(NODE)
    if node.is_leaf() then return LeafEncoder(node)
    else
        for k ← 1 to 8 do
            child ← child + Encoding(node.get_child(k))
        return ψ(child)
procedure DECODING(NODE)
    if node.pred_leaf() then LeafDecoder(node)
    else
        for k ← 1 to 8 do
            Decoding(node.get_child(k))
```

3.6. Complexity analysis

In this section we analyze the model size as well as the computational time and space complexity of our proposed method both analytically and empirically.

Generally speaking, our method benefits from the recursive structure and hence can be regarded as a lightweight and scalable model. With the increase of input voxel resolution, the number of trainable parameters of our model increases linearly with respect to the logarithm of input/leaf resolutions: $\mathcal{O}(\log(N/k))$ where N is the input resolution and k is the leaf size (both should be a power of 2). Table 1 shows the model size of our method (right column) and other comparison methods (left column). Since all the other methods were designed specifically for classification, we show the number of trainable parameters of the encoder part of our model for a fair comparison. The total size of our model is approximately twice that shown as the encoder and decoder have roughly the same architecture. For our method we present models for 6 different resolutions. Each method is labeled as RocNet-$N-k$ and k is set to 32. RocNet-256-32, for instance, accommodates an input voxel grid of size $256 \times 256 \times 256$. The actual number of trainable parameters shown in Table 1 coincides with our analysis. Among the comparison methods, VoxNet (Maturana and Scherer, 2015) and LightNet (Zhi et al., 2018) are also lightweight yet their models only consider a fixed voxel grid of $32 \times 32 \times 32$ as they were designed for classification. This resolution should be adequate for a classification task. For more resolution-intensive tasks such as reconstruction and generation, it will lead to severe artifacts and imperfection.

The space and time complexity of our method are $\mathcal{O}((N/k)^3)$ and $\mathcal{O}(\log(N/k))$ respectively. The space complexity roughly depends on the number of mixed leaves which, in worst case, is $(N/k)^3$. However, the mixed leaves usually make up only a very small population in all leaves. Therefore our method enjoys a moderate memory consumption even for $256 \times 256 \times 256$ volumes, which easily fits on a modern GPU. Since the nodes within the same depth can be processed in parallel, the computational time for an octree is related to its depth, which is $3\log_8(N/k)$.

4. Experiments and comparisons

4.1. Experiments setup

We evaluate RocNet on four representative tasks: 3D shape classification, 3D shape reconstruction, 3D shape generation, and 3D semantic segmentation. The implementation of our algorithm is based on PyTorch (Paszke et al., 2019) and TorchFold (Polosukhin and Zavershynskyi, 2018) for dynamic batching. All experiments were done on a server with Intel Core i7-6700K CPU (4.00 GHz) and a GeForce GTX Titan X GPU (12 GB memory). We set leaf size $k = 16$ for voxel size $N = 32$ and $k = 32$ for all the other input resolutions across all experiments. To clearly demonstrate the advantages of our proposed method, we simply use binary occupancy voxels as input in all experiments. However, the performance can be further improved by employing a more informative input format such as surface normal and truncated signed distance function. Variations of input formats can be easily integrated into our network.

4.2. 3D shape classification

Dataset. We perform the classification task on the ModelNet40 dataset (Wu et al., 2015). This dataset contains 12,311 labeled CAD meshes from 40 categories. It is split into training (9843) and testing (2468) sets and the training set is augmented by rotating each sample 12 times along its upright axis uniformly. Since the original representation of these 3D shapes is a triangular mesh, to use our proposed method, the mesh needs to be converted to 3D voxel grids by voxelization. We generate their voxel representations at four different resolutions for each of the samples: 32, 64, 128, and 256.

Network architecture. We simply use the encoder architecture as feature extractor followed by an additional fully-connected layer, a dropout layer and a softmax layer. The input of the fully-connected layer is the output of the tree encoder defined in Section 3.2.

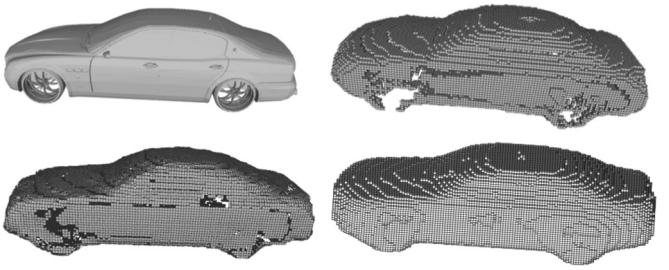


Fig. 5. Qualitative comparisons. Top-left: original model. Top-right: binary O-CNN reconstruction. Bottom-left: patch-based O-CNN and bottom-right: our result[1]. 128^3 is used for all three methods. Our RocNet has fewer missing regions than the other two methods.

Results and discussion. The classification results are shown in Table 2. We compare our method with 4 other alternatives: 3DShapeNets (Wu et al., 2015), VoxNet (Maturana and Scherer, 2015), Geometry image (Sinha et al., 2016) and O-CNN (Wang et al., 2017), all of which are voxel-based methods with the exception of Geometry image.

Similar to previous literature, our method achieves the best result when input voxel resolution has size 32 and the performance begins to drop slightly when the resolution increases. The reason is that categories in ModelNet40 are visually different even at a low resolution, hence less detail is required and 32^3 is adequate for distinguishing one class from another. From Table 2, it can be observed that resolutions higher than 32^3 leads to small amounts of overfitting.

Our method has inferior accuracy compared to that of O-CNN (Wang et al., 2017), which is partly due to the influence of the input data. In their implementation they used more informative surface normals as input while in our experiments, we simply use the binary occupancy voxel grid. They have shown in their paper that there is approximately a 2% drop in accuracy by using binary input. Another reason is that instead of using max-pooling, we employ convolution of stride 2 in the leaf encoder which could lead to fewer patterns being captured. We did an additional experiment where we substituted the 2-stride convolution with 1-stride convolution followed by a max-pooling layer. The accuracy of 32^3 increases from 85.5% to 86.7%. However, this substitution increases the memory requirements of the network.

Note that the focus of this paper is mainly 3D shape autoencoder, we show in this section that by simply using the encoder our method can be easily converted into a lightweight classifier with good accuracy.

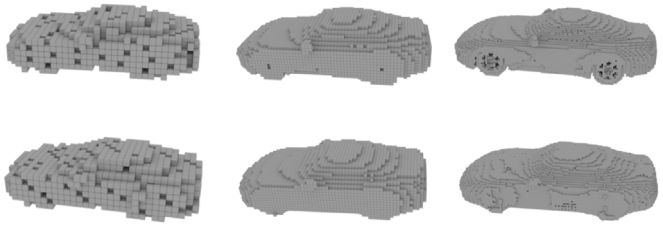
4.3. 3D shape reconstruction

Datasets. To evaluate 3D shape reconstruction, we employ two datasets: ShapeNet-Car and ShapeNetCorev2 (Chang et al., 2015). ShapeNet-Car contains 7497 car CAD models. ShapeNetCorev2 consists of 39,715 3D models from 13 categories.

Training protocol. We use batchsize = 50 in training. The average total number of iterations is around 300 for each class. Training takes approximately 20 h for a class containing 2000 samples in 128^3 resolution. For both ShapeNet-Car and ShapeNetCorev2 we split training and testing sets by 80% and 20%, respectively, as done by Groueix et al. (2018) and Wang et al. (2018). Note that the ground-truth node type is used in the training stage to choose the correct decoder. In the test stage, the type of a node is predicted by the trained node classifier.

Measurements. We show the qualitative reconstruction results in Figs. 5 and 6. For more visual reconstruction results please refer to the supplementary materials. For quantitative measurements, we use intersection over union (IoU) for ShapeNet-Car and Chamfer distance for

[1] Pictures of binary O-CNN and patch-based O-CNN are from Wang et al. (2018).



(a) 32^3 (b) 64^3 (c) 128^3

Fig. 6. Car reconstruction examples of different resolutions. Top row: ground-truth voxels. Bottom row: reconstructed voxels.

Table 2
3D shape classification accuracy on ModelNet40 dataset. Voxel-based methods are followed by their tested resolution. RocNet is followed by its resolution and the leaf size.

	acc.		acc.
3DShapeNets(32)	77.3%	RocNet(32-16)	85.5%
VoxNet(32)	83.0%	RocNet(64-32)	85.0%
Geometry image	83.9%	RocNet(128-32)	84.4%
OCNN(32)	**89.6** %	RocNet(256-32)	84.1%

Table 3
Reconstruction accuracy on ShapeNet-Car dataset. Numbers shown are intersection over union (IoU%) between reconstructed and ground-truth voxel in 3 different resolutions. Accuracy for OGN and dense in 128 is not reported in their paper. Boldfaced numbers emphasize the best results.

	OGN-p	OGN-k	Dense	RocNet-p	RocNet-k
32	92.4	93.9	92.4	**95.1**	94.6
64	88.4	90.4	89.0	**92.0**	**92.0**
128	–	–	–	87.0	**87.5**

Table 4
Computational efficiency on ShapeNet-Car dataset. Averaged GPU memory usage and time for one iteration are shown. Batch size is set to 1. Boldfaced numbers emphasize the best results.

	Memory (GB)		Time (s)	
	OGN	RocNet	OGN	RocNet
32	0.29	**0.017**	**0.016**	0.05
64	0.36	**0.026**	**0.06**	**0.06**
128	0.43	**0.089**	0.18	**0.10**

ShapeNetCorev2 dataset. This is mainly for the purpose of comparison with existing methods. Using IoU is straightforward since our network produces binary voxels. Chamfer distance is usually used for evaluating the performance of surface-generating methods such as Groueix et al. (2018) and Wang et al. (2018). To compare with this group of methods, we densely sample a set of points, $\mathcal{P}$, from the boundary voxels and a set of points, $\mathcal{G}$, from the ground-truth mesh. The Chamfer distance is calculated as:

$$d(\mathcal{P},\mathcal{G}) = \frac{1}{|\mathcal{P}|}\sum_{x\in\mathcal{P}}\min_{y\in\mathcal{G}}\|x-y\|^2 + \frac{1}{|\mathcal{G}|}\sum_{x\in\mathcal{G}}\min_{y\in\mathcal{P}}\|x-y\|^2. \quad (5)$$

Results and comparisons. Fig. 5 shows that RocNet produces fewer missing regions (the wheels) compared to O-CNN and patch-based O-CNN, which explains why RocNet outperforms the rest in the following quantitative analysis. We also observe the reconstructed shapes are more blurred than the ground-truth voxels (see the details of wheels at 128^3 resolution in Fig. 6).

IoU accuracy of the ShapeNet-Car dataset and the that of OGN (Tatarchenko et al., 2017) is shown in Table 3 and computational requirements are shown in Table 4. The Chamfer distance results of ShapeNetCorev2 are shown in Table 5. These results show that RocNet outperforms OGN at all three resolutions with lower memory consumption and faster execution time. Similar to OGN, we also test

Table 5
Chamfer distance tested on ShapeNetCorev2 dataset. The Chamfer distance is multiplied by 10^3 for better display. Boldfaced numbers emphasize the best results. Results of PSG and AtlasNet are from Groueix et al. (2018). Results of O-CNN and Adaptive-OCNN are provided in Wang et al. (2018).

	avg.	pla.	ben.	cab.	car.	cha.	mon.	lam.	spe.	fir.	cou.	tab.	cel.	wat.
PSG	1.91	1.11	1.46	1.91	1.59	1.90	2.20	3.59	3.07	0.94	1.83	1.83	1.71	1.69
AtlasNet(125)	1.51	0.86	**1.15**	1.76	1.56	1.55	1.69	**2.26**	2.55	**0.59**	1.69	1.47	1.31	1.23
OCNN	1.60	1.12	1.30	1.06	1.02	1.79	1.62	3.71	2.56	0.98	1.17	1.67	0.79	1.88
Adaptive OCNN	1.44	1.19	1.27	**1.01**	0.96	1.65	1.41	2.83	1.97	1.06	1.14	1.46	0.73	1.82
RocNet	**1.05**	**0.49**	1.34	1.09	**0.83**	**1.29**	**0.74**	2.32	**1.58**	0.73	**0.80**	**0.91**	**0.72**	**0.82**

Table 6
Shape segmentation accuracy and mIoU. We compare our method with four alternatives (Guo et al., 2015; Kalogerakis et al., 2017; Wang and Lu, 2019; Hegde and Gangisetty, 2021). Guo et al. (2015) and Kalogerakis et al. (2017) are reported in accuracy while Wang and Lu (2019) and Hegde and Gangisetty (2021) are reported in mIoU. For each category we show the number of training/test samples as well as the number of parts. In the last four rows we show the mean accuracy either category-wise or sample-wise. The averaged accuracy of categories which have more than 3 parts is also reported.

	#train/test	#parts	Guo	ShapePFCN	RocNet(acc)	Wang	PIG-Net	RocNet(mIoU)
Airplane	250/250	4	87.4	**90.3**	90.2	86.2	84.2	80.5
Bag	38/38	2	91.0	**94.6**	93.2	88.7	83.1	68.0
Cap	27/28	2	85.7	**94.5**	90.0	91.9	88.9	77.5
Car	250/250	4	80.1	**86.7**	81.9	79.8	78.6	65.5
Chair	250/250	4	66.8	82.9	**88.3**	92.0	91.7	80.0
Earphone	34/35	3	79.8	**84.9**	75.1	76.5	78.2	62.8
Guitar	250/250	3	89.9	91.8	**92.7**	92.0	94.4	84.4
Knife	196/196	2	77.1	82.8	**86.7**	86.4	89.5	77.7
Lamp	250/250	4	71.6	78.0	**80.0**	84.2	94.2	62.5
Laptop	222/223	2	82.7	95.3	**95.6**	96.1	96.3	92.0
Motorbike	101/101	6	80.1	**87.0**	**87.0**	78.4	66.2	64.9
Mug	92/92	2	95.1	96.0	**97.1**	96.3	91.6	86.0
Pistol	137/138	3	84.1	**91.5**	**91.5**	83.7	85.1	77.2
Rocket	33/33	3	76.9	**81.6**	75.9	65.4	64.8	52.4
Skateboard	76/76	3	89.6	91.9	**92.6**	77.0	93.5	75.7
Table	250/250	3	77.8	84.8	**90.5**	86.2	94.2	75.6
Avg.(Category)	–	–	82.2	**88.4**	88.0	–	85.9	73.3
Avg.(Category >3)	–	–	77.2	85.0	**85.4**	–	–	–
Avg.(Dataset)	–	–	80.6	87.5	**88.3**	87.5	90.5	76.0
Avg.(Dataset >3)	–	–	76.8	84.7	**84.9**	–	–	–

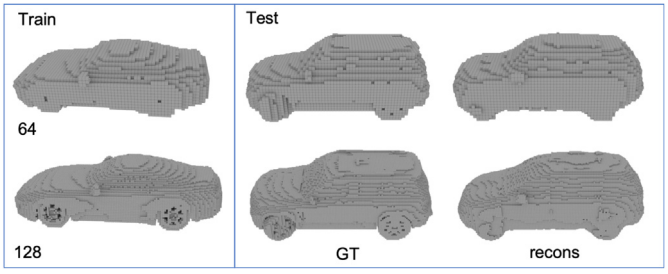


Fig. 7. Multi-resolutional reconstruction.

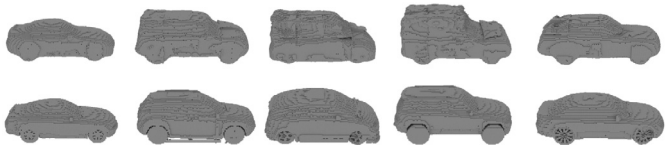


Fig. 8. Model generation results. Top row: 5 generated models. Bottom row: nearest model in training samples.

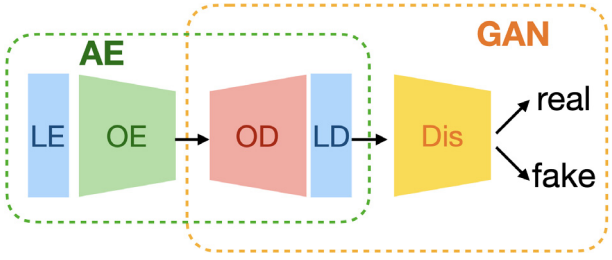


Fig. 9. Adversarial autoencoder structure.

our algorithm in two modes: with and without octree structure known. In the octree-prediction mode (named as "RocNet-p"), the type of each node is predicted by the node classifier while in octree-known mode (named as "RocNet-k"), the octree structure is given during decoding. Specifically, the node classifier is disabled in octree-known mode. Intuitively, less information is encoded in octree-known mode hence higher reconstruction accuracy should be achieved. However, as shown in the last two column of Table 3, these two modes have very similar accuracy. This implies that the capacity of the hidden representation is sufficient to store both the octree topology and the leaves. This validates our choice of $64 \times 4 \times 4 \times 4$ voxels for the hidden representation since using a higher resolution would not lead to a significant increase in accuracy. The rest of our experiments use prediction mode only.

We also report the GPU memory usage and the computational time for one single training iteration in Table 4. It coincides with the complexity analysis in Section 3.6. For all three resolutions tested, our algorithm consumes far less memory than OGN. We also observe our algorithm has a larger relative increase in memory compared to OGN, but a smaller increase in time. According to the complexity analysis, the memory consumption increases cubically with N. We expect our algorithm to take more memory with larger resolutions. However, it is still tractable when processing a 512^3 voxel grid in our experiments.

Table 5 shows the reconstruction accuracy on the ShapeNetCorev2 dataset, as measured by the Chamfer distance. Compared to ShapeNet-Car, this has a more diverse range of samples. We compare our method with four alternative schemes: PSG (Fan et al., 2017), AtlasNet (Groueix et al., 2018) with 125 predicted mesh patches, O-CNN (Wang et al., 2017) and Adaptive O-CNN (Wang et al., 2018), among which PSG is

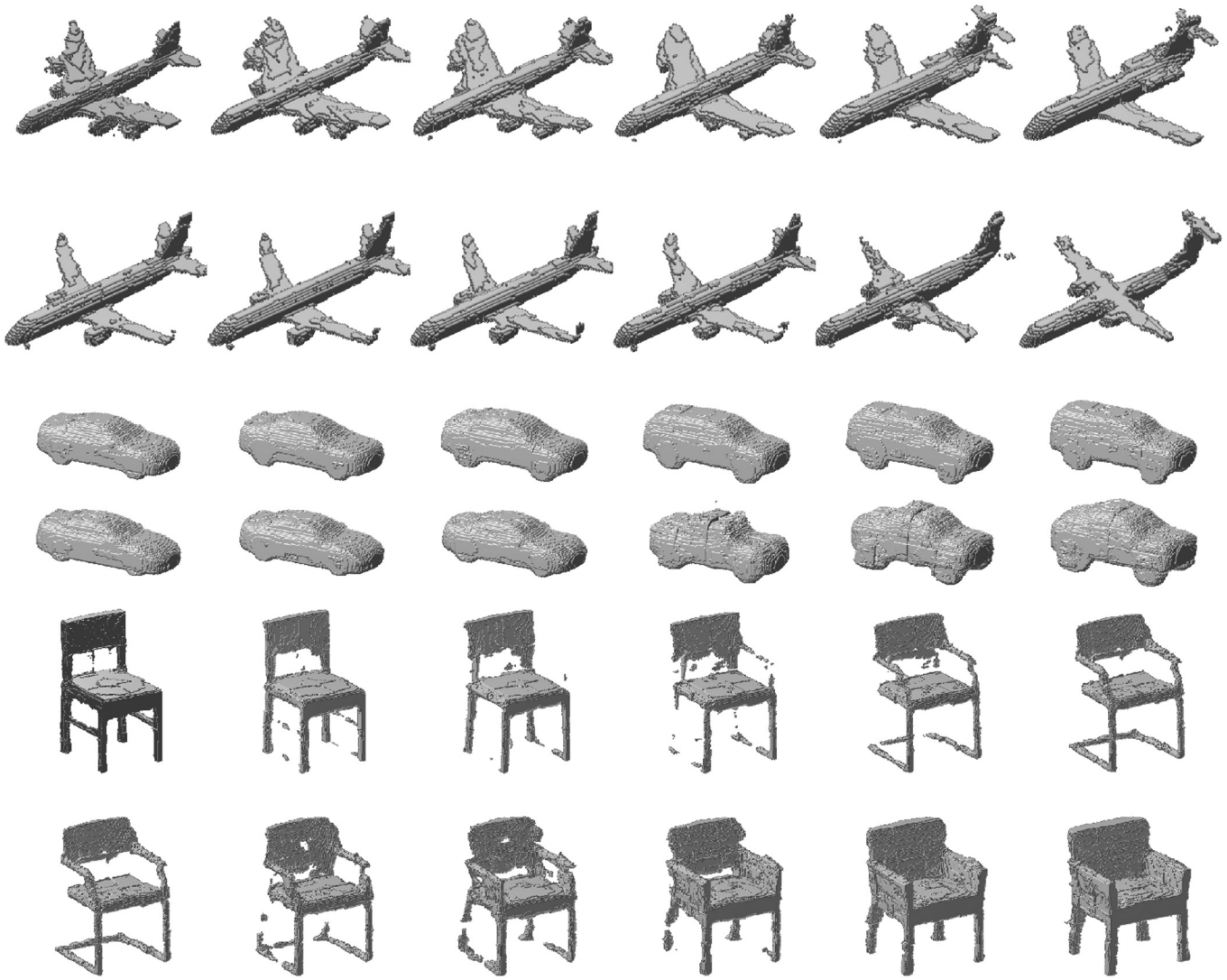


Fig. 10. Shape interpolation.

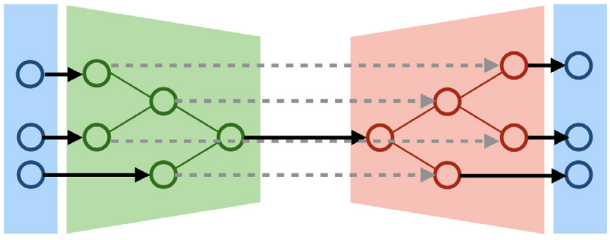


Fig. 11. Segmentation structure. Dotted lines: skip connections.

the only point set generating method while the others generate meshes. For comparison, we calculate the Chamfer distance in the same protocol as for O-CNN. RocNet has the best performance in most categories followed by Adaptive O-CNN and AtlasNet. It is worth noticing that since RocNet generates voxels instead of a mesh, there is an intrinsic inaccuracy between the cube-like voxels and the smooth mesh. Mesh-based methods such as Adaptive O-CNN and AtlasNet do not suffer from this drawback. We expect our method to perform even better with a patch-based representation, which is an area for future work.

To demonstrate the flexibility of our multi-resolution architecture, we train the networks on a mixture of 64^3 and 128^3 voxel grids with equal population and the reconstruction still works well as shown in Fig. 7. In this case each encoder/decoder learns multiple resolutions flexibly.

4.4. 3D shape generation

In this section we present the results of shape generation using RocNet. We first generate new shapes by the default autoencoder structure and then we augment the loss function with an adversarial term to achieve more plausibility.

Fig. 8 shows five models generated by a RocNet autoencoder trained on the ShapeNet-Car dataset. We generate new models by randomly sampling within the convex hull of the trained samples in 80D feature space. Please note that the octree topology is generated on the fly in this experiment since it is impossible to obtain the ground-truth which exists in reconstruction task. From Fig. 8 we observe that RocNet is able to establish a semantically plausible shape manifold where each generated model can be visually identified as a car. We also present the corresponding training model closest to the sampled point in feature space. Some generated models are very similar to existing models such as the first example. There also exist some examples where there are visible differences between existing ones. This indicates that RocNet has captured some semantic information from training samples. However, when we try to morph between shapes, we observe some

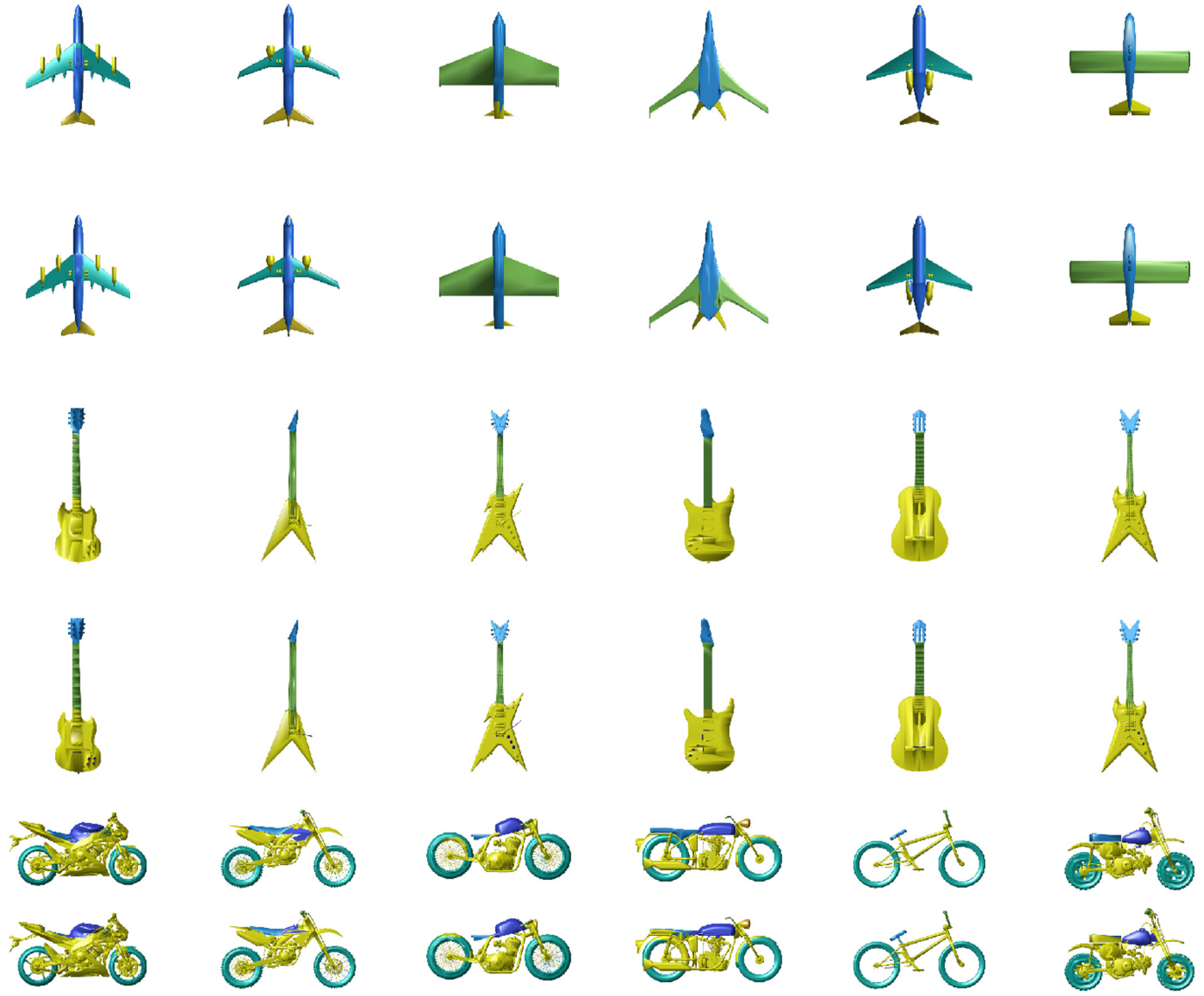


Fig. 12. Segmentation examples. Row 1,3 and 5 contain 6 ground-truth shapes respectively. Row 2,4 and 6 show the corresponding segmented shapes. Each color indicates an individual part. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

noisy intermediate shapes. This indicates simply using the autoencoder loss might be insufficient to regularize the space.

In order to preserve more plausibility in generated shapes, we integrate our autoencoder with an adversarial (GAN Goodfellow et al., 2014) term. The resulting network is shown in Fig. 9. It features an autoencoder structure as well as an adversarial component. For the discriminator, we first tried to use the same structure as the recursive encoder used in 4.3. However, the performance was not satisfactory. As a minor modification, we use 3D convolutional layers instead of recursive layers for the discriminator, the output of which is a real number produced by sigmoid non-linearity indicating the confidence of real samples. The decoder remains a recursive structure, and is shared by both the autoencoder and the GAN component.

We show several shape morphing examples in Fig. 10. In each row the shape morphs from the left to right by linearly interpolating between two feature vectors in the latent space. The interpolated feature vector is then fed into the octree decoder for decoding. The intermediate shapes preserve plausibility while smooth changes are observed. This is due to the adversarial term we add to the loss function, which further regularizes the latent space.

4.5. 3D shape segmentation

Datasets. We adopt ShapeNetCore dataset for the evaluation of segmentation. The ground-truth segmentations are manually labeled (Yi et al., 2016). There are 17,773 expert-verified segmentations of 3D models across 16 categories in the dataset. The training and test sets are roughly equally split within each category. To compare with the existing methods fairly we use the same splits as used in Kalogerakis et al. (2017).

Network architecture. We use RocNet for 3D semantic segmentation with minor modifications. The segmenting network has a U-shape as shown in Fig. 11. The input and output are multi-channel 3D voxels indicating the confidence for each semantic category instead of binary occupancy as used in reconstruction. Additionally, we add skip-connections between the corresponding octants in encoder and decoder. With skip connections, more fine-grained details and low-level features (Ronneberger et al., 2015) can be recovered in the prediction.

Measurements. Similar to Kalogerakis et al. (2017), the labeling accuracy of a mesh is given by the percentage of the correctly labeled points sampled on the mesh (Yi et al., 2016). Since our method segments volumetric octrees instead of polygonal meshes, we first map

voxel labels to the corresponding polygon faces, and the labelings of sampled points are obtained by mapping from the faces they belong to. We use 128 × 128 × 128 resolution across all our experiments.

Results and comparisons. Segmentation accuracy is reported in Table 6, and examples of segmented shapes are shown in Fig. 12. We achieve comparable results to Kalogerakis et al. (2017) and better performance than (Guo et al., 2015). One major difference between our method and many existing ones is our method segments voxels directly instead of meshes or views, but the 3D convolution can require more training data to escape from overfitting. This also explains why for classes with very limited training samples such as *Cap*, *Earphone* and *Rocket*, our method has inferior performance compared to view-based (Kalogerakis et al., 2017) and mesh feature-based (Guo et al., 2015) methods. The other reason is that Kalogerakis et al. (2017) is initialized with filters pre-trained on image processing tasks (Yu and Koltun, 2015) while our method is only trained by the provided 3D shapes in the dataset. As reported in Kalogerakis et al. (2017), a degraded variant of their method without pretraining has an accuracy of 86.3% while ours is 88.0%. Furthermore, they use a CRF term for better integration of segmented views. We could also integrate such a term for higher accuracy, but restrict ourselves to the simple case in order to clearly illustrate the effectiveness of our proposed structure.

In addition to the segmentation accuracy, we also report comparison results with two dedicated semantic segmentation methods (Wang and Lu, 2019; Hegde and Gangisetty, 2021) in mIoU (mean intersection over union). The results are shown in last 3 columns of Table 6.

We use the same protocol as used in Wang and Lu (2019). For each shape, IoUs of all parts are averaged to obtain the mIoU. Per category average mIoU is computed by averaging across all shapes within the certain category. Dataset mIoU is computed through a weighted average of per category mIoU with the weights being the number of shapes in each category. We observe a drop in mIoU for categories such as earphone, lamps. This drop is due to the fact that voxel resolution we used (128^3) is relatively low for dense point-cloud segmentation tasks. In this case, the voxels are not fine-grained enough to accommodate tiny (such as airplane engines) and non-rigid parts (such as earphone line and car wheels). Improvements can be made by using higher resolutions or using pointcloud-based leaf encoder and decoder.

Since our method directly segments 3D voxels, it can be easily applied to many scenarios where a clean mesh is not available or polygonal mesh representation is not preferred such as robotics navigation and self-driving applications. For these applications methods such as Guo et al. (2015) and Kalogerakis et al. (2017) might be inapplicable.

5. Conclusion and future work

We propose a general-purpose recursive octree-based network which is demonstrated to be effective in multiple 3D processing tasks such as classification, reconstruction, generation and semantic segmentation. It encodes and decodes octrees by either recursively merging or producing octants. Our key insight is the recursive nature of the proposed network fits the hierarchical feature of octrees well. By using an octree representation we are able to save a large amount of storage and computational time and higher accuracy is achieved compared to existing methods for 3D shape reconstruction and generation. The architecture also provides good results for 3D shape classification and semantic segmentation. We demonstrated the advantages of our method by four experiments. The complexity of our method is analyzed both theoretically and empirically.

Our proposed method can be directly applied to explicit voxels. However, our method is more of an efficient way of recursively dividing and combining the space. So it is also possible to apply it to other representations such as pointclouds with certain modifications. Specifically, we need a pointcloud-based leaf encoder instead of a 3D convolutional NN. The rest of the architecture will remain the same.

A promising improvement to RocNet is to use more input information. As suggested by existing work (Wang et al., 2019, 2017), using more informative input information such as normals or a truncated signed distance field leads to better performance. RocNet is compatible with arbitrary input formats as long as it dose not violate the sparseness of 3D data. This sparseness arises naturally from the fact that surfaces in the world form manifolds in 3D space. We will also explore the possibility of directly generating meshes as output of the network.

CRediT authorship contribution statement

Juncheng Liu: Conceptualization, Methodology, Software, Investigation, Writing – original draft. **Steven Mills:** Supervision, Project administration, Funding acquisition, Writing – review & editing. **Brendan McCane:** Supervision, Project administration, Funding acquisition, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This project was funded by Science for Technological Innovation[2] under the spearhead project: Adaptive learning robots to complement the human workforce.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cviu.2022.103555.

References

Brock, A., Lim, T., Ritchie, J.M., Weston, N., 2016. Generative and discriminative voxel modeling with convolutional neural networks. arXiv preprint arXiv:1608.04236.

Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al., 2015. Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012.

Clevert, D.-A., Unterthiner, T., Hochreiter, S., 2015. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289.

Fan, H., Su, H., Guibas, L.J., 2017. A point set generation network for 3d object reconstruction from a single image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 605–613.

Garcia-Garcia, A., Gomez-Donoso, F., Garcia-Rodriguez, J., Orts-Escolano, S., Cazorla, M., Azorin-Lopez, J., 2016. Pointnet: A 3d convolutional neural network for real-time object class recognition. In: 2016 International Joint Conference on Neural Networks. IJCNN, IEEE, pp. 1578–1584.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. In: Advances in Neural Information Processing Systems. pp. 2672–2680.

Groueix, T., Fisher, M., Kim, V.G., Russell, B.C., Aubry, M., 2018. A papier-mâché approach to learning 3d surface generation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 216–224.

Guo, K., Zou, D., Chen, X., 2015. 3D mesh labeling via deep convolutional neural networks. ACM Trans. Graph. 35 (1), 1–12.

Häne, C., Tulsiani, S., Malik, J., 2017. Hierarchical surface prediction for 3d object reconstruction. In: 2017 International Conference on 3D Vision (3DV). IEEE, pp. 412–420.

Hegde, S., Gangisetty, S., 2021. PIG-Net: Inception based deep learning architecture for 3D point cloud segmentation. Comput. Graph. 95, 13–22.

Hegde, V., Zadeh, R., 2016. Fusionnet: 3d object classification using multiple data representations. arXiv preprint arXiv:1607.05695.

Kalogerakis, E., Averkiou, M., Maji, S., Chaudhuri, S., 2017. 3D shape segmentation with projective convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3779–3788.

Klokov, R., Lempitsky, V., 2017. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 863–872.

[2] https://www.sftichallenge.govt.nz/.

Li, J., Xu, K., Chaudhuri, S., Yumer, E., Zhang, H., Guibas, L., 2017. Grass: Generative recursive autoencoders for shape structures. ACM Trans. Graph. 36 (4), 1–14.

Liu, J., Mills, S., McCane, B., 2020. RocNet: Recursive octree network for efficient 3D deep representation. In: 2020 International Conference on 3D Vision (3DV). pp. 414–422. http://dx.doi.org/10.1109/3DV50981.2020.00051.

Maturana, D., Scherer, S., 2015. VoxNet: A 3D convolutional neural network for real-time object recognition. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, pp. 922–928.

Maturana, D., Scherer, S., 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, IEEE, pp. 922–928.

Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R., 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In: European Conference on Computer Vision. Springer, pp. 405–421.

Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S., 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 165–174.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. PyTorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, Vol. 32. Curran Associates, Inc., pp. 8024–8035, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Polosukhin, I., Zavershynskyi, M., 2018. Nearai/torchfold: v0.1.0. http://dx.doi.org/10.5281/zenodo.1299387.

Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J., 2018. Frustum pointnets for 3d object detection from rgb-d data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 918–927.

Qi, C.R., Su, H., Mo, K., Guibas, L.J., 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 652–660.

Qi, C.R., Yi, L., Su, H., Guibas, L.J., 2017b. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. Adv. Neural Inf. Process. Syst. 30.

Riegler, G., Osman Ulusoy, A., Geiger, A., 2017a. Octnet: Learning deep 3d representations at high resolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3577–3586.

Riegler, G., Ulusoy, A.O., Bischof, H., Geiger, A., 2017b. Octnetfusion: Learning depth fusion from data. In: 2017 International Conference on 3D Vision (3DV). IEEE, pp. 57–66.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, pp. 234–241.

Shen, X., Stamos, I., 2020. Frustum VoxNet for 3D object detection from RGB-D or Depth images. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 1698–1706.

Sinha, A., Bai, J., Ramani, K., 2016. Deep learning 3D shape surfaces using geometry images. In: European Conference on Computer Vision. Springer, pp. 223–240.

Socher, R., 2014. Recursive Deep Learning for Natural Language Processing and Computer Vision (Ph.D. thesis). Citeseer.

Socher, R., Lin, C.C.-Y., Ng, A.Y., Manning, C.D., 2011. Parsing natural scenes and natural language with recursive neural networks. In: ICML.

Tatarchenko, M., Dosovitskiy, A., Brox, T., 2017. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2088–2096.

Wang, C., Cheng, M., Sohel, F., Bennamoun, M., Li, J., 2019. NormalNet: A voxel-based CNN for 3D object classification and retrieval. Neurocomputing 323, 139–147.

Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., Tong, X., 2017. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. ACM Trans. Graph. 36 (4), 1–11.

Wang, Z., Lu, F., 2019. VoxSegNet: Volumetric CNNs for semantic part segmentation of 3D shapes. IEEE Trans. Vis. Comput. Graphics 26 (9), 2919–2930.

Wang, P.-S., Sun, C.-Y., Liu, Y., Tong, X., 2018. Adaptive O-CNN: A patch-based deep representation of 3D shapes. ACM Trans. Graph. 37 (6), 1–11.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J., 2015. 3d shapenets: A deep representation for volumetric shapes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1912–1920.

Wu, J., Zhang, C., Xue, T., Freeman, B., Tenenbaum, J., 2016. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. Adv. Neural Inf. Process. Syst. 29.

Yi, L., Kim, V.G., Ceylan, D., Shen, I.-C., Yan, M., Su, H., Lu, C., Huang, Q., Sheffer, A., Guibas, L., 2016. A scalable active framework for region annotation in 3d shape collections. ACM Trans. Graph. (ToG) 35 (6), 1–12.

Yu, F., Koltun, V., 2015. Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122.

Zhi, S., Liu, Y., Li, X., Guo, Y., 2018. Toward real-time 3D object recognition: A lightweight volumetric cnn framework using multitask learning. Comput. Graph. 71, 199–207.