

# Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs

Maxim Tatarchenko<sup>1</sup>

tatarchm@cs.uni-freiburg.de

Alexey Dosovitskiy<sup>1,2</sup>

adosovitskiy@gmail.com

Thomas Brox<sup>1</sup>

brox@cs.uni-freiburg.de

<sup>1</sup>University of Freiburg<sup>2</sup>Intel Labs

## Abstract

We present a deep convolutional decoder architecture that can generate volumetric 3D outputs in a compute- and memory-efficient manner by using an octree representation. The network learns to predict both the structure of the octree, and the occupancy values of individual cells. This makes it a particularly valuable technique for generating 3D shapes. In contrast to standard decoders acting on regular voxel grids, the architecture does not have cubic complexity. This allows representing much higher resolution outputs with a limited memory budget. We demonstrate this in several application domains, including 3D convolutional autoencoders, generation of objects and whole scenes from high-level representations, and shape from a single image.

## 1. Introduction

Up-convolutional<sup>1</sup> decoder architectures have become a standard tool for tasks requiring image generation [10, 28, 21] or per-pixel prediction [26, 9]. They consist of a series of convolutional and up-convolutional (upsampling+convolution) layers operating on regular grids, with resolution gradually increasing towards the output of the network. The architecture is trivially generalized to volumetric data. However, because of cubic scaling of computational and memory requirements, training up-convolutional decoders becomes infeasible for high-resolution three-dimensional outputs.

Poor scaling can be resolved by exploiting structure in the data. In many learning tasks, neighboring voxels on a voxel grid share the same state — for instance, if the voxel grid represents a binary occupancy map or a multi-class labeling of a three-dimensional object or a scene. In this case, data can be efficiently represented with octrees — data structures with adaptive cell size. Large regions of space sharing the same value can be represented with a single

<sup>1</sup>Also known as deconvolutional

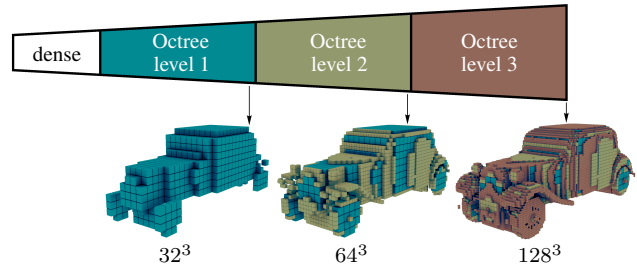


Figure 1. The proposed OGN represents its volumetric output as an octree. Initially estimated rough low-resolution structure is gradually refined to a desired high resolution. At each level only a sparse set of spatial locations is predicted. This representation is significantly more efficient than a dense voxel grid and allows generating volumes as large as  $512^3$  voxels on a modern GPU in a single forward pass.

large cell of an octree, resulting in savings in computation and memory compared to a fine regular grid. At the same time, fine details are not lost and can still be represented by small cells of the octree.

We present an octree generating network (OGN) - a convolutional decoder operating on octrees. The coarse structure of the network is illustrated in Figure 1. Similar to a usual up-convolutional decoder, the representation is gradually convolved with learned filters and up-sampled. The difference is that, starting from a certain layer in the network, dense regular grids are replaced by octrees. Therefore, the OGN predicts large uniform regions of the output space already at early decoding stages, saving the computation for the subsequent high-resolution layers. Only regions containing fine details are processed by these more computationally demanding layers.

In this paper, we focus on generating shapes represented as binary occupancy maps. We thoroughly compare OGNs to standard dense nets on three tasks: auto-encoding shapes, generating shapes from a high-level description, and reconstructing 3D objects from single images. OGNs yield the same accuracy as conventional dense decoders while consuming significantly less memory and being much faster at

high resolutions. For the first time, we can generate shapes of resolution as large as  $512^3$  voxels in a single forward pass. Our OGN implementation is publicly available<sup>2</sup>.

## 2. Related work

The majority of deep learning approaches generate volumetric data based on convolutional networks with feature maps and outputs represented as voxel grids. Applications include single- and multi-view 3D object reconstruction trained in supervised [15, 6, 8, 18] and unsupervised [38, 13, 20] ways, probabilistic generative modeling of 3D shapes [37, 36, 30], semantic segmentation [3, 5] and shape deformation [40]. A fundamental limitation of these approaches is the low resolution of the output. Memory and computational requirements of approaches based on the voxel grid representation scale cubically with the output size. Thus, training networks with resolutions higher than  $64^3$  comes with memory issues on the GPU or requires other measures to save memory, such as reducing the batch size or generating the volume part-by-part. Moreover, with growing resolution, training times become prohibitively slow.

Computational limitations of the voxel grid representation led to research on alternative representations of volumetric data in deep learning. Tatarchenko et al. [33] predict RGB images and depth maps for multiple views of an object, and fuse those into a single 3D model. This approach is not trainable end to end because of the post-processing fusion step, and is not applicable to objects with strong self-occlusion. Sinha et al. [31] convert shapes into two-dimensional geometry images and process those with conventional CNNs – an approach only applicable to certain classes of topologies. Networks producing point clouds have been applied to object generation [11] and semantic segmentation [27]. By now, these architectures have been demonstrated to generate relatively low-resolution outputs. Scaling these networks to higher resolution is yet to be explored. Tulsiani et al. [34] assemble objects from volumetric primitives. Yi et al. [39] adapt the idea of graph convolutions in the spectral domain to semantic segmentation of 3D shapes. Their approach requires all samples to have aligned eigenbasis functions, thus limiting possible application domains.

Promising alternative representations that are not directly applicable to generating 3D outputs have been explored on analysis tasks. Masci et al. [24] proposed geodesic CNNs for extracting local features in non-Euclidean domains. Our approach is largely inspired by Graham’s sparse convolutional networks [16, 17], which enable efficient shape analysis by storing a sparse set of non-trivial features instead of dense feature maps. The

OGN essentially solves the inverse problem by predicting which regions of the output contain high-resolution information and by restricting extensive calculations only to those regions.

The recent pre-print by Riegler et al. [29] builds on the same general idea as our work: designing convolutional networks that operate on octrees instead of voxel grids. However, the implementation and the application range of the method is very different from our work. When generating an octree, Riegler et al. assume the octree structure to be known at test time. This is the case, for example, in semantic segmentation, where the structure of the output octree can be set to be identical to that of the input. However, in many important scenarios — any kind of 3D reconstruction, shape modeling, RGB-D fusion, superresolution — the structure of the octree is not known in advance and must be predicted. The method of Riegler et al. is not applicable in these cases. Moreover, the OGN is more flexible in that it allows for octrees with an arbitrary number of levels.

## 3. Octrees

An octree [25] is a 3D grid structure with adaptive cell size, which allows for lossless reduction of memory consumption compared to a regular voxel grid. Octrees have a long history in classical 3D reconstruction and depth map fusion [22, 2, 12, 35, 7, 32]. A function defined on a voxel grid can be converted into a function defined on an octree. This can be done by starting from a single cell representing the entire space and recursively partitioning cells into eight octants. If every voxel within a cell has the same function value, this cell is not subdivided and becomes a leaf of the tree. The set of cells at a certain resolution is referred to as an octree level. The recursive subdivision process can also be started not from the whole volume, but from some initial coarse resolution. Then the maximal octree cell size is given by this initial resolution. The most straightforward way of implementing an octree is to store in each cell pointers to its children. In this case, the time to access an element scales linearly with the tree’s depth, which can become costly at high resolutions. We use a more efficient implementation that exploits hash tables. An octree cell with spatial coordinates  $\mathbf{x} = (x, y, z)$  at level  $l$  is represented as an index-value pair  $(m, v)$ , where  $v$  can be any kind of discrete or continuous signal.  $m$  is calculated from  $(\mathbf{x}, l)$  using Z-order curves [14]

$$m = \mathcal{Z}(\mathbf{x}, l), \quad (1)$$

which is a computationally cheap transformation implemented using bit shifts. An octree  $O$  is, hence, a set of all pairs

$$O = \{(m, v)\}. \quad (2)$$

<sup>2</sup><https://github.com/lmb-freiburg/ogn>

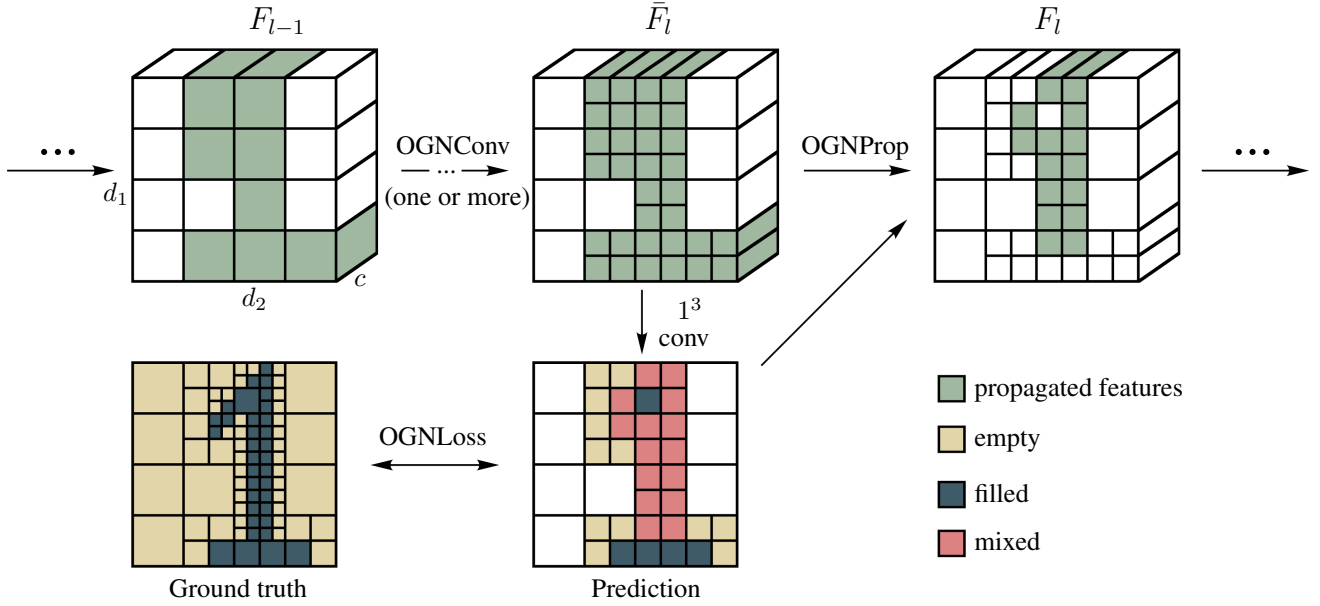


Figure 2. Single block of an OGN illustrated as 2D quadtree for simplicity. After convolving features  $F_{l-1}$  of the previous block with weight filters, we directly predict the occupancy values of cells at level  $l$  using  $1^3$  convolutions. Features corresponding to “filled” and “empty” cells are no longer needed and thus not propagated, which yields  $F_l$  as the final output of this block.

Storing this set as a hash table allows for constant-time element access.

When training networks, we will need to compare two different octrees  $O^1$  and  $O^2$ , i.e. for each cell  $(\mathbf{x}, l)$  from  $O^1$ , query the corresponding signal value  $v$  in  $O^2$ . Since different octrees have different structure, two situations are possible. If  $\mathcal{Z}(\mathbf{x}, k)$  is stored at a level  $k$  in  $O^2$ , which is the same or lower than  $l$ , the signal value of this cell can be uniquely determined. If  $\mathcal{Z}(\mathbf{x}, k)$  is stored at one of the later levels, the cell is subdivided in  $O^2$ , and the value of the whole cell is not defined. To formalize this, we introduce a function  $f$  for querying the signal value of an arbitrary cell with index  $m = \mathcal{Z}(\mathbf{x}, l)$  from octree  $O$ :

$$f(m, O) = \begin{cases} v, & \text{if } \exists k \leq l : (\mathcal{Z}(\mathbf{x}, k), v) \in O \\ \emptyset, & \text{otherwise} \end{cases}, \quad (3)$$

where  $\emptyset$  denotes an unavailable value.

#### 4. Octree Generating Networks

An Octree Generating Network (OGN) is a convolutional decoder that yields an octree as output: both the structure, i.e. which cells should be subdivided, and the signal value of each cell. In this work we concentrate on binary occupancy values  $v \in \{0, 1\}$ , but the proposed framework can be easily extended to support arbitrary signals. As shown in Figure 1, an OGN consists of a block operating on dense regular grids, followed by an arbitrary number of hash-table-based octree blocks.

The dense block is a set of conventional 3D convolutional and up-convolutional layers producing a feature map of size  $d_1 \times d_2 \times d_3 \times c$  as output, where  $\{d_i\}$  are the spatial dimension and  $c$  is the number of channels.

From here on, the representation is processed by our custom layers operating on octrees. The regular-grid-based feature map produced by the dense block is converted to a set of index-value pairs stored as a hash table (with values being feature vectors), and is further processed in this format. We organize octree-based layers in blocks, each responsible for predicting the structure and the content of a single level of the generated octree.

Figure 2 illustrates the functioning of a single such block that predicts level  $l$  of an octree. For the sake of illustration, we replaced three-dimensional octrees by two-dimensional quadtrees. Feature maps in Figure 2 are shown as dense arrays only for simplicity; in fact the green cells are stored in hash maps, and the white cells are not stored at all. We now give a high-level overview of the block and then describe its components in more detail.

Input to the block is a sparse hash-table-based convolutional feature map  $F_{l-1}$  of resolution  $(d_1 \cdot 2^{l-1}, d_2 \cdot 2^{l-1}, d_3 \cdot 2^{l-1})$  produced by the previous block. First this feature map is processed with a series of custom convolutional layers and one up-convolutional layer with stride 2, all followed by non-linearities.

This yields a new feature map  $\bar{F}_l$  of resolution  $(d_1 \cdot 2^l, d_2 \cdot 2^l, d_3 \cdot 2^l)$ . Based on this feature map, we directly predict the content of level  $l$ . For each cell, there is a two-fold decision to be made: should it be kept at level  $l$ , and if yes,

what should be the signal value in this cell? In our case making this decision can be formulated as classifying the cell as being in one of three states: "empty", "filled" or "mixed". These states correspond to the outputs of state-querying function  $f$  from eq. (3), with "empty" and "filled" being the signal values  $v$ , and "mixed" being the state where the value is not determined. We make this prediction using a convolutional layer with  $1^3$  filters followed by a three-way softmax. This classifier is trained in a supervised manner with targets provided by the ground truth octree.

Finally, in case the output resolution has not been reached, features from  $\bar{F}_l$  that correspond to "mixed" cells are propagated to the next layer<sup>3</sup> and serve as an input feature map  $F_l$  to the next block.

In the following subsections, we describe the components of a single octree block in more detail: the octree-based convolution, the loss function, and the feature propagation mechanism.

#### 4.1. Convolution

We implemented a custom convolutional layer *OGNConv*, which operates on feature maps represented as hash tables instead of usual dense arrays. Our implementation supports strided convolutions and up-convolutions with arbitrary filter sizes. It is based on representing convolution as a single matrix multiplication, similar to standard *caffe* [19] code for dense convolutions.

In the dense case, the feature tensor is converted to a matrix with the *im2col* operation, then multiplied with the weight matrix of the layer, and the result is converted back into a dense feature tensor using the *col2im* operation. In OGN, instead of storing full dense feature tensors, only a sparse set of relevant features is stored at each layer. These features are stored in a hash table, and we implemented custom operations to convert a hash table to a feature matrix and back. The resulting matrices are much smaller than those in the dense case. Convolution then amounts to multiplying the feature matrix by the weight matrix. Matrix multiplication is executed on GPU with standard optimized functions, and our conversion routines currently run on CPU. Even with this suboptimal CPU implementation, computation times are comparable to those of usual dense convolutions at  $32^3$  voxel resolution. At higher resolutions, *OGNConv* is much faster than dense convolutions (see section 5.2).

Unlike convolutions on regular grids, OGN convolutions are not shift invariant by design. This is studied in Section E of the Appendix.

<sup>3</sup>Additional neighboring cells may have to be propagated if needed by subsequent convolutional layers. This is described in section 4.3.

#### 4.2. Loss

The classifier at level  $l$  of the octree outputs the probabilities of each cell from this level being "empty", "filled" or "mixed", that is, a three-component prediction vector  $\mathbf{p}_m = (p_m^0, p_m^1, p_m^2)$  for cell with index  $m$ . We minimize the cross-entropy between the network predictions and the cell states of the ground truth octree  $O_{GT}$ , averaged over the set  $M_l$  of cells predicted at layer  $l$ :

$$\mathcal{L}_l = \frac{1}{|M_l|} \sum_{m \in M_l} \left[ \sum_{i=0}^2 h^i(f(m, O_{GT})) \log p_m^i \right], \quad (4)$$

where function  $\mathbf{h}$  yields a one-hot encoding  $(h^0, h^1, h^2)$  of the cell state value returned by  $f$  from eq. (3). Loss computations are encapsulated in our custom *OGNLoss* layer.

The final OGN objective is calculated as a sum of loss values from all octree levels

$$\mathcal{L} = \sum_{l=1}^L \mathcal{L}_l. \quad (5)$$

#### 4.3. Feature propagation

At the end of each octree block there is an *OGNProp* layer that propagates to the next octree block features from cells in the "mixed" state, as well as from neighboring cells if needed to compute subsequent convolutions. Information about the cell state can either be taken from the ground truth octree, or from the network prediction. This spawns two possible propagation modes: using the known tree structure (*Prop-known*) and using the predicted tree structure (*Prop-pred*). Section 4.4 describes use cases for these two modes.

The set of features to be propagated depends on the kernel size in subsequent *OGNConv* layers. The example illustrated in Figure 2 only holds for  $2^3$  up-convolutions which do not require any neighboring elements to be computed. To use larger convolutional filters or multiple convolutional layers, we must propagate not only the features of the "mixed" cells, but also the features of the neighboring cells required for computing the convolution at the locations of the "mixed" cells. The size of the required neighborhood is computed based on the network architecture, before the training starts. Details are provided in Section C of the Appendix.

#### 4.4. Training and testing

The OGN decoder is end-to-end trainable using standard backpropagation. The only subtlety is in selecting the feature propagation modes during training and testing. At training time the octree structure of the training samples is always available, and therefore the *Prop-known* mode can be used. At test time, the octree structure may or may not

be available. We have developed two training regimes for these two cases.

If the tree structure is available at test time, we simply train the network with *Prop-known* and test it the same way. This regime is applicable for tasks like semantic segmentation, or, more generally, per-voxel prediction tasks, where the structure of the output is exactly the same as the structure of the input.

If the tree structure is not available at test time, we start by training the network with *Prop-known*, and then fine-tune it with *Prop-pred*. This regime is applicable to any task with volumetric output.

We have also tested other regimes of combining *Prop-pred* and *Prop-known* and found those to perform worse than the two described variants. This is discussed in more detail in Section B of the Appendix.

## 5. Experiments

In our experiments we verified that the OGN architecture performs on par with the standard dense voxel grid representation, while requiring significantly less memory and computation, particularly at high resolutions. The focus of the experiments is on showcasing the capabilities of the proposed architecture. How to fully exploit the new architecture in practical applications is a separate problem that is left to future work.

### 5.1. Experimental setup

For all OGN decoders used in our evaluations, we followed the same design pattern: 1 or 2 up-convolutional layers interleaved with a convolutional layer in the dense block, followed by multiple octree blocks depending on the output resolution. In the octree blocks we used  $2^3$  up-convolutions. We also evaluated two other architecture variants, presented in section 5.3.1. ReLU non-linearities were applied after each (up-)convolutional layer. The number of channels in the up-convolutional layers of the octree blocks was set to 32 in the outermost layer, and was increased by 16 in each preceding octree block. The exact network architectures used in individual experiments are shown in Section F of the Appendix.

The networks were trained using ADAM [23] with initial learning rate 0.001,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ . The learning rate was decreased by a factor of 10 after 30K and 70K iterations. We did not apply any additional regularization.

For quantitative evaluations, we converted the resulting octrees back to regular voxel grids, and computed the Intersection over Union (IoU) measure between the ground truth model and the predicted model. To quantify the importance of high-resolution representations, in some experiments we upsampled low-resolution network predictions to high-resolution ground truth using trilinear interpolation, and later binarization with a threshold of 0.5. We explicitly

specify the ground truth resolution in all experiments where this was done.

If not indicated otherwise, the results were obtained in the *Prop-pred* mode.

#### 5.1.1 Datasets

In our evaluations we used three datasets:

**ShapeNet-all** Approximately 50.000 CAD models from 13 main categories of the ShapeNet dataset [4], used by Choy et al. [6]. We also used the renderings provided by Choy et al. [6].

**ShapeNet-cars** A subset of ShapeNet-all consisting of 7497 car models.

**BlendSwap** A dataset of 4 scenes we manually collected from [blendswap.com](http://blendswap.com), a website containing a large collection of Blender models.

All datasets were voxelized in multiple resolutions from  $32^3$  to  $512^3$  using the *binvox*<sup>4</sup> tool, and were converted into octrees. We set the interior parts of individual objects to be filled, and the exterior to be empty.

### 5.2. Computational efficiency

We start by empirically demonstrating that OGNs can be used at high resolutions when the voxel grid representation becomes impractical both because of the memory requirements and the runtime.

The number of elements in a voxel grid is uniquely determined by its resolution, and scales cubically as the latter increases. The number of elements in an octree depends on the data, leading to variable scaling rates: from constant for cubic objects aligned with the grid, to cubic for pathological shapes such as a three-dimensional checkerboard. In practice, octrees corresponding to real-world objects and scenes scale approximately quadratically, since they represent smooth two-dimensional surfaces in a three-dimensional space.

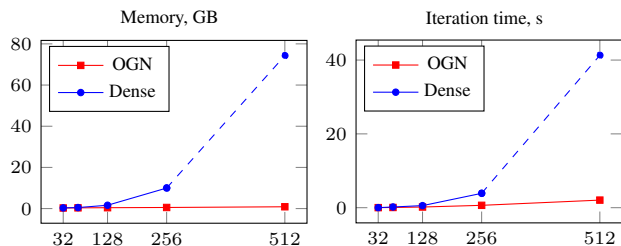


Figure 3. Memory consumption (left) and iteration time (right) of OGN and a dense network at different output resolutions. Forward and backward pass, batch size 1.

<sup>4</sup><http://www.patrickmin.com/binvox>

| Resolution | Memory, GB |             | Iteration time, s |             |
|------------|------------|-------------|-------------------|-------------|
|            | Dense      | OGN         | Dense             | OGN         |
| $32^3$     | 0.33       | <b>0.29</b> | <b>0.015</b>      | 0.016       |
| $64^3$     | 0.50       | <b>0.36</b> | 0.19              | <b>0.06</b> |
| $128^3$    | 1.62       | <b>0.43</b> | 0.56              | <b>0.18</b> |
| $256^3$    | 9.98       | <b>0.54</b> | 3.89              | <b>0.64</b> |
| $512^3$    | (74.28)    | <b>0.88</b> | (41.3)            | <b>2.06</b> |

Table 1. Memory consumption and iteration time of OGN and a dense network at different output resolutions. Batch size 1.

We empirically compare the runtime and memory consumption values for a dense network and OGN, for varying output resolution. Architectures of the networks are the same as used in Section 5.4 – three fully connected layers followed by an up-convolutional decoder. We performed the measurements on an NVidia TitanX Maxwell GPU, with 12Gb of memory. To provide actual measurements for dense networks at the largest possible resolution, we performed the comparison with batch size 1. The  $512^3$  dense network does not fit into memory even with batch size 1, so we extrapolated the numbers by fitting cubic curves.

Figure 3 and Table 1 show the results of the comparison. The OGN is roughly as efficient as its dense counterpart for low resolutions, but as the resolution grows, it gets drastically faster and consumes far less memory. At  $512^3$  voxel resolution, the OGN consumes almost two orders of magnitude less memory and runs 20 times faster. In Section A of the Appendix we provide a more detailed analysis and explicitly demonstrate the near-cubic scaling of dense networks against the near-quadratic scaling of OGNs.

To put these numbers into perspective, training OGN at  $256^3$  voxel output resolution takes approximately 5 days. Estimated training time of its dense counterpart would be almost a month. Even if the  $512^3$  voxel dense network would fit into memory, it would take many months to train.

### 5.3. Autoencoders

Autoencoders and their variants are commonly used for representation learning from volumetric data [15, 30]. Therefore, we start by comparing the representational power of the OGN to that of dense voxel grid networks on the task of auto-encoding volumetric shapes.

We used the decoder architecture described in section 5.1 both for the OGN and the dense baseline. The architecture of the encoder is symmetric to the decoder. Both encoders operate on a dense voxel grid representation<sup>5</sup>.

We trained the autoencoders on the ShapeNet-cars dataset in two resolutions:  $32^3$  and  $64^3$ . We used 80% of the data for training, and 20% for testing. Quantitative results

<sup>5</sup>In this paper, we focus on generating 3D shapes. Thus, we have not implemented an octree-based convolutional encoder. This could be done along the lines of Riegler et al. [29]

are summarized in Table 2. With predicted octree structure, there is no significant difference in performance between the OGN and the dense baseline.

| Network                | $32^3$ | $64^3$ |
|------------------------|--------|--------|
| Dense                  | 0.924  | 0.890  |
| OGN+ <i>Prop-known</i> | 0.939  | 0.904  |
| OGN+ <i>Prop-pred</i>  | 0.924  | 0.884  |

Table 2. Quantitative results for OGN and dense autoencoders. Predictions were compared with the ground truth at the corresponding resolution, without upsampling.

#### 5.3.1 Flexibility of architecture choice

To show that OGNs are not limited to up-convolutional layers with  $2^3$  filters, we evaluated two alternative  $64^3$  OGN auto-encoders: one with  $4^3$  up-convolutions and one with  $2^3$  up-convolutions interleaved with  $3^3$  convolutions. The results are summarized in Table 7. There is no significant difference between the architectures for this task. With larger filters, the network is roughly twice slower in our current implementation, so we used  $2^3$  filters in all further experiments.

#### 5.3.2 Using known structure

Interestingly, OGN with known tree structure outperforms the network based on a dense voxel grid, both qualitatively and quantitatively. An example of this effect can be seen in Figure 4: the dense autoencoder and our autoencoder with predicted propagation struggle with properly reconstructing the spoiler of the car. Intuitively, the known tree structure provides additional shape information to the decoder, thus simplifying the learning problem. In the autoencoder scenario, however, this may be undesirable if one aims to encode all information about a shape in a latent vector. In tasks like semantic segmentation, the input octree structure could help introduce shape features implicitly in the learning task.

### 5.4. 3D shape from high-level information

We trained multiple OGNs for generating shapes from high-level parameters similar to Dosovitskiy et al. [10]. In all cases the input of the network is a one-hot encoded object ID, and the output is an octree with the object shape.

| Mode                   | 2x2 filters | 4x4 filters | IntConv |
|------------------------|-------------|-------------|---------|
| OGN+ <i>Prop-known</i> | 0.904       | 0.907       | 0.907   |
| OGN+ <i>Prop-pred</i>  | 0.884       | 0.885       | 0.885   |

Table 3. Using more complex architectures in  $64^3$  OGN autoencoders does not lead to significant performance improvements.

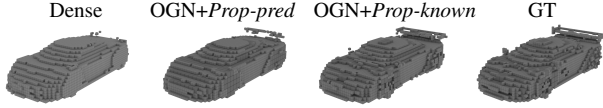


Figure 4. Using the known tree structure at test time leads to improved performance.

|               | 64    | 128          | 256   | 512          |
|---------------|-------|--------------|-------|--------------|
| ShapeNet-cars | 0.856 | <b>0.901</b> | 0.865 | -            |
| BlendSwap     | 0.535 | 0.649        | 0.880 | <b>0.969</b> |

Table 4. Quantitative evaluation of 3D shapes generated from high-level information. Lower-resolution predictions from ShapeNet-cars were upsampled to 256<sup>3</sup> ground truth, scenes from BlendSwap — to 512<sup>3</sup>.

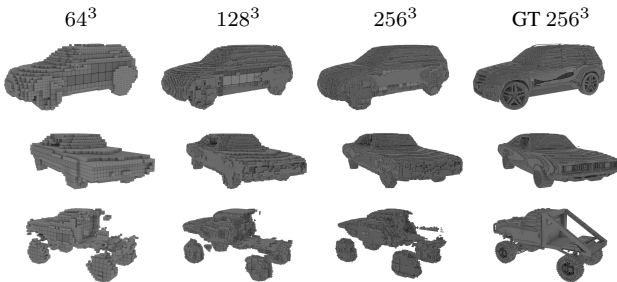


Figure 5. Training samples from the ShapeNet-cars dataset generated by our networks. Cells at different octree levels vary in size and are displayed in different shades of gray.

#### 5.4.1 ShapeNet-cars

First, we trained on the whole ShapeNet-cars dataset in three resolutions: 64<sup>3</sup>, 128<sup>3</sup> and 256<sup>3</sup>. Example outputs are shown in Figure 5 and quantitative results are presented in Table 4. Similar to the two-dimensional case [10], the outputs are accurate in the overall shape, but lack some fine details. This is not due to the missing resolution, but due to general limitations of the training data and the learning task. Table 4 reveals that a resolution of 128<sup>3</sup> allows the reconstruction of a more accurate shape with more details than a resolution of 64<sup>3</sup>. At an even higher resolution of 256<sup>3</sup>, the overall performance decreased again. Even though the higher-resolution network is architecturally capable of performing better, it is not guaranteed to train better. Noisy gradients from outer high-resolution layers may hamper learning of deeper layers, resulting in an overall decline in performance. This problem is orthogonal to the issue of designing computationally efficient architectures, which we aim to solve in this paper. We further discuss this in the Appendix.

Notably, the network does not only learn to generate objects from the training dataset, but it can also generalize to unseen models. We demonstrate this by interpolating between

pairs of one-hot input ID vectors. Figure 6 shows that for all intermediate input values the network produces consistent output cars, with the style being smoothly changed between the two training points.

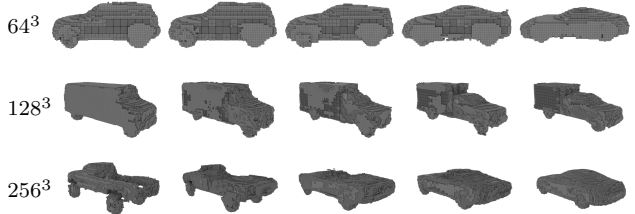


Figure 6. Our networks can generate previously unseen cars by interpolating between the dataset points, which demonstrates their generalization capabilities.

#### 5.4.2 BlendSwap

To additionally showcase the benefit of using higher resolutions, we trained OGNs to fit the BlendSwap dataset containing 4 whole scenes. In contrast to the ShapeNet-cars dataset, such amount of training data does not allow for any generalization. The experiment aims to show that OGNs provide sufficient resolution to represent such high-fidelity shape data.

Figure 7 shows the generated scenes. In both examples, 64<sup>3</sup> and 128<sup>3</sup> resolutions are inadequate for representing the details. For the bottom scene, even the 256<sup>3</sup> resolution still struggles with fine-grained details. This example demonstrates that tasks like end-to-end learning of scene reconstruction requires high-resolution representations, and the OGN is an architecture that can provide such resolutions.

These qualitative observations are confirmed quantitatively in Table 4. Higher output resolutions allow for more accurate reconstruction of the samples in the dataset. More results for this experiment are shown in Section D of the Appendix, and the accompanying video<sup>6</sup>.

#### 5.5. Single-image 3D reconstruction

In this experiment we trained networks with our OGN decoder on the task of single-view 3D reconstruction. To demonstrate that our dense voxel grid baseline, as already used in the autoencoder experiment, is a strong baseline, we compare to the approach by Choy et al. [6]. This approach operates on 32<sup>3</sup> voxel grids, and we adopt this resolution for our first experiment. To ensure a fair comparison, we trained networks on ShapeNet-all, the exact dataset used by Choy et al. [6]. Following the same dataset splitting strategy, we used 80% of the data for training, and 20% for testing. As a baseline, we trained a network with a dense

<sup>6</sup><https://youtu.be/kmMvKNNyYF4>

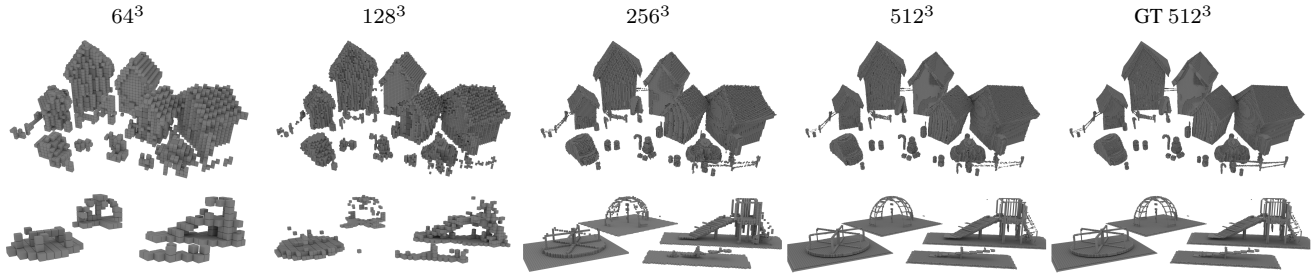


Figure 7. OGN is used to reproduce large-scale scenes from the dataset, where high resolution is crucial to generate fine-grained structures.

| Category   | R2N2 [6]     | OGN          | Dense        |
|------------|--------------|--------------|--------------|
| Plane      | 0.513        | <b>0.587</b> | 0.570        |
| Bench      | 0.421        | <b>0.481</b> | 0.481        |
| Cabinet    | 0.716        | 0.729        | <b>0.747</b> |
| Car        | 0.798        | 0.816        | <b>0.828</b> |
| Chair      | 0.466        | <b>0.483</b> | 0.481        |
| Monitor    | 0.468        | 0.502        | <b>0.509</b> |
| Lamp       | 0.381        | <b>0.398</b> | 0.371        |
| Speaker    | <b>0.662</b> | 0.637        | 0.650        |
| Firearm    | 0.544        | <b>0.593</b> | 0.576        |
| Couch      | 0.628        | 0.646        | <b>0.668</b> |
| Table      | 0.513        | 0.536        | <b>0.545</b> |
| Cellphone  | 0.661        | <b>0.702</b> | 0.698        |
| Watercraft | 0.513        | <b>0.632</b> | 0.550        |
| Mean       | 0.560        | <b>0.596</b> | 0.590        |

Table 5. Single-view 3D reconstruction results on the  $32^3$  version of ShapeNet-all from Choy et al. [6] compared to OGN and a dense baseline. OGN is competitive with voxel-grid-based networks.

decoder which had the same configuration as our OGN decoder. Table 5 shows that compared to single-view reconstructions from [6], both the OGN and the baseline dense network compare favorably for most of the classes. In conclusion, the OGN is competitive with voxel-grid-based networks on the complex task of single-image class-specific 3D reconstruction.

We also evaluated the effect of resolution on the ShapeNet-cars dataset. Figure 8 shows that OGNs learned to infer the 3D shapes of cars in all cases, and that high-resolution predictions are clearly better than the  $32^3$  models commonly used so far. This is backed up by quantitative results shown in Table 6:  $32^3$  results are significantly worse than the rest. At  $256^3$  performance drops again for the same reasons as in the decoder experiment in section 5.4.1.

## 6. Conclusions

We have presented a novel convolutional decoder architecture for generating high-resolution 3D outputs represented as octrees. We have demonstrated that this architecture is flexible in terms of the exact layer configuration, and

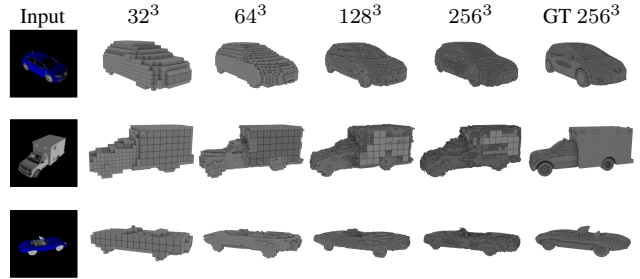


Figure 8. Single-image 3D reconstruction on the ShapeNet-cars dataset using OGN in different resolutions.

| Resolution     | 32    | 64    | 128          | 256   |
|----------------|-------|-------|--------------|-------|
| Single-view 3D | 0.641 | 0.771 | <b>0.782</b> | 0.766 |

Table 6. Single-image 3D reconstruction results on ShapeNet-cars. Low-resolution predictions are upsampled to  $256^3$ . Commonly used  $32^3$  models are significantly worse than the rest.

that it provides the same accuracy as dense voxel grids in low resolution. At the same time, it scales much better to higher resolutions, both in terms of memory and runtime.

This architecture enables end-to-end deep learning to be applied to tasks that appeared unfeasible before. In particular, learning tasks that involve 3D shapes, such as 3D object and scene reconstruction, are likely to benefit from it.

While in this paper we have focused on shapes and binary occupancy maps, it is straightforward to extend the framework to multi-dimensional outputs attached to the octree structure; for example, the output of the network could be a textured shape or a signed distance function. This will allow for an even wider range of applications.

## Acknowledgements

This work was supported by the Excellence Initiative of the German Federal and State Governments: BIOS Centre for Biological Signalling Studies (EXC 294). We would like to thank Benjamin Ummenhofer for valuable discussions and technical comments. We also thank Nikolaus Mayer for his help with 3D model visualization and manuscript preparation.



## References

- [1] F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *CVPR*, 2014. 11
- [2] F. Calakli and G. Taubin. SSD: Smooth Signed Distance Surface Reconstruction. *Computer Graphics Forum*, 2011. 2
- [3] Ö. Çiçek, A. Abdulkadir, S. Lienkamp, T. Brox, and O. Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. In *MICCAI*, 2016. 2
- [4] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. *CoRR*, abs/1512.03012, 2015. 5
- [5] H. Chen, Q. Dou, L. Yu, and P. Heng. Voxresnet: Deep voxelwise residual networks for volumetric brain segmentation. *CoRR*, abs/1608.05895, 2016. 2
- [6] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *ECCV*, 2016. 2, 5, 7, 8
- [7] C. Connolly. Cumulative generation of octree models from range data. In *ICRA*, 1984. 2
- [8] X. Di, R. Dahyot, and M. Prasad. Deep shape from a low number of silhouettes. In *ECCV Workshops*, 2016. 2
- [9] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *ICCV*, 2015. 1
- [10] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox. Learning to generate chairs, tables and cars with convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016. 1, 6, 7
- [11] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. *CoRR*, abs/1612.00603, 2016. 2
- [12] S. Fuhrmann and M. Goesele. Fusion of depth maps with multiple scales. In *SIGGRAPH Asia*, 2011. 2
- [13] M. Gadelha, S. Maji, and R. Wang. 3d shape induction from 2d views of multiple objects. *CoRR*, abs/1612.05872, 2016. 2
- [14] I. Gargantini. Linear octrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, 20(4):365 – 374, 1982. 2
- [15] R. Girdhar, D. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation for objects. In *ECCV*, 2016. 2, 6
- [16] B. Graham. Spatially-sparse convolutional neural networks. *CoRR*, abs/1409.6070, 2014. 2
- [17] B. Graham. Sparse 3d convolutional neural networks. In *BMVC*, 2015. 2
- [18] E. Grant, P. Kohli, and M. van Gerven. Deep disentangled representations for volumetric reconstruction. In *ECCV Workshops*, 2016. 2
- [19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014. 4
- [20] D. Jimenez Rezende, S. M. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess. Unsupervised learning of 3d structure from images. In *NIPS*. 2016. 2
- [21] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 1
- [22] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *SGP*, 2006. 2
- [23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014. 5
- [24] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vnderghynst. Geodesic convolutional neural networks on riemannian manifolds. In *ICCV Workshops*, 2015. 2
- [25] D. Meagher. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. Technical Report IPL-TR-80-111, 1980. 2
- [26] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 1
- [27] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016. 2
- [28] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. 1
- [29] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. *CoRR*, abs/1611.05009, 2016. 2, 6
- [30] A. Sharma, O. Grau, and M. Fritz. Vconv-dae: Deep volumetric shape learning without object labels. In *ECCV Workshops*, 2016. 2, 6
- [31] A. Sinha, J. Bai, and K. Ramani. Deep learning 3d shape surfaces using geometry images. In *ECCV*, 2016. 2
- [32] F. Steinbrücker, J. Sturm, and D. Cremers. Volumetric 3d mapping in real-time on a cpu. In *ICRA*, 2014. 2
- [33] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3d models from single images with a convolutional network. In *ECCV*, 2016. 2
- [34] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik. Learning shape abstractions by assembling volumetric primitives. *CoRR*, abs/1612.00404, 2016. 2
- [35] B. Ummenhofer and T. Brox. Global, dense multiscale reconstruction for a billion points. In *ICCV*, 2015. 2
- [36] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NIPS*, 2016. 2
- [37] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 2
- [38] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *NIPS*, 2016. 2
- [39] L. Yi, H. Su, X. Guo, and L. J. Guibas. Syncspecnn: Synchronized spectral CNN for 3d shape segmentation. *CoRR*, abs/1612.00606, 2016. 2
- [40] M. E. Yumer and N. J. Mitra. Learning semantic deformation flows with 3d convolutional networks. In *ECCV*, 2016. 2

## Appendix

### A. Computational efficiency

In the main paper we have shown that with a practical architecture our networks scale much better than their dense counterparts both in terms of memory consumption and computation time. The numbers were obtained for the "houses" scene from the BlendSwap dataset.

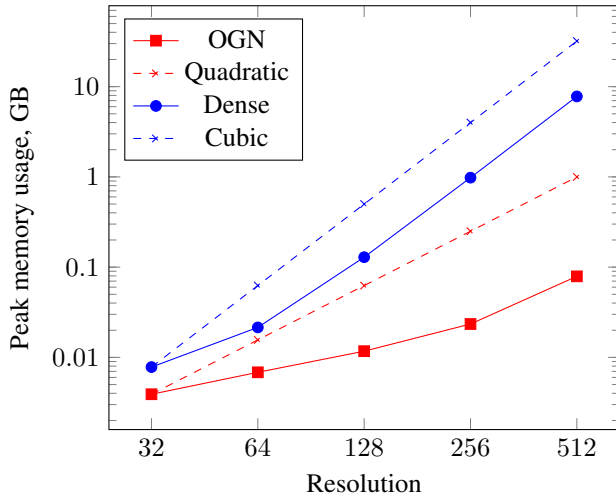


Figure 9. Memory consumption for very slim networks, forward and backward pass, batch size 1. Shown in log-log scale - lines with smaller slope correspond to better scaling.

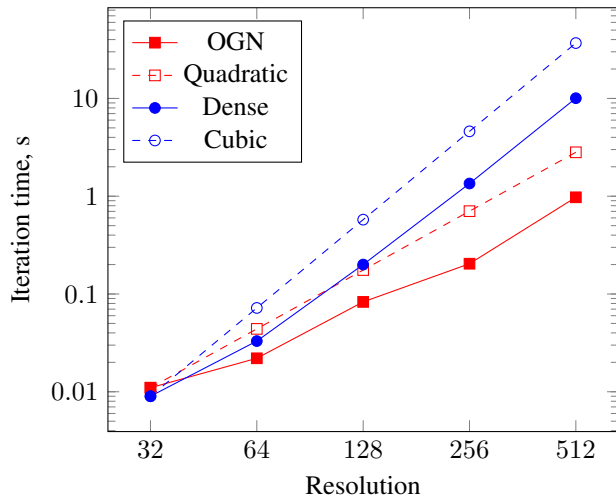


Figure 10. Iteration time for very slim networks, forward and backward pass, batch size 1. Shown in log-log scale - lines with smaller slope correspond to better scaling.

In order to further study this matter, we have designed a set of slim decoder networks that fit on a GPU in every resolution, including  $512^3$ , both with an OGN and a dense

representation. The architectures of those networks are similar to those from Table 13, but with only 1 channel in every convolutional layer, and a single fully-connected layer with 64 units in the encoder. The resulting measurements are shown in Figure 9 for memory consumption and Figure 10 for runtime. To precisely quantify the scaling, we subtracted the constant amount of memory reserved on a GPU by *caffe* (190 MB) from all numbers.

Both plots are displayed in the log-log scale, i.e., functions from the family  $y = ax^k$  are straight lines. The slope of this line is determined by the exponent  $k$ , and the vertical shift by the coefficient  $a$ . In this experiment we are mainly interested in the slope, that is, how do the approaches scale with increasing output resolution. As a reference, we show dashed lines corresponding to perfect cubic and perfect quadratic scaling.

Starting from  $64^3$  voxel resolution both the runtime and the memory consumption scale almost cubically in case of dense networks. For this particular example, OGN scales even better than quadratically, but in general scaling of the octree-based representation depends on the specific data it is applied to.

### B. Train/test modes

In Section 4.4 of the main paper, we described how we use the two propagation modes (*Prop-known* and *Prop-pred*) during training and testing. Here we motivate the proposed regimes, and show additional results with other combinations of propagation modes.

When the structure of the output tree is not known at test time, we train the networks until convergence with *Prop-known*, and then additionally fine-tune with *Prop-pred* - line 4 in Table 7. Without this fine-tuning step (line 2), there is a decrease in performance, which is more significant when using larger convolutional filters. Intuitively, this happens because the network has never seen erroneous propagations during training, and does not now how to deal with them at test time.

When the structure of the output is known at test time, the best strategy is to simply train in *Prop-known*, and test the same way (line 1). Additional fine-tuning in the *Prop-pred* mode slightly hurts performance in this case (line 3). The overall conclusion is not surprising: the best results are obtained when training networks in the same propagation modes, in which they are later tested.

### C. Feature propagation

In the main paper we mentioned that the number of features propagated by an *OGNProp* layer depends on the sizes of the convolutional filters in all subsequent blocks. In case of  $2^3$  up-convolutions with stride 2, which were used in most of our experiments, no neighboring features need to

| Training     | Testing      | $2^3$ filters | $4^3$ filters | IntConv |
|--------------|--------------|---------------|---------------|---------|
| <i>Known</i> | <i>Known</i> | 0.904         | 0.907         | 0.907   |
| <i>Known</i> | <i>Pred</i>  | 0.862         | 0.804         | 0.823   |
| <i>Pred</i>  | <i>Known</i> | 0.898         | 0.896         | 0.897   |
| <i>Pred</i>  | <i>Pred</i>  | 0.884         | 0.885         | 0.885   |

Table 7. Reconstruction quality for autoencoders with different decoder architectures:  $2^3$  up-convolutions,  $4^3$  up-convolutions, and  $2^3$  up-convolutions interleaved with  $3^3$  convolutions, using different configurations of *Prop-known* and *Prop-pred* propagation modes.

be propagated. This situation is illustrated in Figure 11-A in a one-dimensional case. Circles correspond to cells of an octree. The green cell in the input is the only one for which the value was predicted to be "mixed". Links between the circles indicate which features of the input are required to compute the result of the operation (convolution or up-convolution) for the corresponding output cell. In this case, we can see that the output cells in the next level are only affected by their parent cell from the previous level.

A more general situation is shown in Figure 11-B. The input is processed with an up-convolutional layer with  $4^3$  filters and stride 2, which is followed by a convolutional layer with  $3^3$  filters and stride 1. Again, only one cell was predicted to be "mixed", but in order to perform convolutions and up-convolutions in subsequent layers, we additionally must propagate some of its neighbors (marked red). Therefore, with this particular filter configuration, two cells in the output are affected by four cells in the input.

Generally, the number of features that should be propagated by each *OGNProp* layer is automatically calculated based on the network architecture before starting the training.

## D. 3D shape from high-level information: additional experiments

### D.1. MPI-FAUST

To additionally showcase the benefit of using higher resolutions, we trained OGNs to fit the MPI-FAUST dataset [1]. It contains 300 high-resolution scans of human bodies of 10 different people in 30 different poses. Same as with the BlendSwap, the trained networks cannot generalize to new samples due to the low amount of training data.

Figure 12 and Table 8 demonstrate qualitative and quantitative results respectively. Human models from MPI-FAUST include finer details than cars from ShapeNet, and therefore benefit from the higher resolution.

### D.2. Fitting reduced ShapeNet-cars

To better understand the performance drop at  $256^3$  resolution observed in section 5.4.1 of the main paper, we

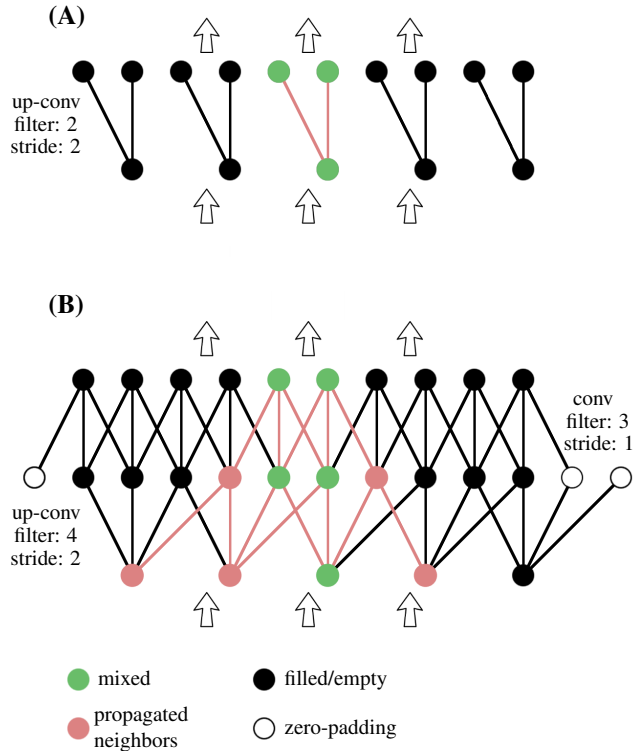


Figure 11. The *OGNProp* layer propagates the features of "mixed" cells together with the features of the neighboring cells required for computations in subsequent layers. We show the number of neighbors that need to be propagated in two cases:  $2^3$  up-convolutions (A), and  $4^3$  up-convolutions followed by  $3^3$  convolutions (B). Visualized in 1D for simplicity.

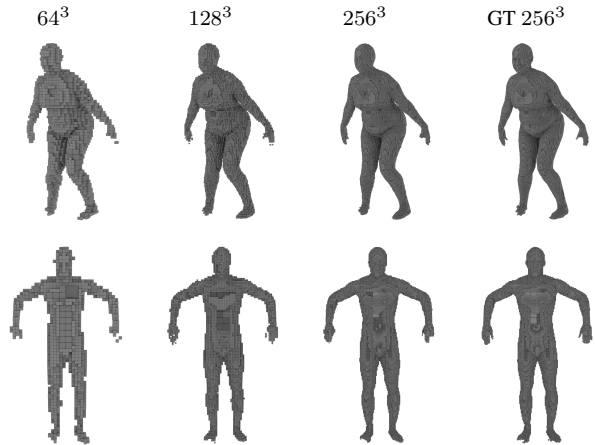


Figure 12. Training samples from the FAUST dataset reconstructed by OGN.

performed an additional experiment on the ShapeNet-Cars dataset. We trained an OGN for generating car shapes from their IDs on a reduced version of ShapeNet-Cars, including

|       |       |              |
|-------|-------|--------------|
| 64    | 128   | 256          |
| 0.890 | 0.933 | <b>0.969</b> |

Table 8. 3D shape from high-level information on the FAUST dataset. Lower-resolution predictions were upsampled to  $256^3$  ground truth.

| Dataset                | $128^3$ | $256^3$ |
|------------------------|---------|---------|
| Shapenet-cars (full)   | 0.901   | 0.865   |
| Shapenet-cars (subset) | 0.922   | 0.931   |

Table 9. There is no drop in performance in higher resolution, when training on a subset of the Shapenet-cars dataset.

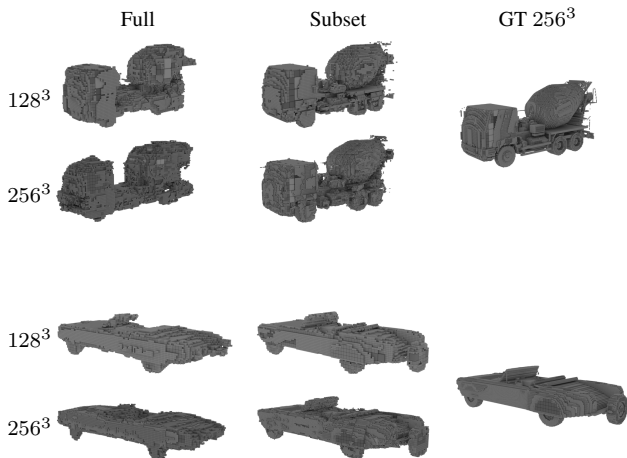


Figure 13. When training on a subset of the Shapenet-cars dataset, higher resolution models contain more details.

just 500 first models from the dataset. Quantitative results for different resolutions, along with the results for the full dataset, are shown in Table 9. Interestingly, when training on the reduced dataset, high resolution is beneficial. This is further supported by examples shown in Figure 13 – when training on the reduced dataset, the higher-resolution model contain more fine details. Overall, these results support our hypothesis that the performance drop at higher resolution is not due to the OGN architecture, but due to the difficulty of fitting a large dataset at high resolution.

## E. Shift invariance

The convolution operation on a voxel grid is perfectly shift invariant by design. This is no longer true for convolutions on octrees: a shift by a single pixel in the original voxel grid can change the structure of the octree significantly. To study the effect of shifts, we trained two fully convolutional autoencoders - one with an OGN decoder, and one with a dense decoder - on  $64^3$  models, with lowest feature map resolution  $4^3$  (so the networks should be

perfectly invariant to shifts of 16 voxels). Both were trained on non-shifted Shapenet-Cars, and tested in the *Prop-pred* mode on models shifted by a different number of voxels along the z-axis. The results are summarized in Table 10.

| Shift (voxels) | OGN   | Dense |
|----------------|-------|-------|
| 0              | 0.935 | 0.932 |
| 1              | 0.933 | 0.93  |
| 2              | 0.929 | 0.925 |
| 4              | 0.917 | 0.915 |
| 8              | 0.906 | 0.904 |

Table 10. Fully-convolutional networks tested on shifted data. Even though not shift invariant by design, OGN shows robust performance.

There is no significant difference between OGN and the dense network. A likely reason is that different training models have different octree structures, which acts as an implicit regularizer. The network learns the shape, but remains robust to the exact octree structure.

## F. Network architectures

In this section, we provide the exact network architectures used in the experimental evaluations.

### F.1. Autoencoders

The architectures of OGN autoencoders are summarized in Table 12. For the dense baselines, we used the same layer configurations with usual convolutions instead of *OGN-Conv*, and predictions being made only after the last layer of the network. All networks were trained with batch size 16.

### F.2. 3D shape from high-level information

OGN decoders used on the Shapenet-cars dataset are shown in Table 13. Encoders consisted of three fully-connected layers, with output size of the last encoder layer being identical to the input size of the corresponding decoder.

For FAUST and BlendSwap the  $256^3$  output octrees had four levels, not five like those in Table 13. Thus, the dense block had an additional deconvolution-convolution layer pair instead of one octree block. The  $512^3$  decoder on BlendSwap had one extra octree block with 32 output channels.

All  $64^3$  and  $128^3$  networks were trained with batch size 16,  $256^3$  — with batch size 4,  $512^3$  — with batch size 1.

### F.3. Single-image 3D reconstruction

In this experiment we again used decoder architectures shown in Table 13. The architecture of the convolutional encoder is shown in Table 11. The number of channels in

the last encoder layer was set identical to the number of input channels of the corresponding decoder.

|                             |
|-----------------------------|
| $[137 \times 137 \times 3]$ |
| Conv ( $7 \times 7$ )       |
| $[69 \times 69 \times 32]$  |
| Conv ( $3 \times 3$ )       |
| $[35 \times 35 \times 32]$  |
| Conv ( $3 \times 3$ )       |
| $[18 \times 18 \times 64]$  |
| Conv ( $3 \times 3$ )       |
| $[9 \times 9 \times 64]$    |
| Conv ( $3 \times 3$ )       |
| $[5 \times 5 \times 128]$   |
| FC                          |
| $[1024]$                    |
| FC                          |
| $[1024]$                    |
| FC                          |
| $[4^3 \times c]$            |

Table 11. Convolutional encoder used in the single-image 3D reconstruction experiment.

| $32^3$   | $64^3$ ( $2^3$ filters)  | $64^3$ ( $4^3$ filters)  | $64^3$ (InvConv)  |
|--|--|--|---|
|  |  | $[64^3 \times 1]$  |   |
|  |  | Conv ( $3^3$ )<br>$[32^3 \times 32]$   |   |
| $[32^3 \times 1]$  |  | Conv ( $3^3$ )<br>$[16^3 \times 48]$   |   |
| Conv ( $3^3$ )<br>$[16^3 \times 32]$   |  | Conv ( $3^3$ )<br>$[8^3 \times 64]$  |   |
| Conv ( $3^3$ )<br>$[8^3 \times 48]$  |  | Conv ( $3^3$ )<br>$[4^3 \times 80]$  |   |
| Conv ( $3^3$ )<br>$[4^3 \times 64]$  |  | FC<br>$[1024]$   |   |
| FC<br>$[1024]$   |  | FC<br>$[1024]$   |   |
| FC<br>$[1024]$   |  | FC<br>$[1024]$   |   |
| FC<br>$[4^3 \times 80]$  |  | FC<br>$[4^3 \times 96]$  |   |
| Deconv ( $2^3$ )<br>$[8^3 \times 64]$  |  | Deconv ( $2^3$ )<br>$[8^3 \times 80]$  |   |
| Conv ( $3^3$ ) $\rightarrow$ <b>11</b><br>$[8^3 \times 64]$                              |  | Conv ( $3^3$ )<br>$[8^3 \times 80]$  |   |
| <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>12</b><br>$[16^3 \times 48]$ |  | Deconv ( $2^3$ )<br>$[16^3 \times 64]$   |   |
| <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>13</b><br>$[32^3 \times 32]$ |  | Conv ( $3^3$ ) $\rightarrow$ <b>11</b><br>$[16^3 \times 64]$                             |   |
|  | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>12</b><br>$[32^3 \times 48]$ | <i>OGNProp</i><br><i>OGNConv</i> ( $4^3$ ) $\rightarrow$ <b>12</b><br>$[32^3 \times 48]$ | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ )<br>$[32^3 \times 48]$<br><i>OGNConv</i> *( $3^3$ ) $\rightarrow$ <b>12</b><br>$[32^3 \times 48]$ |
|  | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>13</b><br>$[64^3 \times 32]$ | <i>OGNProp</i><br><i>OGNConv</i> ( $4^3$ ) $\rightarrow$ <b>13</b><br>$[64^3 \times 32]$ | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ )<br>$[64^3 \times 32]$<br><i>OGNConv</i> *( $3^3$ ) $\rightarrow$ <b>13</b><br>$[64^3 \times 32]$ |

Table 12. OGN architectures used in our experiments with autoencoders. *OGNConv* denotes up-convolution, *OGNConv*\* — convolution. Layer name followed by ' $\rightarrow$  1k' indicates that level  $k$  of an octree is predicted by a classifier attached to this layer.

| <b>32<sup>3</sup></b>  | <b>64<sup>3</sup></b>  | <b>128<sup>3</sup></b>  | <b>256<sup>3</sup></b>  |
|--|--|---|---|
| $[4^3 \times 80]$  | $[4^3 \times 96]$  | $[4^3 \times 112]$  | $[4^3 \times 112]$  |
| Deconv ( $2^3$ )<br>$[8^3 \times 64]$  | Deconv ( $2^3$ )<br>$[8^3 \times 80]$  | Deconv ( $2^3$ )<br>$[8^3 \times 96]$   | Deconv ( $2^3$ )<br>$[8^3 \times 96]$   |
| Conv ( $3^3$ ) $\rightarrow$ <b>11</b><br>$[8^3 \times 64]$                              | Conv ( $3^3$ )<br>$[8^3 \times 80]$  | Conv ( $3^3$ )<br>$[8^3 \times 96]$   | Conv ( $3^3$ )<br>$[8^3 \times 96]$   |
| <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>12</b><br>$[16^3 \times 48]$ | Deconv ( $2^3$ )<br>$[16^3 \times 64]$   | Deconv ( $2^3$ )<br>$[16^3 \times 80]$  | Deconv ( $2^3$ )<br>$[16^3 \times 80]$  |
| <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>13</b><br>$[32^3 \times 32]$ | Conv ( $3^3$ ) $\rightarrow$ <b>11</b><br>$[16^3 \times 64]$                             | Conv ( $3^3$ ) $\rightarrow$ <b>11</b><br>$[16^3 \times 80]$                              | Conv ( $3^3$ ) $\rightarrow$ <b>11</b><br>$[16^3 \times 80]$                              |
|  | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>12</b><br>$[32^3 \times 48]$ | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>12</b><br>$[32^3 \times 64]$  | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>12</b><br>$[32^3 \times 64]$  |
|  | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>13</b><br>$[64^3 \times 32]$ | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>13</b><br>$[64^3 \times 48]$  | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>13</b><br>$[64^3 \times 48]$  |
|  |  | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>14</b><br>$[128^3 \times 32]$ | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>14</b><br>$[128^3 \times 32]$ |
|  |  |   | <i>OGNProp</i><br><i>OGNConv</i> ( $2^3$ ) $\rightarrow$ <b>15</b><br>$[256^3 \times 32]$ |

Table 13. OGN decoder architectures used in shape from ID, and single-image 3D reconstruction experiments.