Enhanced Convolutional Occupancy Network: Leveraging Unprojected 3D Coordinates

Youjin Song

Abstract—The Enhanced Convolutional Occupancy Network aims to create a more effective representation of 3D scenes by incorporating all three coordinate values of 3D coordinates in its plane encoding process. In contrast to the Convolutional Occupancy Network(ConvONet), which discards non-projected coordinates during orthographic projection onto canonical planes for 2D plane encoder, the proposed method integrates these additional dimensions to improve feature extraction. Experimental results showcase the method's effectiveness in achieving faster convergence and generating higher-quality 3D reconstructions, as evidenced by higher Intersection over Union (IoU) scores, while maintaining a comparable computational burden in terms of time and memory usage when compared to existing methods. The code is available at https://github.com/ujinsong23/enhanced_ConvONet/.

Index Terms—Computational Imaging, 3D Reconstruction, Occupancy Network, Implicit Neural Representations

1 Introduction

Given the increasing demand for AR and MR applications in spatial computing, accurate, fast, and memory efficient 3D scene representation from obtainable sources like 2D image inputs and depth information is gaining more and more importance.

Recently, several works have introduced deep implicit representations that represent 3D structures using learned occupancy or signed distance functions. One such effort includes the convolutional occupancy network, which successfully structured a pipeline to map point clouds into 2D features by projection in order to introduce convolutional layers. It yielded finer geometric details and allowed for a flexible translation equivariant architecture compared to its previous models, which only utilized fully connected layers.

This paper focuses on effectively incorporating the 2D encoder so that it does not lose 3D information during the projection to the plane. With minor modifications to the encoder, we demonstrate achieving **better qualitative and quantitative outcomes**, along with **faster convergence**, with **minimal changes in computational time and memory usage**. We also illustrate which information from the unprojected coordinate is necessary for reconstruction by analyzing the functionality of the added module.

2 Related Work

Classical methods for representing 3D configurations include voxels, point clouds, and meshes. Voxel representations extend the concept of pixels into 3D space but face limitations due to memory growth, typically restricting resolution. Point clouds offer memory efficiency by only occupying space where objects exist but lack structural connectivity. Mesh representations, formed by deforming a template mesh, are constrained by predefined topologies.

The concept of a continuous **3D Occupancy Function** [1] emerged to address these limitations. This approach involves learning an occupancy function through neural networks, mapping 3D point coordinates to a scalar occupancy probability between 0 and 1. By overcoming previous discretization issues, it enables the representation of realistic, high-resolution meshes within a fixed resolution. The figure 1 illustrates the differences of each representations.

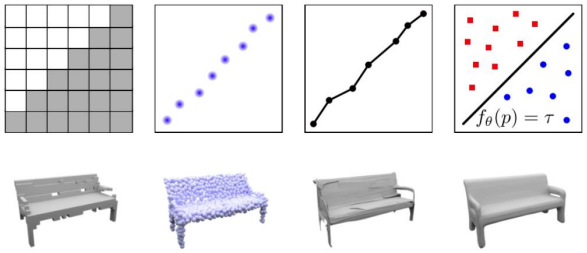

Fig. 1. The figure illustrates various classical 3D representations(left 3) that discretize the output space differently. From left to right, it shows voxel representation, point cloud, and vertices for mesh representations. The representation on the rightmost side, however, depicts a continuous decision boundary of a classifier as 3D surfaces, which is also known as an Occupancy network.

Following the emergence of the continuous 3D occupancy function, numerous methods have employed neural networks to learn continuous shape representations. Deep SDF [2], for instance, focuses on determining the distance from the surface for each point in 3D space. It achieves this by training a shape-conditioned classifier, where the decision boundary corresponds to the shape's surface, supervised by actual 3D Signed Distance Function (SDF) values. Scene Representation Networks(SRN) [3] portray scenes as continuous functions, mapping world coordinates to latent vectors containing local scene properties, solely using 2D images and camera poses(without access to depth or shape information). There also have been efforts to learn implicit shape and texture representations directly from RGB images through approaches like Differentiable Volumetric Render-

- *Department of Electrical Engineering, Stanford University*
- *E-mail: ujinsong@stanford.edu*

ing (DVR) [4]. NeRF(Neural Radiance Fields) [5] introduced an innovative approach to represent continuous scenes with intricate geometry and materials using 5D neural radiance fields, parameterized as basic MLP networks. This method leverages volume rendering techniques to achieve photorealistic reconstruction and store semantic features, reducing reliance on depth sensors with their inherent limitations.

While numerous outstanding works have explored learning fine-grained continuous 3D scenes with various input settings, this project takes a step back to focus on a simpler and more basic model at the earlier stage of this field. Specifically, it aims to develop the **Convolutional Occupancy Network (ConvONet)** [6], which is a subsequent model that enhances the **Occupancy Network** [1] further.

To be more specific, ConvONet has achieved a more flexible implicit representation for detailed reconstruction compared to its predecessor by successfully introducing convolution layers instead of fully connected layers.

Their pipeline is as follows: Given a point cloud representation, which is provided as a list of 3D vectors, they pass through several fully connected layers to expand features at each point. At this point, there are two variants of the model. One involves projecting points onto 2D xy, yz, or zx planes and then using a plane encoder, while the other directly uses a Volume Encoder without projection. Although the volume encoder preserves 3D spatial data better, processing high-resolution data with it is challenging due to the massive amount of computation required for 3D data. Therefore, after adjusting the resolution to match the computation and memory requirements of the 2D encoder, 3D encoder's performance does not outperform that of the 2D encoder due to its lower resolution. Therefore, we focus on how ConvONet utilizes the 2D encoder at this stage. They projects the extracted features to xy, yz, and zx plane, and the each projected 2D plane undergoes processing through a UNet. This forms the extracted 2D feature of the model, which then goes to the decoder, finally mapping occupancy probabilities via a fully connected network.

3 Proposed Method

The objective of the task is to accurately predict whether any given point from a continuous domain is within or outside of the object, given a set of point clouds (sets of 3D vectors). Therefore, during the training phase, three types of inputs are provided to the model: sparse point cloud data $\mathbf{x} \subset \mathbb{R}^3$, arbitrary query 3D points $\mathbf{p} \in \mathbb{R}^3$, and occupancy values for the 3D points $\mathbf{o_p} \in \{0, 1\}$. During the inference phase, the goal is to infer an accurate 3D representation from the point cloud, so point cloud data $\mathbf{x}$ and a dense enough set of points $\mathbf{p}$ that matches the resolution are provided.

The overall pipeline for training follows that of ConvONet, which includes an encoder for obtaining embeddings from the point cloud, a decoder that takes into account the extracted feature map and the given arbitrary 3D points, and a final classifier that predicts the probability value of occupancy ranging from 0 to 1. The model architecture of our model can be seen in the figure 2. Following sections describe in more detail about the training and inference phases, specifically how the 2D feature is extracted from sparse point clouds(section 3.1), how unprojected coordinates are effectively being aggregated to the extracted features(section 3.2), how the decoder functions(section 3.3), and which loss function is used(section 3.4).

3.1 Encoder

The input data is a point cloud representation from a single object $\mathbf{x}$, represented as a list of 3D vectors where each element indicates an x, y, z coordinate. After preprocessing the input (adding small gaussian noise for training robustness and normalizing so that every element falls within [0,1]), the noisy, sparse points passes through several fully connected layers to expand features at each point. Specifically, it goes through 5 residual blocks of fully connected layers, and the feature dimension is increased to 32 (which was 3 for the input). Until this step, the feature only corresponds to each point in the point cloud independently, without considering any potentially neighboring points, so the spatial information within the point cloud is neglected.

This is where the orthographic projection to a 2D plane comes into play. Each extracted feature of the point is projected to the 2D plane of its original location. Since training with 2D convolutional layers requires a fixed resolution for 2D input, some approximation of points is introduced, meaning that a fixed number of bins, in our case 64x64 assuming a 2D resolution of 64, is used. Following that, it is possible to have multiple points that are projected to the same bin, so **local max-pooling** is used. Among all the points that are projected to the same bin, the feature with the highest value is selected and projected. After going through this process, we obtain a $64 \times 64 \times 32$ plane, with each feature extracted from the fully connected layer and local max-pooling. The extracted plane undergoes processing through a UNet, and this forms the final feature embedding of our model. For each xy, yz, and zx plane, the UNet weights are shared.

3.2 Aggregation of Remaining Coordinates

Projecting spread-out and discrete points from point cloud data onto a 2D dimension is indeed a beneficial approach to utilize spatial information, but we've observed that we are discarding one coordinate value among the three during this process. We propose the idea of incorporating values from the remaining coordinate (for example, the z-coordinate for xy-plane projection), into the encoding process.

There are several methods to aggregate these values, including using other neural networks, but to streamline the process as much as possible, we attempted to utilize simple statistics from the points that do not necessarily require a significant increase in additional computational overhead. **Sum** adds up the number of points divided by total number of points, **Range** represents the difference between the maximum and minimum z-coordinate values, and **Mean** calculates the average z-coordinate value of the points that are projected to the plane.

We use the same plane resolution that we used in the encoding process, so using one of these pieces of information results in a 1-dimensional 64×64 image showing the intensity of each measure at each bin. We will use this image as weights that are element-wise multiplied to the

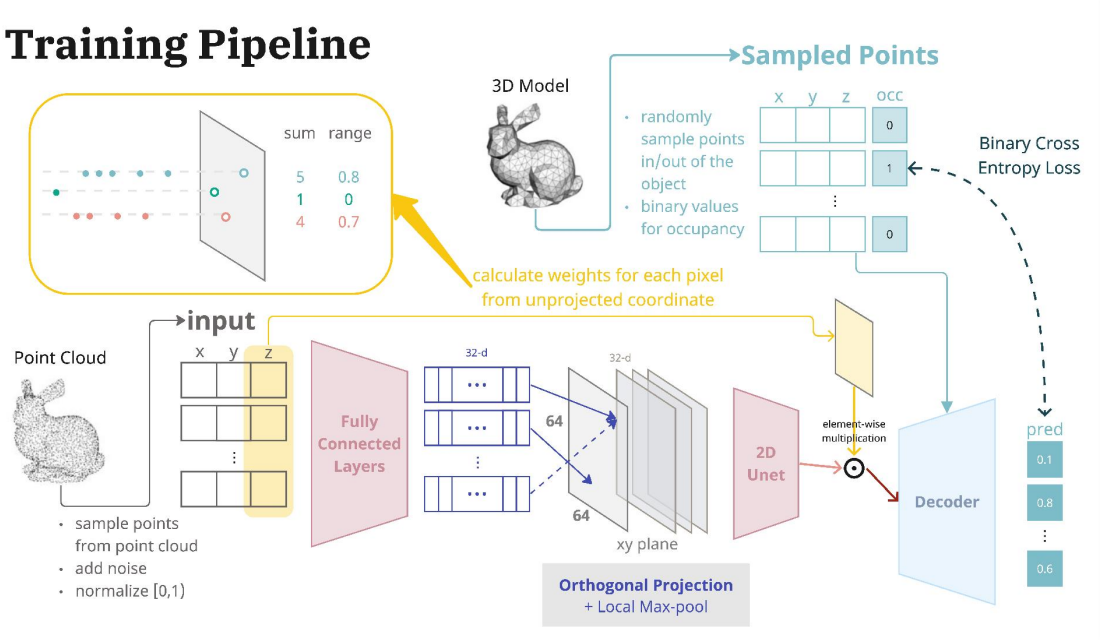

Fig. 2. The figure illustrates the overall pipeline of the model, including the encoder and decoder. It's worth noting that this figure depicts a specific scenario where points are projected onto the xy plane, thereby computing spatial weights that take into account the previously discarded z coordinates. However, in the actual implementation, all three situations of projecting onto the xy, yz, and zx planes are considered.

feature embedding from the encoder, and then passed to the decoder.

To assess the contribution of each aggregation method to the performance, we conducted a simple experimentation within the network. We stacked four layers consisting of sum, min, max, and mean, and passed them through a very shallow UNet. After training, we observed the weights of the UNet corresponding to each layer.

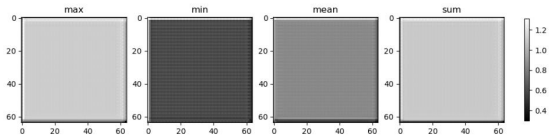

Fig. 3. Contribution of each aggregation method

The results of this experiment are shown in the figure 3. Sum has the highest intensity, indicating that it provides the most valuable information to the projection. Intuitively, having more points projected onto the plane suggests a higher probability that a point is likely to be within the object (occupancy is 1). Max has a larger intensity, while Min has a smaller intensity, as a greater maximum coordinate value and a smaller minimum coordinate value suggest a wider range of points being inside the object. The mean value of the coordinates does not necessarily contribute to increasing our understanding of the unused coordinates.

3.3 Decoder

By this point, we have obtained weighted feature embeddings by performing element-wise multiplication of the encoder output and a simple aggregation of remaining coordinates like *sum* and *range*. Using the weighted feature plane from the encoder, the convolutional decoder processes the final feature for classification for given query 3D points $\mathbf{p} \in \mathbb{R}^3$. Pointwise feature vector for each query point $\mathbf{p}$, denoted as $\psi(\mathbf{x}, \mathbf{p})$, is computed by bilinear interpolation of 3 plane feature plane.

3.4 Occupancy Prediction

Given the extracted feature vector $\psi(\mathbf{x}, \mathbf{p})$, our goal is to estimate how likely the point is inside the object, i.e., occupancy. By passing through fully-connected Residual Blocks, we predict the probability for each query point as $\hat{o}_p = f_\theta(\mathbf{p}, \psi(\mathbf{x}, \mathbf{p})) \in [0, 1]$. Here, for aggregating $\mathbf{p}$ and $\psi(\mathbf{x}, \mathbf{p})$, we added $\psi(\mathbf{x}, \mathbf{p})$ to processed $\mathbf{p}$ at each step of residual blocks, using the network architecture of [4].

The model is essentially a classifier of binary classes for given 3d point, so we used binary cross entropy loss for objective function as below:

$$\mathcal{L}(\hat{o}_p, o_p) = -[o_p \log \hat{o}_p + (1 - o_p) \log(1 - \hat{o}_p)]$$

3.5 Other Experimental Details

- **Dataset**: We used 7 classes of indoor objects(sofa, lamp, telephone, chair, cabinet, table, video display) of the ShapeNet Dataset and train/val/test split from [7].
- **Optimizer**: Adam Optimizer with learning rate of 10^{-4}
- **GPU**: Titan RTX is used. It took $\sim$ 2 hours for 100 epoch(around 50000 iterations) during training.
- **Inference**: To make 3d mesh representation with our trained model, we used Multiresolution IsoSurface Extraction(MISE) from [1].

4 Experimental Results

4.1 Quantitative Results

We quantitatively evaluated our method using different aggregation methods from section 3.2 during training time.

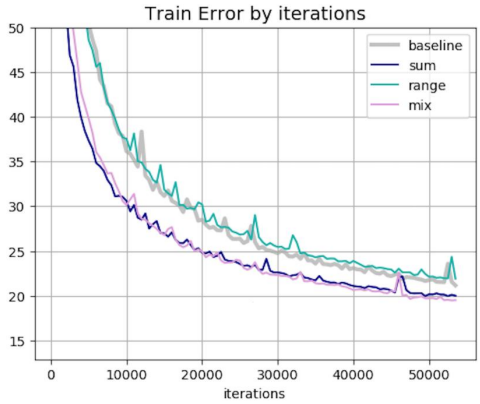

Fig. 4. The training error decreases faster when the aggregation methods of sum and mix are introduced. However, the impact of using the range is not different from the baseline model.

TABLE 1
Training time (100 epoch)

Model	baseline	sum	range	mix
Time(sec)	6149	6075	6105	6378

Since our methods obtain weights using minimal computation, the training time is almost the same as with the baseline model(refer to table 1). In terms of convergence, the aggregation methods of 'sum' and 'mix' (a mixture of sum and range) converged much faster than the baseline model, while the usage of the 'range' method showed almost the same tendency as the baseline model, indicating that 'sum' is playing a significant role.

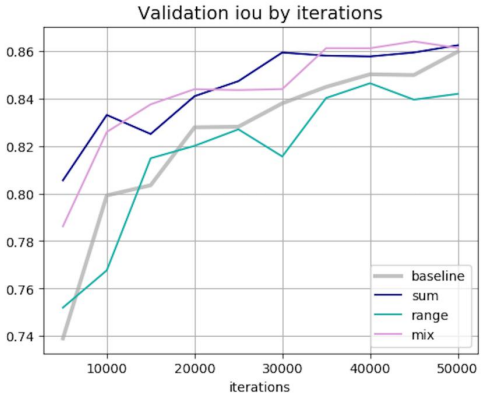

Fig. 5. The 3D Intersection over Union (3D IOU) of the 3D representation quickly reaches its maximum when the aggregation methods of sum and mix are introduced. However, the impact of using the range is not different from the baseline model.

4.2 Qualitative Results

The qualitative results before and after finishing the training also indicate the superiority of our method. As faster convergence in train error by iteration also implies in the previous section 4.1, our method showed a more fine-grained output with fewer artifacts when sampled validation data during the earlier phase of training (10000 iterations).

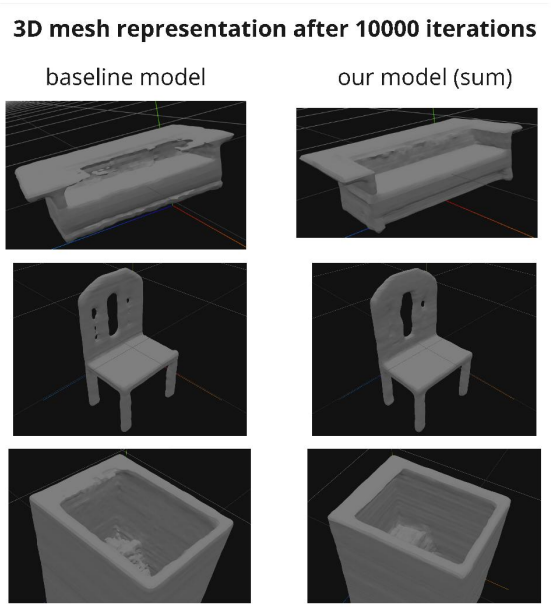

Fig. 6. The 3D representation in the earlier phase of the training exhibits differences in smoothness.

Both models produced satisfactory results after final training, but occasionally, the baseline model encountered difficulties with 3D shapes that demanded intricate detailing. Our model, however, performed well on this particular set of images.

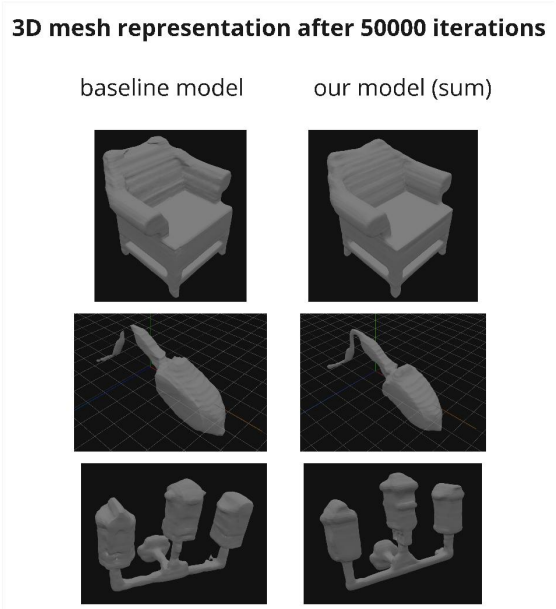

Fig. 7. Both models generally yielded good results after training(1st row), but the baseline model struggles with 3D shapes that require intricate detailing(2nd and 3rd row). Our model produces a more fine-grained output.

5 Conclusion

This paper has addressed methods for predicting the occupancy of an object in a continuous domain based on sparse point information. Unlike Convolutional Occupancy Network (ConvONet), which relies solely on orthographic projection and average pooling for feature aggregation, our method effectively employs a 3D encoder to extract comprehensive features, leading to superior reconstruction performance. Importantly, these improvements were achieved with minimal additional computational cost.

Of course, there is still room for improvement. Since the use of three orthogonal projections to xy, yz, and zx planes is somewhat arbitrary, exploring a smaller number of 2D planes that allow for more efficient feature extraction for a given scene could enhance performance. Additionally, there may be better aggregation methods that map the probability distribution at projected bins, incorporating concepts beyond simple sum and range calculations.

While 3D imagery presents greater computational demands and observation requirements, there is potential to streamline the process by leveraging human intuition or incorporating physical models. Further research in this direction could lead to even more efficient and accurate 3D scene representations, opening up new possibilities for applications in spatial computing and beyond.

References

[1] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[2] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 165–174.

[3] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene representation networks: Continuous 3d-structure-aware neural scene representations," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[4] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, "Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3504–3515.

[5] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[6] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, "Convolutional occupancy networks," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 2020, pp. 523–540.

[7] 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 *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*. Springer, 2016, pp. 628–644.