Octree Generating Network: Generating 3D Shapes With High-resolution Using Sparse Volumetric Representation

Karen Yang* Kaichun Mo* Te-Lin Wu*
Stanford University
kaiyuany03@gmail.edu kaichun@cs.stanford.edu telin@stanford.edu

Abstract

We reproduce the results and extend the architecuture of Octree Generating Network (OGN), which is a deep convolutional decoder architecture capable of generating 3D voxel grids in a computational and memory-efficient manner empowered by an octree structural representation. OGN reduces the cubic time complexity when processing high resolution volumetric data using sparse representation such as hash tables, in contrast to standard dense 3D convolutional modules acts on a regular voxel grid. This allows the module to surpassed the memory issue and slow training time for standard 3D deep neural networks. OGN shows success being adopted in various architectures, such as 3D convolutional autoencoders, single-view 3D reconstruction, and shape from id reconstruction. We further extend the OGN modules to be incorporated in an adversarial nets training, and verify our modules using shape classifications. We also combine OGN and our O-CNN customized layer to perform 3D autoencoder in a even more efficient way.

1. Introduction

Deep learning have shown unbelievably strong power on tackling a lot of computer vision tasks. However, in the field of 3D vision, it is still unclear how should we represent 3D models and what neural network architectures we should use to consume these data. Nowadays, most 3D data are represented by polygon meshes or point clouds. Although some research efforts have been put on designing graph CNN [13, 26] and point cloud CNN [16, 7], it is still hard to consume these irregular data.

Instead, most researchers choose to convert these irregular data into volumetric representation. The volumetric representation is extremely friendly to the existing deep learning frameworks and it offers a natural extension of the 2D convolution operation in the 3D case. Many papers [2, 23, 3] demonstrate good performance of using 3D Convolutional Neural Network (3D-CNN). However, due to the computational time and memory limitation, most 3D-CNN approaches can only process 3D inputs of relatively low resolution, e.g. 32^3 or 64^3, which limits many 3D tasks that requires high-resolution inputs or outputs, such as scene understanding or fine-grained model analysis.

Recently, inspired by sparse deep learning, many research efforts propose to use Octree [14] as the sparse volumetric representation for 3D shapes. OctNet [18] and O-CNN [22] designs convolution, pooling and unpooling operations that can be applied on Octree sparse data. Octree Generating Network (OGN) [20] propose an approach to generate 3D shapes of higher resolution, e.g. 128^3, 256^3 or even higher, using Octree sparse volumetric representation.

In this paper, we survey the three Octree based methods, design and implement a simplified version of O-CNN method in Caffe and perform some experiments on several 3D tasks that have high resolution 3D shapes as inputs or outputs, such as 3D shape classification, image to shape reconstruction, 3D Auto-encoder and 3D-GAN. In these experiments, we use OGN as the network to generate high-resolution 3D shapes and use O-CNN as the network to consume the high-resolution 3D shapes. We provide the results that we get for this course project and discuss the future works that we need to do for the next steps.

2. Related Work

Deep Learning for 3D Data Deep neural networks have been successfully adopted in various computer vision tasks to achieve remarkable results, such as object detection / recognition, image classification, and scene understanding. Recent interest in applying deep learning algorithms to modeling volumetric representations and graphics shapes have shown further success in 3D data. A large proportion of these deep learning based methods aim to generate volumetric data based on convolutional networks with feature maps and outputs voxel grids representations. These in-

*Indicate equal contribution.
†This is the report for CS468 course project in Spring 2017.

cludes supervised or unsupervised learning such as single or multi-view object reconstruction [9, 3, 5, 25, 8], 3D generative models [24, 12], 3D semantic segmentation [4, 2], and shape deformation [27]. However, these approaches are limited by the resolution of the output voxel grids. High resolution dense 3D voxel grid is memory hungry and consumes computational power severely. The computational complexity scales in $O(N^3)$ with respect to the output size. Thus training the generative models for resolution higher than 64^3 is troublesome for contemporary GPUs. To remedy this issue, simple approaches such as reducing the batch size or generating the volume part-by-part were adopted, but still could not fundamentally solve the problem. Moreover, the significantly slow training time brings another issue for training these 3D generative networks.

Octree-Based Learning Frameworks Largely inspired by sparse convolutional networks proposed by Graham $et\ al.$ [11], which is capable of achieving efficient shape analysis by storing a sparse set of non-trivial features instead of dense feature maps. The basic idea of Octree Generating Network (OGN) [21] for solving the inverse problem is to predict which parts of the output contain high-resolution information using an Octree [15] structure, and then restrict the computations of certain octree level to only those parts of the entire output representation. Riegler $et\ al.$ [19] also learned an Octree-based convolutional networks (OctNet) to be successfully applied to various 3D vision tasks in a much more efficient way as compared to dense 3D convolutions.

Different from the OGN, OctNet generates an octree assuming that the octree structure is in hand at inference time. This holds for taaks such as semantic segmentation, where the structure of the output octree can be set to be identical to that of the input. However, there are many situations that the structure of the octree is not known in advance and thus requires to be predicted by the generative network itself. Moreover, the OGN has better flexibility since it allows an arbitrary number of levels of an octree.

Generative Adversarial Networks More recently, great interest have shown in generative models using adversarial training (GAN) [10]. In the work of [17], they successfully applied deep convolutional nerual networks to the GAN settings, achieving great success in generating realistic facial images and performed image arithematics. JiaJun $et\ al.$ leanrd a GAN empowered 3D generative networks to output realistic voxel grid of resolution 64^3 from a low-dimensional latent vector. Our Octree-based 3D GAN network is inspired by this work, and aim for incorporating more efficient module such as OGN for advesarial trainings.

3. Octree Generating Network (OGN)

In majority of deep learning approaches regarding 3D volumetric data based on CNN use voxel grids asrepresentation. However, normal fine grid representation can hardly go beyond 32^3 resolution with the cubic scaling of computational and memory requirements, thus limits a lot of interesting 3D application that requires higher resolution. By contrast, Octree generating network (OGN) is a convolutional decoder operating on octrees. Octree representation allows for efficient high-resolution 3D operations such as shape classification, shape generation in the resolution of 512^3 or even higher .

The high level idea behind octree structure is to strategically partition a three-dimensional space by recursively subdividing it into smaller eight octants. Using this data structure, a 3D model can be efficiently represented using adaptive cell size. If a large regions of space shared the same functional value or is sparse by its nature, it can be represented by a single cell in the octree structure, thus saving a lot of computation and memory compared to a fine grid coordinate system. As the finer details are still represented by the smaller size cells of the octree, no fine information is lost. The nice features of octrees thus enables an efficient and loss-less representation of high resolution volumetric 3D representation.

Moreover, by exploiting hash-table , we can efficiently store and access data elements with constant time. Each octree cell with spatial coordinates $X = (x, y, z)$ at level l is represented as an index-value pair (m, v), where v is the occupation binary value, and m, calculated from (x, l) using Z-order curves, is the index of the hash map. Thus a sparse 3D model can be represented using a set of index-values pairs $O = \{(m, v)\}$ and stored in hash map. Figure 1 shows the memory and computation friendly characteristic of hash-table implementation of octree data structure.

Octree Generating Network (OGN) is a convolutional decoder that yields an octree as output. In Tatarchenko1 et al's paper[20], their OGN first operates on dense regular grids consisting of conventional 3D conv and up-conv layers producing a feature map of $d_1 \times d_2 \times d_3 \times c$, then this feature map will be converted to octree data structure stored in a hash table. This sparse hash-table based convolutional feature map will be further processed by an arbitrary number of hash-table-based octree blocks, each producing a new feature map with resolution of $d_1\dot{2}^l \times d_2\dot{2}^l \times d_3\dot{2}^l$. Meanwhile at each level, the feature map is processed by a 1×1 filter convolution to produce an prediction output at level l of the octree. The prediction is one of the three states: "empty", "filled" or "mixed", where "empty" and "filled" are the signal values v, and "mixed"

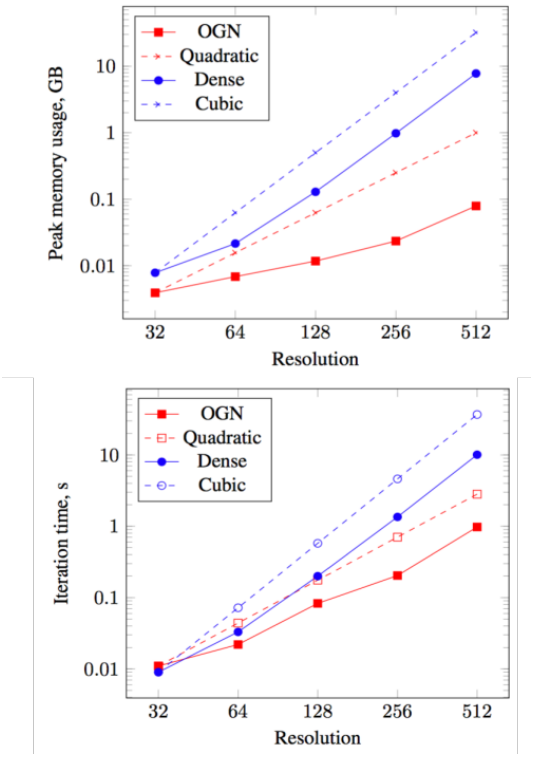

Figure 1. Top:Memory consumption for very slim networks, forward and backward pass, batch size 1. Shown in log-log scale - lines with smaller slope correspond to better scaling. Bottom: Iteration time in the same experiment setup. Taken from original Tatarchenko1 et al's paper [20]

means the state has not been determined. The prediction output at each level will be compared with the ground truth model (also converted to octree structure) and produce a three-way Softmax loss, as part of the total final loss. Finally, as long as the output resolution has not reached to the specified value, the predicted output's "mixed" state cells as well as their neighbors will be propagated to the next layer and serve as the input feature map F_l to then next OGNConv block. Figure 2 illustrates how a single block of OGN operates. For simplicity it only presented a 2D Quadtree structure in the graph, and the "mixed" state cells' neighbours are not shown. Figure 3 explains how previous predicted "mixed" state cells as well as their neighbours are propagated as feature map into the subsequent layer.

4. OGN Experiments

To better familiarize and understand the operations of the original OGN implementation, we first reproduce two of the convolutions decoder experiments presented in the original paper, Reconstruction from Shape ID 4.1, and Reconstruction from Single Image 4.2. The expriments all use known structures that provides additional shape information to the decoder, thus simplifying the learning problem. Instead of training on the ShapeNet-cars dataset, we trained on different categories of models from ShapeNet and converted them into the ground truth octree data format. We them tune the original model to produce 128^3 voxel outputs resolution.

We using Intersection over Union (IoU) on 3D voxel occupation values in comparison to the grouth truth model as our evaluation metric.

$$IoU = \frac{\sum_{i,j,k} I[(pred_{(i,j,k)} = `filled')y_{i,j,k}]}{\sum_{i,j,k} I[(pred_{(i,j,k)} = `filled') + y_{i,j,k}]}$$

4.1. Reconstruction from Shape ID

We trained OGN for generating object shapes from high-level parameters; model architecture is similar to the original paper [20] and Dosovitskiy et al [6]. The input of the model is onehot encoded shape ID, the ouput is predicted object shape with 128^3 resolution. We ran this experiment on ShapeNet-couch and ShapeNet-planes.

Figure 4 shows the OGN predicted output of planes in 4 OGN layers. Figure 5 shows the OGN predicted output on more plane models with comparison of the ground truth. The qualitative results show that overall the outputs are accurate in comparison to the groud truth model, with lack of some fine details. Similarly Figure 6 shows the qualitative results. The quantitative result, Intersection over Union for these are 0.68 for ShapeNet-planes and 0.5 on ShapeNet-couch.

4.2. Reconstruction from Single Image

For the Reconstruction from Single Image, we train the OGN model on 32^3 ShapeNet-couches and 128^3 ShapeNet-planes. For each shape, we use 24 low-resolution 137×137 rendered images to be the input data. Figure 5 and Figure 8 shows the qualitative results. Despite the input image is small has even lower resolution than the model output, the model is able to capture the overall shapes.

5. Octree-based CNN (O-CNN)

Octree is a powerful tool to represent sparse data. In the above sections, we have seen its power on generating high-resolutional 3D shapes with much less computational expense, in terms of both time and memory. In this section, we discuss how can we design neural network architectures that can consume octree-represented 3D shapes. Recent works [18] and [22] explore many possibilities of the network de-

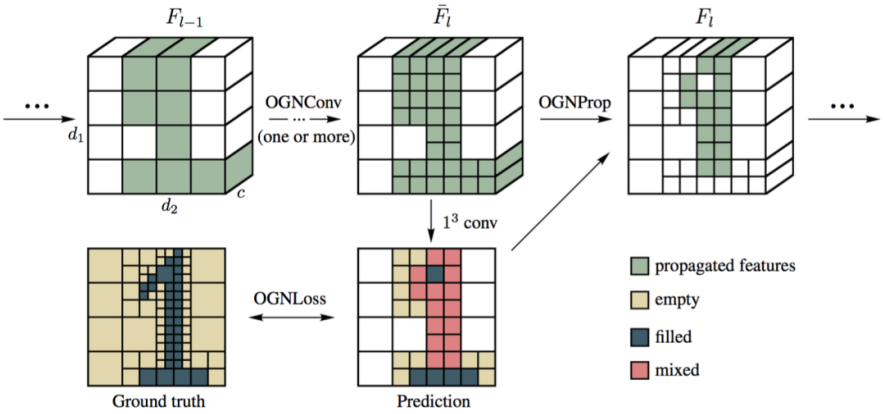

Figure 2. Single block of an OGN illustrated as 2D Quadtree for simplicity. After convolving features F_{l1} of the previous block with weight filters, it directly predicts the occupancy values of cells at level l using 1^3 convolutions. Features corresponding to filled and empty cells are no longer needed and thus not propagated. In this illustration, the propagated neighbours are not shown. [20]

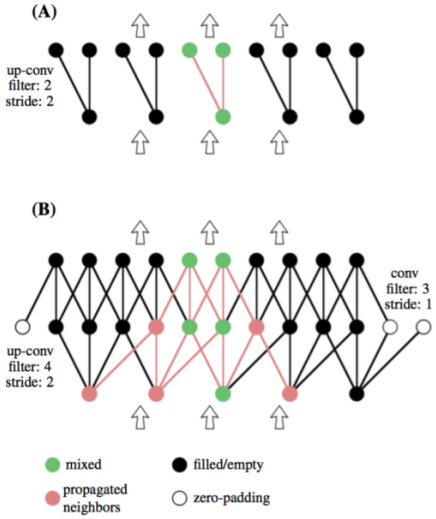

Figure 3. The OGNProp layer propagates the features of mixed cells together with the features of the neighboring cells required for computations in subsequent layers.Visualized in 1D for simplicity. [20]

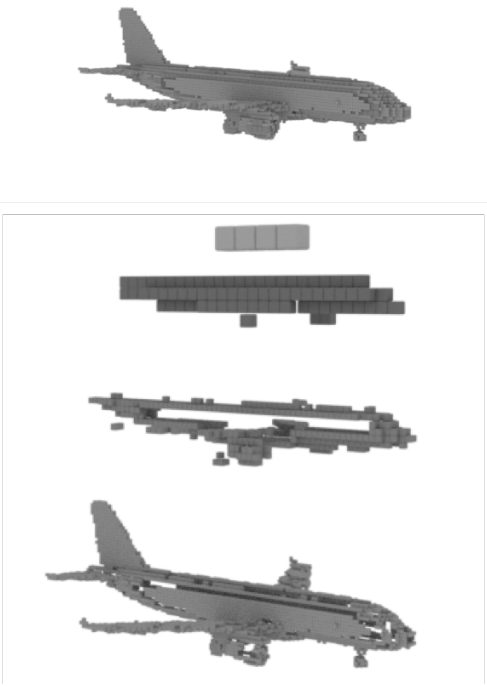

Figure 4. An illustration of OGN output in Reconstruction from Shape ID experiment, trained with 128^3 Shapenet-planes model. The top image is the entire output shape, the following four images show each layer of the Octree representation the final output, each indicating the predicted filled state cells in their respective cell size.

sign along this research direction and they successfully produce state-of-the-art results on 3D shape classification, segmentation and view estimation tasks. We survey the two papers in Section 5.1.

We implement a simplified version of Octree-CNN (O-CNN) [22] in Caffe, which applies the 2x2x2 sparse convolution operation on each level of the octree and gradually propagate the features from leaf level to the root level, which is finally served as the final global descriptor for the given shape. In section 5.2, we discuss our simplified O-CNN architecture in details. And then, we discuss the implementation and usage of our customized Caffe layers in Section 5.3.

Having both Octree Generating Network (OGN) and Octree-CNN (O-CNN) in hand, we can perform a lot of other interesting applications that are not discussed in both [22] and [20]. We can connect O-CNN and OGN together and train a Octree-based 3D shape auto-encoder to

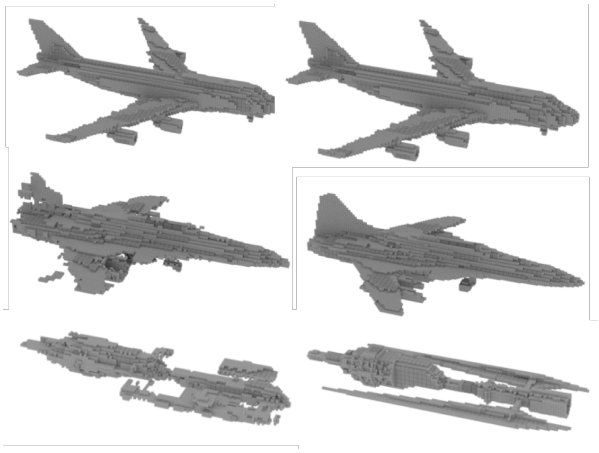

Figure 5. [Left] Output of OGN in Reconstruction from Shape ID experiment using 128^3 Shapenet-planes models.[Right] Original ground truth models. The third example has more missing voxels, due to a lack of representation of that shape in the dataset. Final mean IoU on plane dataset is 0.68.

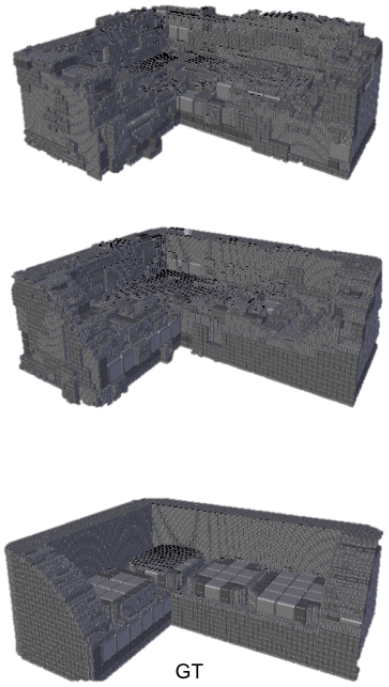

Figure 6. Output of OGN in Reconstruction from Shape ID experiment using 128^3 Shapenet-couch models.The top two images are output result from 10000 iterations training and15 training, respectively. The bottom image is the groud truth model. While the model successfully capture the overall shape of the original model, there is still lack some of the fine details on the couch surface, which might due to the general limitations of the training data and the learning task. Final mean IoU on plane dataset is 0.5.

learn latent code from 3D shapes of high-resolution. Using OGN as generator and O-CNN as discriminator, we can

Figure 7. Output of OGN in Reconstruction from Single Image experiment using 128^3 Shapenet-plane models. In this case, after training for 28000 iterations, if given an ideal angle (rendered imaged that includes most of the details of a plane), the model is then able to output 3D models that capture overall shape and some details, despite input image is relatively small and low resolution (137×137).

train Generative Adversarial Network [10] to generate high-resolutional 3D shapes represented by Octree sparse data structure. We discuss our progress in performing these experiments in Section 6.

5.1. O-CNN and OctNet: A Brief Survey

There are two recent research efforts on designing time and memory efficient neural network architectures to consume Octree-represented volumetric 3D data. Riegler et.al. [18] proposes a method on how to apply convolution, pooling and unpooling operations on the sparse Octree-based 3D volumetric data. The paper uses the bit-encoding technique to flatten the Octree representation and designs an efficient way to perform dense convolution on the sparse data. The key component is how to reduce duplicate computation for the voxels that have the same features. The implementation is released to the public in Torch. Since the implementation is complicated and not compatible with Caffe, it is hard for us to directly use the code.

Wang et.al. [22] introduces O-CNN architecture that allows 3D convolution, pooling and unpooling operations on the sparse Octree data. Similar to the idea of building up

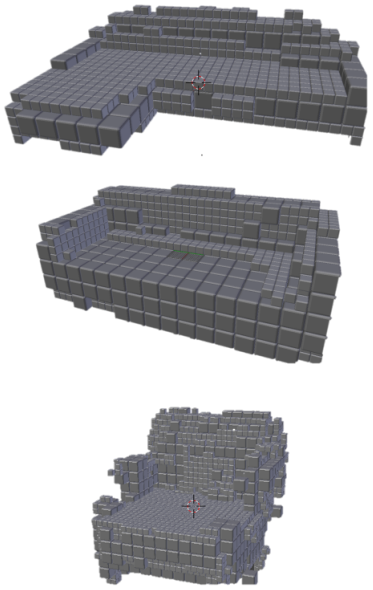

Figure 8. Output of OGN Reconstruction from Single Image experiment using 32^3 ShapeNet-couch models. After training for 41000 iterations. The OGN network is able to output accurate shapes from all different angles.

a HashTable in OGN paper [20], O-CNN also leverages HashTable to store and index the sparse feature maps. Then, 3D convolution, pooling and unpooling are naturally extended to operations on the sparse feature maps. The key difference from OctNet is that O-CNN works on surface voxels instead of filled solid voxel data. This makes the convolution on each level independent to the other levels, since every surface points reside in the leaf nodes and the feature maps are generated level by level from the leaf level to the root level. The approach is quite similar to the idea of Recursive Neural Network in that features are computed level by level from bottom to top.

Due to the complexity of OctNet implementation, we choose to implement a simplified version of O-CNN. We constrict the convolution operation to have $2 \times 2 \times 2$ kernel size and $2 \times 2 \times 2$ stride sizie so that it is applicable on our filled solid volumetric data.

5.2. Our Simplified O-CNN Architecture

In our experiments, we implement a simplified version of O-CNN that we only allow the convolution operation of $2 \times 2 \times 2$ kernel size and $2 \times 2 \times 2$ stride size, to deal with the fact that our data is filled solid volumetric data instead of surface voxels. Due to an intrinsic property of Octree data that each level is a $2 \times 2 \times 2$ sub-division of the parent level, we can perform the $2 \times 2 \times 2$ convolution operation with $2 \times 2 \times 2$ strides naturally just as what O-CNN does. Since it is guaranteed that a voxel in parent level is sub-divided

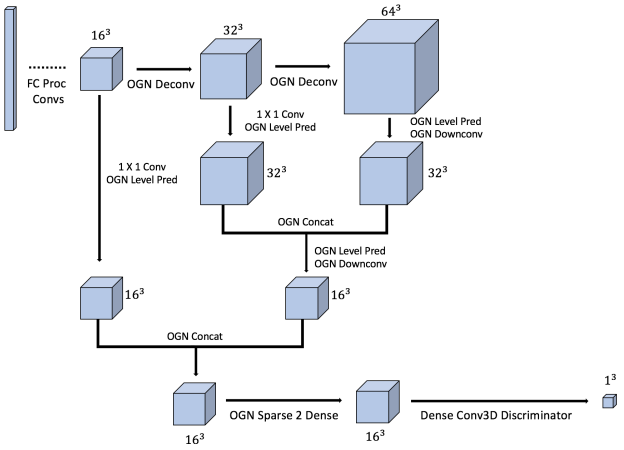

Figure 9. The OGN-based discriminator. Example shows a 64^3 cube discriminator using our customized layers

into 8 sub-voxels if and only if it is neither entirely empty or entirely filled, the $2 \times 2 \times 2$ convolution with stride 2 operation can be applied for each level independently. This $2 \times 2 \times 2$ convolution operation works as an inverse function to the division, summarizing the feature in the 8 sub-voxels to the parent level.

In Figure 9, we illustrate our O-CNN architecture in the GAN setting. The top row shows the procedure of using OGN to generate a Octree-represented volumetric shape. The rest of the network shows our O-CNN design on consuming the generated Octree from the most fine-grained leaf level gradually to the top level. Starting from the leaf level, the $64 \times 64 \times 64$ Octree-represented data is first transformed by our $2 \times 2 \times 2$ convolution operation with $2 \times 2 \times 2$ stride size (**OGNDownConv** operation in the figure) into a $32 \times 32 \times 32$ feature maps. Then, the generated $32 \times 32 \times 32$ Octree data is transformed to have the same channel size to the generated $32 \times 32 \times 32$ feature maps. Then, we use concatenate the two feature maps together using our customized **OGNConcat** layer. Notice that all of these operations are performed on sparse representation, although we show dense 3D cubes in the figure for better clarification. Then, another **OGNDownConv** operation is applied to further shrink the size of the feature map into $16 \times 16 \times 16$. After being concatenated with the transformed $16 \times 16 \times 16$ generated Octree data, we use our customized **OGNS2D** (OGNSparseToDense) layer to convert the feature map into regular grid representation. Now, we can use the standard dense convolution operation and fully connected operaton to compute out the final features for the given shapes.

5.3. O-CNN Caffe Layers

To perform the OGN-Auto-Encoder and OGN-GAN experiments in Section 6, we implement several customized Caffe layers for our O-CNN architecture. Here are the de-

tails for the layers.

OGN Select Level Layer This layer is for the ground truth (or real) 3D voxel of the object which has been transformed into Octree structure. The **OGNData** layer released by OGN authors support loading grount-truth Octree data from disk. However, in the outputs of the layer, all levels are mixed together. We write this layer to separate the mixed data into data with different levels as separate outputs.

OGN Level Prediction Layer For each layer of the OGN features, we filter out the values which have been predicted as "mixed". Since the "mixed" features actually contain information for both current level and the previous level. The input to this layer is a Softmax prediction for each voxels in the current level. The output of this layer is only the voxels that is predicted to be "filled".

OGN Down Convolution Layer This layer is borrowed from the OGN Convolution layer except that now it can take stride equals to 2 while performing the convolutions. The kernel size is fixed with 2 for our discriminator architecture as described above.

OGN Concatenate Layer This layer will concatenate two input OGN features expressed in Hash Tables. This layer would also verify that the key sets is disjoint from two different levels of Octree features. I.e. Only the filled features will be present in a certain level, and thus the corresponding key is unique among the two key sets.

OGN Sparse to Dense Layer This layer is used to produce dense 3D voxel features using the sparse OGN features in the form of hash tables. And hence we can use dense 3D convolution on the features after such layer.

6. OGN-OCNN Experiments

In this section, we use the OGN Caffe code provided by the author of [20] and our own customized O-CNN Caffe layers to do more experiments on generating 3D shapes with high-resolution. First, we use our O-CNN implementation to train a 3D shape classification network. The network can take Octree-represented high-resolutional 3D data as input and output the category labels. This experiment serves as sanity check for our customized layer implementation. Next, in Section 6.2, we train the OCNN-OGN 3D Shape Auto-encoder. Finally, we discuss our attempts on training OGN-OCNN-GAN and the difficulties that we encounter during the experiments.

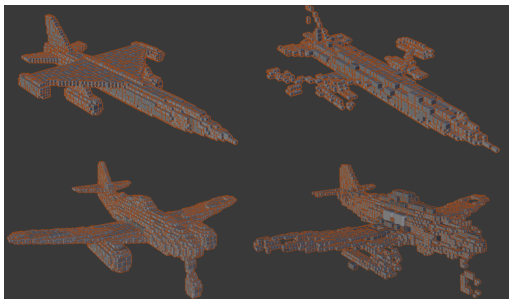

Figure 10. Auto-encoder result for planes. Shapes are randomly picked from testing set. Left columns are the reconstructed shapes. Right columns are the ground-truth data.

6.1. OCNN Shape Classification

We use our O-CNN architecture to train a object category classifier on ShapeNet [1] dataset. We pick ten most common object categories: airplane, bag, bed, bicycle, car, chair, lamp, mug, sofa, table. We randomly split the dataset into training and testing data with ratio 0.8 and 0.2. There are 26,335 models for training and 6,595 models for testing. We use the official solid volumetric data of resolution $128 \times 128 \times 128$ downloaded from the ShapeNet website.

We train a simple model without heavy fine-tuning, since the goal of doing this experiment is to check the correctness of our implementation for the O-CNN layers. We train the model with 128 channel length almost for every feature maps. Four-level Octree models are consumed as inputs and we output ten scores for predictions. After training on NVIDIA Tesla K40c GPU for 1,200 iteration with batch size 32, we get 96.8% training accuracy and 95.4% testing accuracy.

6.2. OCNN-OGN Auto-encoder

In this section, we conduct 3D Shape Auto-encoder experiment with our O-CNN as encoder and OGN as decoder. We use Caffe to train the models on two object categories: plane and chair. We have tried to train the model using *prop_pred* mode from scratch but we find that the training gets stuck in the local minima pretty fastly. It predicts no "fixed" states at all so there will be only one level of outputs. Thus, we use *prop_known* training approach to train from scratch using the ground-truth octree structure information and then fine-tune the models using *prop_pred* mode.

We use Intersection-over-Union as the metric to quantitatively evaluate the reconstruction loss. After training for 9,200 iterations, we get 0.63 mean IoU for plane category on the test set. After training for 3,600 iterations, we get mean 0.54 IoU for chair category on the test set. Here are some results that I get.

We can observe from the reconstruction results that we may need to train the network for longer time, since 9,200 iterations of training provides much better IoU than 3,600

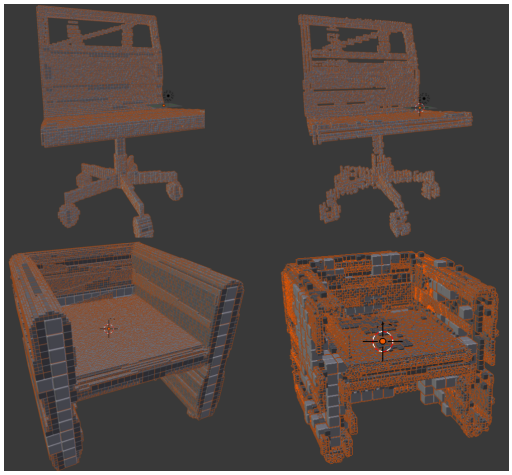

Figure 11. Auto-encoder result for chairs. Shapes are randomly picked from testing set. Left columns are the reconstructed shapes. Right columns are the ground-truth data.

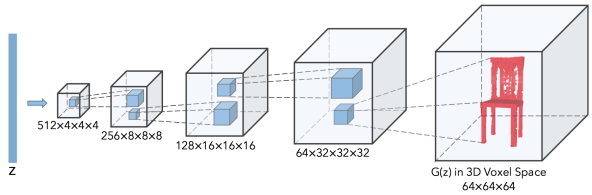

Figure 12. The generator in 3D-GAN. The discriminator mostly mirrors the generator.

iterations. Also, we can see clearly that some details are missing, which requires more training time. Due to the time and resource limit, we do not train the networks for longer time. In the original paper, the authors report that they trained their models for around 100,000 iterations. Thus, we believe that to produce better results we need to train for longer time.

6.3. OGN-OCNN 3D GAN

3D Generative Adversarial Networks In this section, we explore training GAN using OGN as generator and O-CNN as discriminator. We encounter some difficulties in making it work: 1) we have to train from scratch using the *prop_pred* training procedure for OGN generator, which is extremely hard even on shape classification task; 2) 3D GAN training is very unstable, even hard for original 3D-GAN paper [24]. Next, we discuss our experiment setting, training details and further analysis on this experiment.

Jiajun Wu *et al*. [23] learned a 3D generative model using adversarial training (3D GAN). The generator maps a 200-dimensional latent vector z randomly sampled from a probabilistic latent space to a $64 \times 64 \times 64$ cube as $G(z)$ in 3D voxel space. As shown in Fig. 12, the generator upsamples the 3D voxel reshaped from the latent vector z using dense 3D up-convolution, the discriminator mirrors the generator using dense 3D convolution. As the original paper in [10], 3D GAN also uses the binary cross entropy criterion as the classification loss, and hence the overall loss function for the generative adversarial network is as follows:

$$Loss_{3D-GAN} = logD(x) + log(1 - D(G(z))) \quad (1)$$

where x is real 3D voxel of an object rendered from the engine. Here z is randomly sampled from an i.i.d. Gaussian Distribution $p(z)$ between 0 to 1.

OGN-based 3D GAN In Figure 9, we illustrate our GAN setting. Our generator is essentially the same from the shape from id version of the OGN network, but instead of a 4095-dimensional one hot vector, we have it take as input the 200-dimensional latent vector z. The discriminator is somewhat much trickier. The output of an OGN is predicted level by level, so our discriminator has to be capable of consuming different output features at different level and then aggregate them for followed-up convolutional layers.

As shown in Fig. 9, The upper most row is the generator, which outputs 3 features at different octree levels, 16^3, 32^3, and 64^3. In this example the 64^3 features are the final output of the generator. We do the following in a backward manner for the features from the deepest level l_{out} to the first level l_0. For level l and $l-1$, we fist perform an OGN Level Pred layer on the featurs at level l to select out the mixed valued features, followed by a down convolution using OGN Downconv. For the layer $l-1$, we need to convolve it with a $1 \times 1 \times 1$ convolutional filter to obtain same number of features as level l, followed by an OGN Level Pred layer to select out the mixed values. We then concatenate the features of the down convolved layer l and the $1 \times 1 \times 1$ convolved layer $l-1$ to produce the new features for level $l-1$. We repeat this process until reaching the first level of the output octree features. In the example of Fig. 9, this is to the level of 16^3.

We can add an OGN Select Level layer on top of the above process for each inital levels form l_{out} to l_0 to process a known octree structure used as the ground truth (real) input.

Now the features are all concatenated in the level l_0, hence we can apply the OGN sparse to Dense layer to transform the sparse features as the form of hash maps to the dense 3D voxel features. Finally we simply just need to apply a dense 3D convolution with stride equals to 2, kernel size equals to 2 to down sample the feautres until reaching $1 \times 1 \times 1$ real or fake label.

Training Details We used alternative ADAM optimizer in Caffe to perform the two-phase training, setting the initial learning rate to 2×10^{-4}. We used weight decay 5×10^{-4} and batch normalization in the discriminator for regularizing the network during training. The momentum set for the

generator is 0.5 and the momentum for the discriminator is 0.999. We have two versions of the two-phase training due to the flexibility of how many iterations do we train the generator and then update the discriminator. We set the base of such two-phase training to $4:1$ and $1:1$. The former basically means that we update the discriminator after 4 times of updating the generator. With the batch size equals to 32 for both scenario, the whole model is trained on a sinlge Tesla M60 GPU.

Results and Analysis We tried very hard on tuning the 3D-OGN-GAN framework. But finally it fails to work. The generator just generates some empty or full cube shapes. It is well known that GAN is hard to train. But, we thinks the most possible reason is that we train the GAN too end-to-end. Recall that for all of the previous experiments that we have discussed above, we use multiple losses for all the levels in the Octree. However, in our GAN setting, this is impossible because the generator is generating from just a random noise. It is not possible to use per-level supervision. To solve this problem, one potential approach is to add multiple discriminators for every level in the Octree, similar to the idea of stackGAN [28]. We should hope the GAN loss at different resolution can serve as supervision at different level. However, due to the time limitation, we leave this as future work.

7. Conclusion and Future Work

In this paper, we explore the problem of consuming and generating 3D shapes with high resolutions, e.g. 128^3, 256^3 or even higher, using volumetric convolutional neural network, with the help of the sparse Octree-based volumetric representation. To be more specific, we explore the two recent papers: Octree Generating Network (OGN) and Octree-CNN (O-CNN). We use the Caffe code the OGN authors released and implement our own customized Caffe layers for O-CNN. We reproduced two experiments, shape reconstruction from shape id and shape reconstruction from single images, in the original OGN paper. As an extension, we also try to combine OGN and O-CNN to train a 3D shape classifier, 3D high-resolution auto-encoder and 3D high-resolution Generative Adversarial Network (GAN).

For the shape reconstruction, 3D shape classification and 3D high-resolution shape auto-encoder experiments, we successfully train the models using Caffe on solving these tasks. We observe reasonable results for these experiments. For the GAN experiement, we tried very hard to make it work. However, we think the problem is quite hard since we lose the level-by-level supervision that is almost a must for even simpler tasks, such as shape classification and shape reconstruction. We propose to use stackGAN idea to solve this, which is to add several discriminators for all the levels in the Octree. We leave this as future work.

We also consider to re-implement the OGN layers into more flexible deep learning frameworks, such as TensorFlow or PyTorch. Caffe really limits our GAN experiments since we cannot easily adjust the generator and discriminator adaptive training strategy, which is suggested by a lot of GAN papers. However, to implement the layers in TensorFlow or PyTorch is quite challenging since it is unclear how to pass hashtable or octree data structures as inputs and outputs to the layers. One potential solution is to serialize the hashtable as tensors, pass them in and reconstruct the hashtable or octree data structure whenever we need it. Although this solution is not so efficient, it is applicable since the cost for hashtable serialization and reconstruction is negligible compared to the convolution operation.

Acknowledgement

We thank the instructors and TA for their great help on discussing the project and thank Microsoft Azure for providing GPU resources for us to use on running the experiments.

References

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

[2] H. Chen, Q. Dou, L. Yu, and P.-A. Heng. Voxresnet: Deep voxelwise residual networks for volumetric brain segmentation. *arXiv preprint arXiv:1608.05895*, 2016.

[3] 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 *European Conference on Computer Vision*, pages 628–644. Springer, 2016.

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

[5] X. Di, R. Dahyot, and M. Prasad. Deep shape from a low number of silhouettes. In *Computer Vision–ECCV 2016 Workshops*, pages 251–265. Springer, 2016.

[6] 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*, 39(4):692–705, 2017.

[7] H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image. *arXiv preprint arXiv:1612.00603*, 2016.

[8] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[9] R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation

for objects. In *European Conference on Computer Vision*, pages 484–499. Springer, 2016.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[11] B. Graham. Sparse 3d convolutional neural networks. *arXiv preprint arXiv:1505.02890*, 2015.

[12] M. J. Hosseini and S.-I. Lee. Learning sparse gaussian graphical models with overlapping blocks. In *Advances in Neural Information Processing Systems*, pages 3808–3816, 2016.

[13] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[14] D. Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[15] D. J. Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.

[16] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[17] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[18] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. *CoRR*, abs/1611.05009, 2016.

[19] G. Riegler, A. O. Ulusoys, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. *arXiv preprint arXiv:1611.05009*, 2016.

[20] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *CoRR*, abs/1703.09438, 2017.

[21] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *arXiv preprint arXiv:1703.09438*, 2017.

[22] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4), 2017.

[23] 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.

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

[25] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision supplementary materials.

[26] L. Yi, H. Su, X. Guo, and L. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. *arXiv preprint arXiv:1612.00606*, 2016.

[27] M. E. Yumer and N. J. Mitra. Learning semantic deformation flows with 3d convolutional networks. In *European Conference on Computer Vision*, pages 294–311. Springer, 2016.

[28] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint arXiv:1612.03242*, 2016.