

Optimization of 3D Image Reconstruction with a 3D-Object Point Cloud Network

Britanya Wright-Smith, Shengjie Guo, Daniel Everett
Electrical and Computer Engineering Dept.
North Carolina State University
{bwright2, sguo25, deveret2}@ncsu.edu

Abstract—The reconstruction of 3D objects from 2D images is a growing topic in machine learning, and the problem proves to be challenging as we are faced with computations that grow in cubic form as we attempt to process larger inputs. 3D-ReConstnet is a point-cloud-based 3D reconstruction algorithm that aims to eliminate the concerns with voxel-based occupancy grid computational complexity. In this project, we want to explore the architecture of 3D-ReConstnet and recreate the network by hand. After recreating the network, we want to explore improvements and optimizations, so that we can improve the resulting point-cloud generation.

Index Terms—ResNet-50, 3D-ReConstnet, 3D Reconstruction, Point Cloud, loss function

I. INTRODUCTION

3D-ReConstnet is a residual neural network that extracts feature vectors and generates a Gaussian probability distribution to deal with the uncertainty of the self-occluded part of a single-view image and a multi-layer perception to generate the point-cloud model. Compared to voxel-based 3D reconstruction, one big advantage of using point-cloud networks to represent the 3D object is to avoid the problems of sparse information and high computational complexity [1]. A critical challenge is to design an optimized network and a good loss function for comparing the predicted point cloud and a ground-truth 3D model. To plug in a neural network, a suitable distance between the predicted point cloud and ground truth model must satisfy at least three conditions [2]:

- 1) Differentiable with respect to point locations.
- 2) Efficient to compute, as data will be forwarded and back-propagated many times.
- 3) Robust against a small number of outlier points in the set (e.g. Hausdorff distance would fail).

In order to optimize the network we used ResNet-50. Optimization was done by tuning the training parameters of the networks. For optimization on the point cloud generation, a critical part of constructing a 3D model, we studied the properties of different activation functions and experimented with different combinations. In order to optimize the loss function, we need to minimize the error between the output of the network and a ground-truth model to increase the accuracy.

The outline of the steps performed during this project implementation is as followed:

- 1) Obtain 2D images from the ShapeNet Dataset [3] and extract its necessary features to a feature vector using the

ResNet-50 Network. Different combinations of input to the network will be experimented with to obtain different outputs.

- 2) Obtain the mean and standard deviation of the feature vector and obtain the Gaussian probabilistic vector to encode the results.
- 3) Perform decoding of the Probabilistic Vector by implementing a multi-layer perception.
- 4) Experiment with different loss functions and training parameters to improve 3D results of output image (Seen in Figure1).

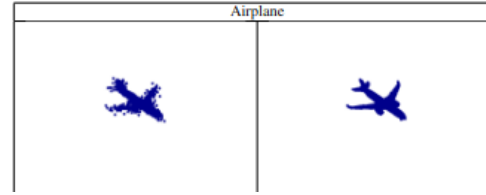


Fig. 1. Single view reconstruction of an airplane

The base model or network will be the results from our reference paper [1]. The steps performed above are the implementation of the same outline of the paper but with different optimization such as the structure of the ResNet50 Network and the loss function. All steps above outline the architecture of the 3D-ReConstnet Network.

II. 3D-RECONSTNET ARCHITECTURE

The 3D-ReConstnet can be broken down into four distinct sections: 2D input feature extraction, probabilistic vector sampling with loss function, point cloud generation, and loss evaluation. This is an end-to-end neural network that allows feature transfer to only occur within the network. With this, there is only one set of inputs and one set of outputs as shown in Figure2. This reduces the possibility of loss of features within the network and it avoids feature propagation.

A. Input Images

The 3D-ReconstNet Neural Network was trained on the ShapeNet Dataset [4] and Pix3D dataset [5]. The ShapeNet dataset consists of a total of 51300 unique 3D CAD models in 13 different categories. The Pix3D dataset consists of 7595

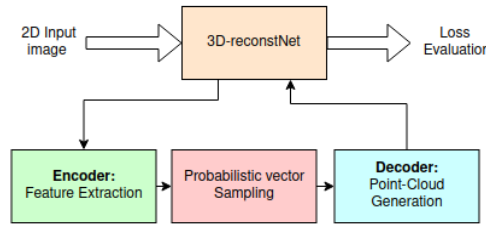


Fig. 2. End-to-end visualization of the network.

real images with real backgrounds along with their masks, ground truth CAD models and poses. The ShapeNet dataset was used in training the network while the Pix3D dataset was used to test the accuracy of the network. We consistently used a ratio of training set to testing set of 4 to 1 similar to that of [1].

B. Feature Extraction

Feature Extraction is the first layer within the 3D-ReconstNet Neural Network. The work done within this layer can be seen in Figure 3. The features were extracted using the ResNet50 which is a 50 layer neural network and then flatten using a Fully Connected layer.

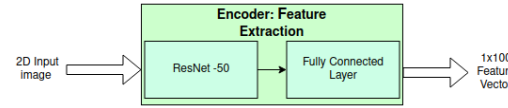


Fig. 3. Detailed flow of the Feature Extraction later within the 3D-ReConst Network

The ResNet-50 architecture contains the following layers as seen in Figure 10:

- 1) The first convolution layer has 63 kernels with the size 7×7 . They all have a stride of size 2.
- 2) A Max pooling layer is followed with a stride of 12 to reduce the dimension of the feature vector.
- 3) Within the next few convolution layer there is a 1×1 64 kernel, followed by a 3×3 64 kernel, and a 1×1 256 kernel. This is repeated 3 times resulting in 9 layers.
- 4) The next few convolution layer there is a 1×1 128 kernel, followed by a 3×3 128 kernel, and a 1×1 512 kernel. This is repeated 4 times resulting in 12 layers.
- 5) The convolution layers are repeated once again but this time 6 times resulting in 18 layers. These layers consist of a 1×1 256 kernel, followed by a 3×3 256 kernel, and a 1×1 1024 kernel.
- 6) The last set of convolution layers are tripled resulting in 9 more layers. These layers consist of a 1×1 512 kernel, followed by a 3×3 512 kernel, and a 1×1 2048 kernel.
- 7) The final layer consists of average pooling followed by a fully connected layer that gives the result of a 1×1000 dimension vector output.

This architecture extracts the features of the input 2D image without vanishing gradients which causes the network to not train appropriately. Followed by the ResNet there is have a fully connected layer that reduces the dimension down to a 1×100 feature vector. This is as stated within the paper [1]. Another way to view this step is the ResNet-50 acts as the encoder network which encodes the 2D image into feature vectors compressing the large input into a smaller feature space which later will be reconstructed into the point cloud output.

C. Probabilistic Vector Sampling

After extracting 2D image features, it is important to map those features into a normal distribution by sampling a probabilistic vector and encoding the 2D features into a condensed vector. The decoding is then done within the Point-Cloud generation portion. The work done within this layer can be seen in Figure 4. This probabilistic vector (or latent sampled vector) maps the 2D input to a multivariate Gaussian distribution.

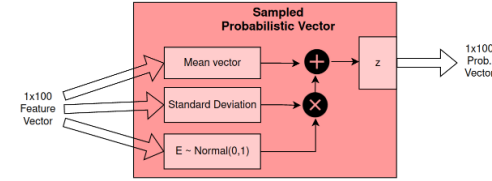


Fig. 4. Detailed flow of the Sampled Probabilistic Vector later within the 3D-ReConst Network

The steps for obtaining the probabilistic vector is as followed:

- 1) The points within the encoded feature vector of size 1×100 gets mapped by a mean (μ) and a variance (σ) vector which defines the Gaussian distribution.
- 2) Points are then sampled from this Distribution and returned as a latent variable.

A detailed outline of the steps used to obtain the probabilistic vector can be found in Figure 5.

D. Point Cloud Generation

After obtaining a sampled probabilistic vector of size 1×100 , a point-cloud will need to be generated. This involves the decoding portion of the 3D-ReConstnet. The work done within this layer can be seen in Figure 6.

The steps for obtaining the point cloud generation is as followed and can be seen in Figure 9:

- 1) The first layer is a Dense Layer with a Leaky ReLU activation outputting a size of 512.
- 2) The second layer is a Dense Layer with a Leaky ReLU activation outputting a size of 1024.
- 3) The final layer is a Dense Layer with a tanh activation outputting a size of 1024×3 .

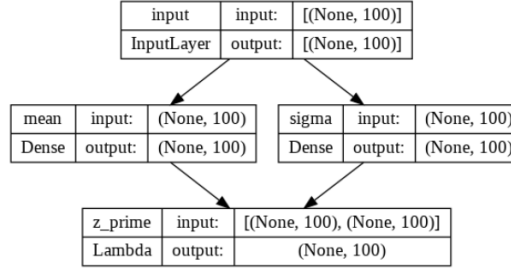


Fig. 5. Schematic Outline of achieving the probabilistic vector

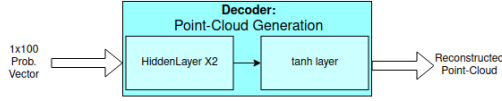


Fig. 6. Schematic Outline of achieving the point cloud generation

In order to obtain this, we implement a multilayer perceptron (MLP) (shown in Figure 9) that consists of 2 hidden layers (dense layers) with a Leaky ReLU activation function followed by a tanh output layer. The Leaky ReLU is purposefully used as the activation function so as to eliminate the loss of negative information being passed through the neural network. Since the traditional ReLU would map all negative values from input to zero, Leaky ReLU function is proposed to fix the "dying ReLU" problem by replacing the response of zero to a slightly positive slope term in the negative part, as shown in Figure 7(a). In this case, the derivative of Leaky ReLU function could obtain valid values of input among negative axis, which could be observed in Figure 7(b). Hence, using Leaky ReLU as the activation function of the 2 hidden layers could help to retain feature information, either positive or negative, as much as possible.

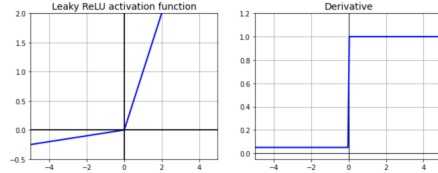


Fig. 7. Leaky ReLU(a) and derivative of Leaky ReLU(b)

The output of multi-layer perceptron should be normalized to the range of $[-1, 1]$. Then for the activation of final layer of MLP, tanh function is a appropriate option. As shown in Figure 8(a), its output is zero centered and map all input to the range of $[-1, 1]$. With this characteristic, tanh would fit our generated point cloud data, which is also between -1 and 1, in a very efficient way. The tanh function is not used in the 2

hidden layer function although it could minimize the difference between the predicted generation and the groundtruth. One important reason is that the gradient of neural network may vanish because of inappropriate activation in the process of training. Also, as shown in Figure 8(b) the derivative of tanh activation function is beyond 1 all the time. Then the result of tanh activation function easily approaches 0, and eventually the vanishing gradient.

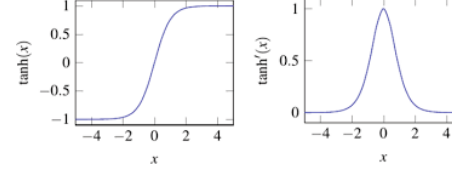


Fig. 8. tanh(a) and derivative of tanh(b)

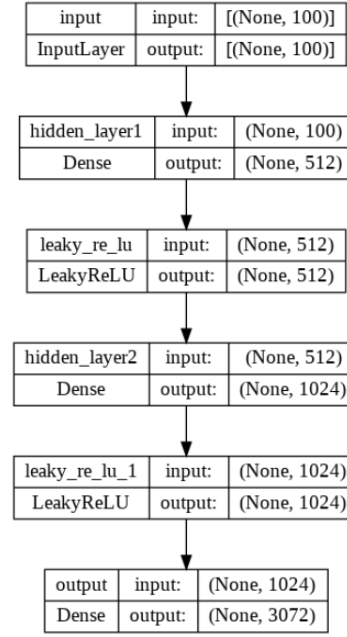


Fig. 9. Schematic Outline of achieving the point cloud generation

E. Loss Evaluation

Point clouds provide an interesting challenge when measuring correctness through a loss function, because the points in the cloud are not necessarily ordered. This means that a loss function must be chosen carefully that is order-agnostic, allowing any point in the prediction to be evaluated with any point in the ground truth, and vice-versa. For this reason, the

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig. 10. Detailed flow of the Feature Extraction later within the 3D-ReConst Network

Chamfer Distance algorithm is a good choice for point cloud evaluation.

$$d = \sum_{X_g \in X_{gt}} \min_{X_p \in X_{pred}} \|X_g - X_p\|^2 + \sum_{X_p \in X_{pred}} \min_{X_g \in X_{gt}} \|X_p - X_g\|^2$$

With Chamfer Distance, each point in the predicted point cloud calculates its square distance to *every* point in the ground truth point cloud, then a minimum distanced pair is selected for each point in both the predicted point cloud and the ground truth point cloud before being summed to create the final Chamfer Distance. As this function essentially sums the ideal error between the point clouds, it serves as a good loss function to minimize total error between the predicted point cloud and the ground truth point cloud. Minimizing this error helps to hone the predicted point cloud towards the general shape of the ground truth point cloud, but is not good at refining the quality of the predicted point cloud to sharp detail. Because every point in one cloud must be compared with every point in the other cloud, Chamfer Distance is a $O(n^2)$ computation, making it increasingly challenging to perform on traditional CPUs for larger point clouds, and thus is usually implemented on data-parallel processors like GPUs.

The second loss function utilized by ReConstNet is the Earth Mover's Distance loss function. This function intends to create a bijection between the predicted point cloud to the ground truth point cloud, which ultimately helps to move individual points of the prediction towards their ground truth counterparts, creating a sharp detailed image.

$$d = \min_{\phi: X_{gt} \rightarrow X_{pred}} \sum_{x \in X_{gt}} \|x - \phi(x)\|$$

EMD attempts to find the minimum total distance as a bijection of the two point clouds, which can become an unbounded task computationally. For this reason, EMD is usually approximated.

III. TRAINING AND IMPLEMENTATION

Training ReConstNet proved to be a challenging task, with many implementation issues arising during our recreation efforts. To properly implement ReConstNet, a few things must be considered:

- 1) Any attempt to train a neural network using complex loss functions requires heavy utilization of a GPU.
- 2) Because we are bound by compute resources, a scaled attempt at ReConstNet was the only practical option.

Because of this, we quickly learned that we would be unable to train a full ReConstNet model as-good-as the one in the original paper, due to limited compute resources we had at our disposal. This led to a modified implementation of ReConstNet, that omitted 2 things:

- 1) The Earth Mover's Distance portion of the loss function.
- 2) The Diversity Loss portion of the loss function.

With these constraints, we were able to implement ReConstNet with a practical training time given our compute constraints. Our ReConstNet model was trained using the ShapeNet [4] dataset, which is a collection of 9 different image classifications containing over 1 million total images to use for training and validation. The model was trained using an Adam optimizer, which is an improved Stochastic Gradient Descent method that involves a dynamic update rate per training step. The Adam optimizer is a popular choice in deep

Trade-offs	3DReconstNet from [1]	Our 3DReconstNet
Architecture	ResNet50 architecture defined from scratch resulting in different size output on each layer but final output is the same	ResNet50 used had a predefined architecture
Loss Functions used	Chamfer Distance and Earth Mover's Distance	Chamfer Distance
Number of Epochs	50	3
Time took to compute one run	Not Specified	At 3 epochs it took 16 hours to compute
Mean Chamfer Distance over ShapeNet Dataset	0.00409	0.03355

Fig. 11. Table of tradeoffs

neural networks (DNN), because of its improved optimization performance [6].

To properly compute a loss function, known ground truth point clouds must be computed over the ShapeNet dataset to use as input to compare against. The original authors of [1] provided these ground truth point clouds, so we were able to reuse those as inputs to our model for training.

During training, the learning rate was 0.00005, and we used a batch size of 32. This means that the Adam optimizer used a weighted 0.00005 to influence its update rate, and the loss function and optimizer were executed every 32 training samples. Our model was trained for 3 epochs over the training data, compared with 50 epochs run on the original ReConstNet. At just 3 epochs, and using just Chamfer Distance in the loss function computation, training took 16 hours to execute. The output of the model is a 1024 point point cloud.

The model was trained using Tensorflow in Python, which provided us with the capability to perform the training on a GPU without extensive development on our parts to leverage the power of the GPU. The GPU used was an Nvidia GTX 1080, which is a Pascal generation consumer graphics card.

IV. RESULTS FOR THE 3D-RECONSTNET

Table I shows the 3D reconstruction results of one image of each object category in the Shapenet dataset. The predicted resolution of the 3D point cloud reconstruction in Table I is 1024 points, compared to a 1024 point ground truth point cloud for each object category. As you can see from the figure, our results came out to be respectable, although not as refined as the original 3D-ReConstNet results. You can see that because of the lack of the Earth Mover's Distance integrated in our loss function, our results are not as sharp as those found in the original paper. There is a lack of refinement

along the edges of each of our predicted point clouds, although the *general* shape of the cloud matches that of the ground truth.

Objectively, our resulting mean Chamfer Distance over the validation split of the Shapenet Dataset was 0.03355, compared to a mean Chamfer Distance of 0.00409 in the original paper. The 10x improvement can be attributed to multiple factors

























- 1) The omission of the Earth Mover's Distance part of the loss function.
- 2) The total training time. The original paper trained their model over 50 epochs, whereas we were bound to 3 epochs of training time.
- 3) A superior understanding of the hyper-parameters associated with training a deep neural network.

A summary of the trade-offs between what the reference paper computed versus what we computed can be seen in Figure 11.

V. CONCLUSION

We faced many challenges throughout the duration of this project. The first of which was understanding a field that none of us had prior experience in (3D Image Reconstruction). It took significant effort to understand the objective of each section of the generated neural network, and how to decompose each section to recreate it from scratch. Along with that, learning industry standard tooling for Neural Networks (Tensorflow and PyTorch) provided a learning curve. After overcoming those barriers, we feel that we have gained a much higher appreciation for the research performed in the original paper, and the research performed in the field collectively. Training a neural network for high quality results is no easy task, and we have only scratched the surface.

TABLE I
ReConstNet RESULTS - PREDICTIONS ON THE LEFT AND GROUND TRUTH ON THE RIGHT FOR EACH CATEGORY

Airplane		Bench	
			
Cabinet		Car	
			
Chair		Lamp	
			
Monitor		Rifle	
			
Sofa		Speaker	
			
Table		Telephone	
			

VI. APPENDIX

A. Daniel Everett

I worked on the following aspects of the project

- 1) Understanding and implementing the Chamfer Distance loss function
- 2) Overcoming the run time setup complications of training the model
- 3) Converting our Google Colab experiments to a trainable model on my personal computer+GPU
- 4) Training the model, including trial and error during the discovery and learning stages of the process
- 5) Accumulating results of the model

B. Britanya Wright-Smith

I worked on the following aspects of the project

- 1) Implementing the encoder feature extraction with ResNet50 network
- 2) Generating the probabilistic sampling vector for the input to the decoder in addition to the Network for the Point Cloud Generation
- 3) Researching on the structure of the ResNet50 network in addition to the point cloud generation

C. Shengjie Guo

I worked on the following aspects of the project

- 1) Researching the architecture and mathematical theory behind the Variation Encoder, and connecting it to the implementation of probabilistic sampling vector
- 2) Understanding and generating a pytorch version of Chamfer Distance function
- 3) Collecting Dataset Pix3d

REFERENCES

- [1] B. Li, Y. Zhang, B. Zhao, and H. Shao, “3d-reconstnet: A single-view 3d-object point cloud reconstruction network,” *IEEE Access*, vol. 8, pp. 83 782–83 790, 2020.
- [2] H. S. H. Fan and L. Guibas, “A point set generation network for 3d object reconstruction from a single image,” *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 605–613, 2017.
- [3] J. Wu, Y. Wang, T. Xue, X. Sun, W. T. Freeman, and J. B. Tenenbaum, “MarrNet: 3D Shape Reconstruction via 2.5D Sketches,” in *Advances In Neural Information Processing Systems*, 2017.
- [4] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, “ShapeNet: An Information-Rich 3D Model Repository,” Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.
- [5] X. Sun, J. Wu, X. Zhang, Z. Zhang, C. Zhang, T. Xue, J. B. Tenenbaum, and W. T. Freeman, “Pix3d: Dataset and methods for single-image 3d shape modeling,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [6] Z. Zhang, “Improved adam optimizer for deep neural networks,” in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–2.