Multi-Resolution 3D Convolutional Neural Networks for Object Recognition

Sambit Ghadai, Xian Lee, Aditya Balu, Soumik Sarkar, and Adarsh Krishnamurthy

Department of Mechanical Engineering, Iowa State University, Ames, Iowa, 50011, USA

Abstract

Learning from 3D Data is a fascinating idea which is well explored and studied in computer vision. This allows one to learn from very sparse LiDAR data, point cloud data as well as 3D objects in terms of CAD models and surfaces etc. Most of the approaches to learn from such data are limited to uniform 3D volume occupancy grids or octree representations. A major challenge in learning from 3D data is that one needs to define a proper resolution to represent it in a voxel grid and this becomes a bottleneck for the learning algorithms. Specifically, while we focus on learning from 3D data, a fine resolution is very important to capture key features in the object and at the same time the data becomes sparser as the resolution becomes finer. There are numerous applications in computer vision where a multi-resolution representation is used instead of a uniform grid representation in order to make the applications memory efficient. Though such methods are difficult to learn from, they are much more efficient in representing 3D data. In this paper, we explore the challenges in learning from such data representation. In particular, we use a multi-level voxel representation where we define a coarse voxel grid that contains information of important voxels (boundary voxels) and multiple fine voxel grids corresponding to each significant voxel of the coarse grid. A multi-level voxel representation can capture important features in the 3D data in a memory efficient way in comparison to an octree representation. Consequently, learning from a 3D object with high resolution, which is paramount in feature recognition, is made efficient.

1 Introduction

Data encountered in real life are often three dimensional (3D) in nature. Previous works [1, 2] has shown that information extracted from the data in two dimensions (in the form of 2D images or 2.5D with depth channel) is often sufficient for most of the traditional object detection problems. However, some of the problems necessitate one for having cognition of the data in its raw format, Eg. Point cloud data obtained from a Lidar [3], 3D object data used for rendering smooth 3D graphics [4], or engineering data used in design for manufacturing [5] etc. These problems may certainly be solvable in the 2D space, yet there is a huge loss of information while converting data from 3D space to 2D space. There are several works to demonstrate the learning from multiple views of the object [6, 7, 8, 9]. Though effective in many applications as shown above, the spatial understanding of the features are not available and this makes it not possible to learn certain features of the object. For example, even for a simple task of recognizing the volume of an object is not possible by taking multiple views, but it is possible to do that using a spatial representation of the object. Furthermore, the 3D data’s spatial features could be ambiguous unless augmented with additional information such as depth, normals etc. of the object and many approaches make use of such an approach. Thus, learning from 3D data in the spatial domain is rather quintessential for 3D object recognition.

Learning from data using only fully connected neural networks is difficult due to the curse of dimensionality. Thus, in order to learn efficiently, sophisticated architectures such as convolutional neural networks
(CNN), which preserves the spatial localization and learn features in a hierarchical fashion are usually used [10, 11]. Since CNN are more efficient and has better learning capabilities due to its shift invariance, there is a strong affinity to use such methods for higher dimensional data. However, extensions of such methods to learning from 3D data also poses the challenge of dimensionality again due to the resolution of the voxel grid. 3D data is usually represented by overlaying an uniform grid over it to convert it from a continuous euclidean representation to a discrete voxelized representation. This voxelized representation is then learnt by the CNN. Though such methods have great performance on standard datasets such as ModelNet10 and ModelNet40, there is the challenge of not being able to learn from data that exceeds a certain number of samples or resolution due to limitation of computation hardware. Thus, intelligent ways to overcome the challenge of high voxel resolution while maintaining the ability to learn from the data effectively at higher resolution and larger data is of great interest.

In this paper, we try to address this issue by taking a unique approach for representing the data and then learning from it using the proposed representations. We take a multi-resolution approach for representing the 3D data where we have multiple levels of resolution of the voxel grid (thus having a non-uniform grid representation). The first level represents very coarse features in the 3D data and wherever there are boundaries or key features, we generate the next level of voxels to get a finer resolution occupancy information of the 3D data. We can further extend this idea to various levels of resolutions to represent the data accurately and efficiently. The following are the specific contributions of this paper:

1. We use a novel multi-resolution voxel representation to efficiently represent the 3D data and develop an algorithm to learn from such a non-uniform voxel representation.
2. We developed a new algorithm to train a new multi-level network with similar architectures for the coarser and finer level of resolution and achieve better performance in the case of benchmark datasets, such as ModelNet10, in compared to 3D data represented by a coarse resolution and a dense resolution that is equivalent to the combined coarse and fine resolution.
3. We also show comparison with other related methods in terms of performance in learning and computation for learning.

The paper is arranged as follows. In section 2, we discuss a few significant works in the field of learning from 3D data with various data representations. Section 3 describes the multi-resolution representation of 3D data in the form of voxels, that we implement using a GPU accelerated algorithm, to efficiently represent the data in a sparse manner. We explain about the Multi-Resolution Convolutional Neural Networks (MRCNN), that effectively learns the features from the multi-resolution voxel representation, in section 4. Finally in section 5, we present the preliminary results from evaluating the MRCNN on multi-resolution voxel data to classify objects in the ModelNet 10 dataset and also explain the effectiveness of MRCNN to learn from sparse data with moderate memory requirements.

2 Related Work

There are several related works, while we try to enlist all the related work, it is certainly possible to have missed some of the works. Learning from 3D data in deep learning started from VoxNet [11] on using 3D-CNN and they faced a lot of challenges for achieving good accuracy on the modelnet10 and modelnet40 dataset. After that, many works involved voxelizing the 3D data and then using different kinds of architectures such as variational convolutional autoencoders and deep convolutional generative adversarial networks [12, 13] etc. But one common challenge with these works was the resolution. Most of papers could go up to (32^3) resolution but recently with the increase in compute power we can go to a resolution of (64^3 or 128^3). But still going beyond that is very difficult. There are several new works on PointNet [3] and similar works to deal with point clouds. This work is the first bifurcation in the 3D data between CAD models which cannot be represented as point clouds. Though, PointNets solve the problem of resolution
in the realm of learning of point clouds, learning from CAD models was still dependent on resolution of the voxels. One of the best attempts and somewhat very closely related to our work is the work on OctNets [4, 14]. OctNets try to learn from an octree based voxel representation of the data. This is related to our work as they could go to an effective resolution of $256^3$ in a depth of 3. Unlike Octrees, we could achieve the same effective resolution in a depth of 2 (i.e. a finer level 2 and a coarser level 1) in the multi-resolution data. Now, we shall describe our work on multi-resolution representation.

3 Multi-resolution Voxelization

In this section, we describe our GPU-accelerated algorithm to create a multi-level voxelization of a B-rep model. To create a multi-level voxelization, we first construct a grid of voxels in the region occupied by the object. We then use a triangle-box intersection test to identify the boundary voxels. Once the boundary voxels are identified, we create an index array that identifies the memory location of each coarse level boundary voxel in the fine level voxel grid. We then make use of the same triangle-box intersection test to identify the fine level voxels that intersect with the triangles of the B-rep model in parallel. Using this method on the GPU, a fine voxelization of the model with an effective voxel resolution of $1000^3$ can be generated. This voxel resolution was chosen because it is sufficient to resolve the fine surface features of the model.

3.1 Coarse Level Voxelization

To identify the boundary voxels of the coarse voxelization, the voxels that intersect with the triangles of the B-rep model are first identified. Since a valid B-rep model does not contain any triangles in its interior, any voxel that intersects with a triangle is then classified as a boundary voxel that contains a part of the solid model’s boundary surface. In our multi-level voxelization process, the boundary voxels are identified for two specific reasons. First, the boundary voxels need to be identified in order to generate the finer level voxel grid inside the boundary voxels. This allows for a higher resolution of voxelization without exponentially increasing the total voxel count. In addition, once the boundary voxels are identified, the average surface normals of the triangles that intersect with the voxel can be embedded into that voxel. This allows for realistic lighting calculation using only the voxelization to generate a surface rendering of the model using ray casting.

The process of identifying the boundary voxels was sped up by parallelizing the triangle-box intersection test on the GPU for each triangle. To accelerate this operation further, we first identify the voxels that contains the triangle’s vertices so that we can cull the voxels that need not be tested for intersections with the triangle. This is attributed to the fact that once the boundaries of the triangle are known, we do not need to perform the triangle-box intersection test on voxels that lie outside of the boundaries as shown in Figure 1. The index of the voxel that a certain vertex lies in can be calculated using the position of the vertex relative to the boundaries of the AABB of the object using

\[
\begin{align*}
  i & = \left\lfloor N_x (x_p - x_{\text{min}}) / (x_{\text{max}} - x_{\text{min}}) \right\rfloor, \\
  j & = \left\lfloor N_y (y_p - y_{\text{min}}) / (y_{\text{max}} - y_{\text{min}}) \right\rfloor, \\
  k & = \left\lfloor N_z (z_p - z_{\text{min}}) / (z_{\text{max}} - z_{\text{min}}) \right\rfloor, \\
  v_{\text{array}} & = k N_y N_x + j N_x + i,
\end{align*}
\]

where $N$ contains the dimensions of the voxel grid generated to encapsulate the model, $*_{\text{min}}$ and $*_{\text{max}}$ corresponds to the minimum and maximum corner of the AABB, and the $v_{\text{array}}$ is the index in the global array of the coarse voxelization. We then perform an intersection test with all the voxels that lie within the bounds and the triangle using the separating-axis test [15] on the GPU. The rendering-based voxelization has already classified the voxel as being inside or outside the model, and hence, once the voxel has been identified as a boundary voxel we can change the value in the voxelization to indicate it as a boundary voxel. For illustration purposes, the boundary voxels are shown as a separate array in Figure 2; in practice,
Figure 1: Identifying the boundary voxels in 2D. The voxels in which the vertices of the triangle lie are first identified. The triangle is then intersected with the AABBS within the bounds along the 2 directions (marked in yellow) to identify all boundary voxels (green).

we use the same array but indicate boundary voxels using a different integer, say 2. The specific algorithm is outlined in Algorithm 1.

Algorithm 1: Classify boundary voxels of the object and create a list of all the triangles contained in it.

```plaintext
forall Triangles t(v1, v2, v3) in Object in parallel do
    Calculate i_{min}, i_{max}, j_{min}, j_{max}, k_{min}, k_{max} using v1, v2, v3;  // Determine voxel bounds of the triangle
    for i = i_{min} to i_{max} do
        for j = j_{min} to j_{max} do
            for k = k_{min} to k_{max} do
                Determine voxel v(i, j, k);
                if TriBoxIntersect(v, t) then
                    AddTriangleToVoxel();
                end
            end
        end
    end
end
```

One of the main challenges in parallelizing the above algorithm with respect to the triangles of the object is that if two different triangles intersect with the same voxel, it can lead to a race condition in storing the triangle indices in the voxel data structure. We overcome this potential race condition by using an atomic addition operation on the GPU to find a free memory location to store the triangle index. The specific code listing for a CUDA implementation is given in the Appendix. We also need to choose an appropriate buffer size for storing all the triangles intersecting a voxel. If the number of triangles intersecting a voxel is more than the buffer length, the code produces a warning and the classification is re-run with a larger buffer length based on the first computation. This reduces the performance, but in practice, having a reasonably large buffer length alleviates this problem in the models that were tested.

Once all the boundary voxels are identified, we make use of an exclusive prefix-sum array [16] to keep track of the address of the fine level voxelization. The size of the prefix sum array will be the same as the coarse level voxelization (see Figure 2). The prefix sum array will be referenced later when performing the fine level voxelization.

3.2 Fine-Level Voxelization

After classifying the voxel centers in the boundary voxel, the classification result along with any additional information about the voxel (such as coordinates, surface normals, etc.) need to be stored in a flat array data structure for retrieval on the GPU. However, since the number of boundary voxels will vary depending on
Figure 2: Data structures for the multi-level voxelization.

the model and the coarse level voxelization, the size of the fine level voxelization is not constant. To keep track of the address locations of the boundary voxels in the fine level array, we make use of the exclusive prefix sum. The prefix sum array keeps track of the boundary voxels in the coarse level voxelization. In our implementation, all the boundary voxels are divided into the same number of user-defined fine level voxels, and hence, using the prefix sum address array, we can directly access the memory location of the fine level voxelization. An example of this operation in 2D is shown in Figure 2.

After the fine level voxels have been classified as inside or outside the B-rep solid model, the boundary voxels in the fine level voxelization can also be identified. Classifying the boundary voxels for the fine level uses the same procedure as the coarse level. However, it is faster than the coarse level since instead of testing all the triangles in every face, we just need to test the triangles that have already been classified as intersecting with the coarse level voxel. We test the triangles that intersect with the coarse level voxels for intersection with the fine level voxels. The GPU kernel again performs the triangle-box intersection test; this operation is parallelized for all fine level voxel boxes in all the boundary voxels simultaneously as shown in Algorithm 2.

Once the boundary voxels in the fine level have been identified, we store the average surface normal of all the triangles that intersect with the voxel. This surface normal is then used while rendering the voxelization. We can also directly render the voxels as wireframe to check for errors in voxelization. We assign different colors to the fine and coarse level voxels, and also to inside and boundary voxels as shown in Figure 2. The voxel resolution in the image is set lower to better differentiate between the coarse and fine level voxels.
Algorithm 2: Classify fine level boundary voxels of the object.

\[
\text{forall Boundary voxels, } v_b \text{ in parallel do}
\]
\[
\text{forall Fine level voxels, } v_b(i) \text{ in parallel do}
\]
\[
\text{forall Triangles that intersect with } v_b(t_b(j)) \text{ in parallel do}
\]
\[
\text{if TriBoxIntersect}(v_b(i), t_b(j)) \text{ then}
\]
\[
\text{AddTriangleToFineLevelVoxel();}
\]
\[
\text{end}
\]
\[
\text{end}
\]
\[
\text{end}
\]

4 Multi-Resolution Convolutional Neural Networks

Multi-Resolution Convolutional Neural Network (MRCNN) was mainly inspired from the network in network architecture and sparse convolutional neural network architecture. We developed an architecture in a hierarchical way that has two architectures; one to learn features from a finer level of resolution and one to learn from the coarse level of resolution. Let $\theta_1$ be the set of weights for the coarse level convolutional network and let $f(x, \theta_1)$ be the predicted output for a given input and $\theta_1$. This provides us with a benchmark of the network’s performance with some loss $\epsilon$ in the prediction. We then augment the performance of the network by adding an additional architecture with another set of weights, $\theta_2$, which are used to learn features from the finer level of resolution. We embed it in the input volume occupancy grid instead of the boundary voxels. Thus, making the learning from the finer grid just any augmentation rather than an integration making it as computationally expensive as sparse convolution. The final output from this multi-resolution network shall be denoted with $f'(x, \theta_1, \theta_2) = f(\{x_1 | x_1 = g(x_2, \theta_2) \text{ for } j \in \text{numBoundaryVoxels}\}, \theta_1)$.

The algorithm for forward evaluation of the MRCNN is as follows:

**Algorithm 3: Forward Computation of MRCNN**

\[
\text{forall Boundary voxels, } i_b, j_b, k_b \text{ in parallel do}
\]
\[
v_b = \text{ForwardCNN}_2(x_2);
\]
\[
x_1(i_b, j_b, k_b) = v_b
\]
\[
\text{end}
\]
\[
y_{\text{pred}} = \text{ForwardCNN}_1(x_1)
\]

**Algorithm 4: Backward Computation of MRCNN**

\[
dx_1 = \text{BackwardCNN}_1(x_1, dy_{\text{pred}})
\]
\[
\text{forall Boundary voxels, } i_b, j_b, k_b \text{ in parallel do}
\]
\[
dv_b = dx_1(i_b, j_b, k_b)
\]
\[
dx_2 = \text{BackwardCNN}_2(x_2, dv_b);
\]
\[
\text{end}
\]

The forward computations of the fine level network are embedded in the input of the coarse level network. This was achieved by making use of the prefix sum array created while storing the voxel information of the finer level. This helps in embedding the output of the fine level network in the coarse level network. This could also be extended further to have a vectorized embedding of the finer level output by embedding it in the activations obtained from first layer of the coarse level voxelization. In this study, we only explore the non-vectorized embeddings.

While the forward computation of MRCNN maybe trivial, the back-propagation of the MRCNN can be tricky. The main challenge of the back-propagation computation is to link the two networks so that the gradients can pass on from the coarser level network to the finer level network. Without this link, the weights of the finer level network wouldn’t be updated accordingly. The final loss between the $y_{\text{pred}}$ and $y_{\text{true}}$ of the coarse level network was first computed using categorical cross-entropy loss. (Note: Any other loss metric may also be used) Back-propagating this loss through the coarse level network is performed using the traditional back-propagation method where the gradient of $y_{\text{pred}}$ from the loss $L$ is back-propagated all the way through the intermediate layers of the network and finally to the input data to obtain their respective gradients. Let the gradient of the loss with respect to coarse input be $dx_1$, using the prefix sum, we track the gradients of the outputs of fine level network and use it to back-propagate through the network.
to obtain the gradients of fine level network. This process is explained in the algorithm below:

It is also worthwhile to note that since the same CNN is shared among all the boundary voxels, the gradients for every parallel operation would be adding gradient to each intermediate layer. This can be understood as an atomic addition of all the gradients from each boundary voxel. With that gradient, the network could be trained to update its weights $\theta_1$ and $\theta_2$ in such a way that the loss $L$, of the final prediction $y_{pred}$, is minimized. The network parameter’s update could be performed using any optimizer, such as SGD, Adam, Adadelta etc. In this paper, we used SGD as the optimizer.

![Multi-Resolution Convolutional Neural Network](image)

**Figure 3: Multi-Resolution Convolutional Neural Network**

## 5 Experimental Results

In this study, we evaluated our proposed method of learning from a multi-resolution 3D data on Princeton’s ModelNet10 dataset that contains 3D CAD models from 10 different categories. The 3D CAD models were voxelized with 3 different resolutions and the neural networks were trained on the 3D data to learn to categorize the 3D models. Although there are areas in this study that needs further exploration, some preliminary results are shown here. A comparison was made between data with a naive coarse resolution of $8^3$, a naive dense resolution of $32^3$ and multi-resolution data with a coarse resolution of $8^3$ and a finer resolution of $4^3$, thus giving an effective resolution of $32^3$. A traditional CNN was used to train on both the coarse resolution ($8^3$) and the dense resolution ($32^3$) data while our MRCNN was trained using the multi-resolution data. All the training of the neural networks were performed on a machine with an Intel Xeon CPU having 320 GB of RAM and a NVIDIA Quadro P40 GPU having 24 GB RAM.

The validation loss and validation accuracy of the three networks are shown in Figure 4. It can be seen that the loss of the CNN trained on coarse data ($8^3$) is much higher than the loss of the CNN trained on the dense data ($32^3$) while MRCNN trained on multi-resolution data has a loss in between both the networks. Figure 5 and Figure 6 compares the validation accuracies of the three CNN. Once again, the CNN trained on the data with a coarse resolution has a poor accuracy as compared to the network trained on dense data. However, the MRCNN that was trained on multi-resolution data has a classification accuracy that is comparable to the CNN that was trained on dense data. This shows that a similar object classification performance can be achieved using a multi-resolution data representation as compared to a dense resolution data representation in 3D object recognition.

A more intriguing insight of the MRCNN is observed when the memory requirements for training the networks are analyzed. Figure 7 shows the memory requirements of the GPU while training the dense resolution network and MRCNN. An unoptimized version of the CNN that was trained on the dense resolution data requires 16 GB of GPU memory and an optimized version of the CNN that was trained the same dense data requires at least 5 GB of memory. In contrast, our MRCNN that was trained on the multi-resolution data only requires 1 GB memory while still achieving a similar classification performance as the CNN that
was trained on dense data. Thus, the benefits of using the MRCNN network are huge since this allows networks to be trained on data with a higher effective resolution without compromising the performances of the learning process.

6 Conclusion

In this paper, we explore a novel method to represent 3D data in a hierarchical manner using multi-resolution voxels and also detailed the process of learning from such hierarchical data. This method can be extended in several other domains. We also showed some preliminary results of the proposed method where our network performs better than networks trained on coarse-resolution data and performs almost as well as networks trained on dense resolution data while keeping the memory requirements 5 times lower. Future work includes exploring our proposed method on different datasets where having high resolution data is more imperative and getting a bound for the network performance in terms of the coarse and fine resolutions.
References

[1] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong 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.

[2] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In Proc. ICCV, 2015.

[3] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. arXiv preprint arXiv:1706.02413, 2017.

[4] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. arXiv preprint arXiv:1703.09438, 2017.

[5] Sambit Ghadai, Aditya Balu, Soumik Sarkar, and Adarsh Krishnamurthy. Learning localized features in 3d cad models for manufacturability analysis of drilled holes. Computer Aided Geometric Design, 62:263 – 275, 2018.

[6] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. arXiv preprint arXiv:1803.04249, 2018.

[7] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. arXiv preprint arXiv:1603.06208, 2016.

[8] Konstantinos Sfikas, Ioannis Pratikakis, and Theoharis Theoharis. Ensemble of panorama-based convolutional neural networks for 3d model classification and retrieval. Computers & Graphics, 2017.

[9] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5648–5656, 2016.

[10] Anastasia Ioannidou, Elisavet Chatzilari, Spiros Nikolopoulos, and Ioannis Kompatsiaris. Deep learning advances in computer vision with 3d data: A survey. ACM Computing Surveys (CSUR), 50(2):20, 2017.
[11] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 922–928, Sept 2015.

[12] André Brock, Theodore Lim, James M. Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. CoRR, abs/1608.04236, 2016.

[13] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. CoRR, abs/1610.07584, 2016.

[14] Gernot Riegler, Ali Osman Ulusoys, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. arXiv preprint arXiv:1611.05009, 2016.

[15] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. OBBTree: A hierarchical structure for rapid interference detection. In Proceedings of the 23rd annual conference on Computer Graphics and Interactive Techniques, pages 171–180. ACM, 1996.

[16] Guy E Blelloch. Prefix sums and their applications. Technical report, Carnegie Mellon University, 1990.