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

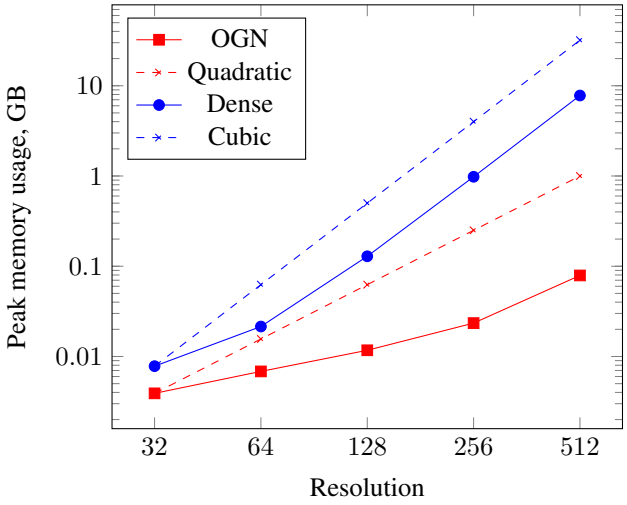

Figure 1. Memory consumption for very slim networks, forward and backward pass, batch size 1. Shown in log-log scale - lines with smaller slope correspond to better scaling.

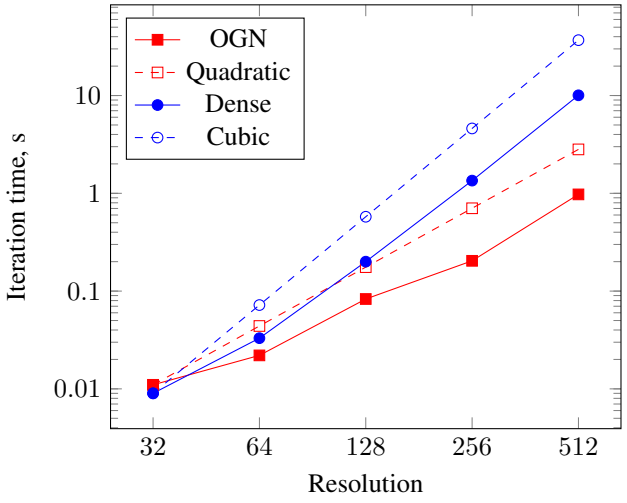

Figure 2. Iteration time for very slim networks, forward and backward pass, batch size 1. Shown in log-log scale - lines with smaller slope correspond to better scaling.

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.

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 7, 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 1 for memory consumption and Figure 2 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.

Training	Testing	2^3 filters	4^3 filters	IntConv
Known	*Known*	0.904	0.907	0.907
Known	*Pred*	0.862	0.804	0.823
Pred	*Known*	0.898	0.896	0.897
Pred	*Pred*	0.884	0.885	0.885

Table 1. 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.

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 1. 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 be propagated. This situation is illustrated in Figure 3-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 3-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.

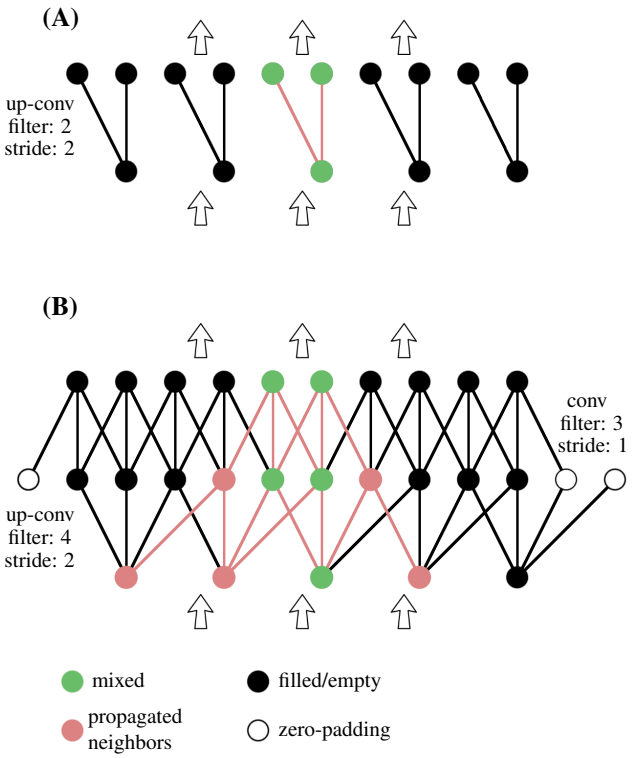

Figure 3. The *OGNProp* layer propagates the features of "mixed" cells together with the features of the neighboring cells required for computations in subsequent layers. We show the number of neighbors that need to be propagated in two cases: 2^3 up-convolutions (A), and 4^3 up-convolutions followed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

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 4 and Table 2 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.

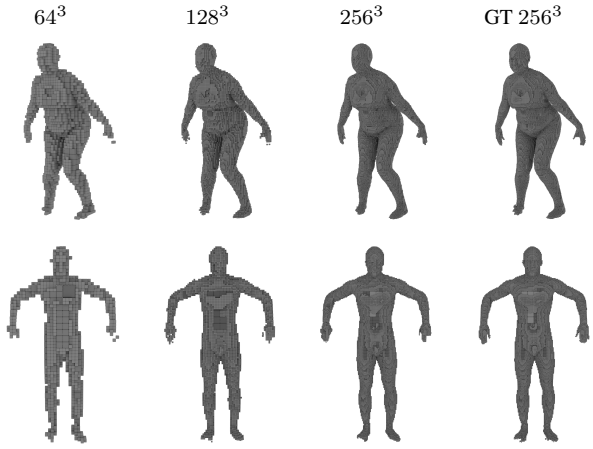

Figure 4. Training samples from the FAUST dataset reconstructed by OGN.

64	128	256
0.890	0.933	**0.969**

Table 2. 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 3. There is no drop in performance in higher resolution, when training on a subset of the Shapenet-cars dataset.

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 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 just 500 first models from the dataset. Quantitative results for different resolutions, along with the results for the full dataset, are shown in Table 3. Interestingly, when training on the reduced dataset, high resolution is beneficial. This is further supported by examples shown in Figure 5 – 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 4.

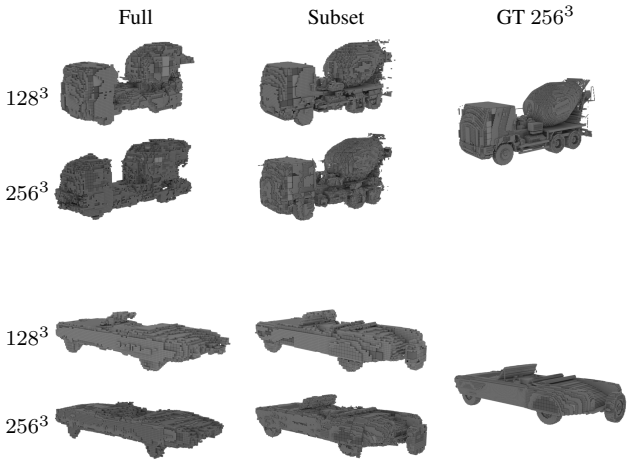

Figure 5. When training on a subset of the Shapenet-cars datset, higher resolution models contain more details.

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 4. 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 6. 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 7. 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 7. 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 7. The architecture of the convolutional encoder is shown in Table 5. 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 (7×7) $[69 \times 69 \times 32]$
Conv (3×3) $[35 \times 35 \times 32]$
Conv (3×3) $[18 \times 18 \times 64]$
Conv (3×3) $[9 \times 9 \times 64]$
Conv (3×3) $[5 \times 5 \times 128]$
FC $[1024]$
FC $[1024]$
FC $[4^3 \times c]$

Table 5. Convolutional encoder used in the single-image 3D reconstruction experiment.

References

[1] F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *CVPR*, 2014. 2

$\mathbf{32^3}$	$\mathbf{64^3}$ (2^3 filters)	$\mathbf{64^3}$ (4^3 filters)	$\mathbf{64^3}$ (InvConv)
	$[64^3 \times 1]$		
$[32^3 \times 1]$	Conv (3^3) $[32^3 \times 32]$		
Conv (3^3) $[16^3 \times 32]$	Conv (3^3) $[16^3 \times 48]$		
Conv (3^3) $[8^3 \times 48]$	Conv (3^3) $[8^3 \times 64]$		
Conv (3^3) $[4^3 \times 64]$	Conv (3^3) $[4^3 \times 80]$		
FC $[1024]$	FC $[1024]$		
FC $[1024]$	FC $[1024]$		
FC $[4^3 \times 80]$	FC $[4^3 \times 96]$		
Deconv (2^3) $[8^3 \times 64]$	Deconv (2^3) $[8^3 \times 80]$		
Conv (3^3)$\rightarrow \mathbf{l1}$ $[8^3 \times 64]$	Conv (3^3) $[8^3 \times 80]$		
OGNProp *OGNConv*$(2^3) \rightarrow \mathbf{l2}$ $[16^3 \times 48]$	Deconv (2^3) $[16^3 \times 64]$		
OGNProp *OGNConv*$(2^3) \rightarrow \mathbf{l3}$ $[32^3 \times 32]$	Conv (3^3)$\rightarrow \mathbf{l1}$ $[16^3 \times 64]$		
	OGNProp *OGNConv*$(2^3) \rightarrow \mathbf{l2}$ $[32^3 \times 48]$	*OGNProp* *OGNConv*$(4^3) \rightarrow \mathbf{l2}$ $[32^3 \times 48]$	*OGNProp* *OGNConv*(2^3) $[32^3 \times 48]$ *OGNConv*$*(3^3) \rightarrow \mathbf{l2}$ $[32^3 \times 48]$
	OGNProp *OGNConv*$(2^3) \rightarrow \mathbf{l3}$ $[64^3 \times 32]$	*OGNProp* *OGNConv*$(4^3) \rightarrow \mathbf{l3}$ $[64^3 \times 32]$	*OGNProp* *OGNConv*(2^3) $[64^3 \times 32]$ *OGNConv*$*(3^3) \rightarrow \mathbf{l3}$ $[64^3 \times 32]$

Table 6. OGN architectures used in our experiments with autoencoders. *OGNConv* denotes up-convolution, *OGNConv** — convolution. Layer name followed by '$\rightarrow \mathbf{lk}$' indicates that level k of an octree is predicted by a classifier attached to this layer.

$\mathbf{32^3}$	$\mathbf{64^3}$	$\mathbf{128^3}$	$\mathbf{256^3}$
$[4^3 \times 80]$	$[4^3 \times 96]$	$[4^3 \times 112]$	$[4^3 \times 112]$
Deconv (2^3) $[8^3 \times 64]$	Deconv (2^3) $[8^3 \times 80]$	Deconv (2^3) $[8^3 \times 96]$	Deconv (2^3) $[8^3 \times 96]$
Conv $(3^3) \rightarrow \mathbf{l1}$ $[8^3 \times 64]$	Conv (3^3) $[8^3 \times 80]$	Conv (3^3) $[8^3 \times 96]$	Conv (3^3) $[8^3 \times 96]$
OGNProp *OGNConv* $(2^3) \rightarrow \mathbf{l2}$ $[16^3 \times 48]$	Deconv (2^3) $[16^3 \times 64]$	Deconv (2^3) $[16^3 \times 80]$	Deconv (2^3) $[16^3 \times 80]$
OGNProp *OGNConv* $(2^3) \rightarrow \mathbf{l3}$ $[32^3 \times 32]$	Conv $(3^3) \rightarrow \mathbf{l1}$ $[16^3 \times 64]$	Conv $(3^3) \rightarrow \mathbf{l1}$ $[16^3 \times 80]$	Conv $(3^3) \rightarrow \mathbf{l1}$ $[16^3 \times 80]$
	OGNProp *OGNConv* $(2^3) \rightarrow \mathbf{l2}$ $[32^3 \times 48]$	*OGNProp* *OGNConv* $(2^3) \rightarrow \mathbf{l2}$ $[32^3 \times 64]$	*OGNProp* *OGNConv* $(2^3) \rightarrow \mathbf{l2}$ $[32^3 \times 64]$
	OGNProp *OGNConv* $(2^3) \rightarrow \mathbf{l3}$ $[64^3 \times 32]$	*OGNProp* *OGNConv* $(2^3) \rightarrow \mathbf{l3}$ $[64^3 \times 48]$	*OGNProp* *OGNConv* $(2^3) \rightarrow \mathbf{l3}$ $[64^3 \times 48]$
		OGNProp *OGNConv* $(2^3) \rightarrow \mathbf{l4}$ $[128^3 \times 32]$	*OGNProp* *OGNConv* $(2^3) \rightarrow \mathbf{l4}$ $[128^3 \times 32]$
			OGNProp *OGNConv* $(2^3) \rightarrow \mathbf{l5}$ $[256^3 \times 32]$

Table 7. OGN decoder architectures used in shape from ID, and single-image 3D reconstruction experiments.