Final Report - Unsupervised learning of shape and pose with differentiable point clouds

Nitesh Gundavarapu
University of California - San Diego
nbgundav@ucsd.edu

Sameeksha Khillan
University of California San Diego
sameeksha.khillan@gmail.com

Raghav K Subramanian
University of California San Diego
rksubram@ucsd.edu

Abstract

This project uses unlabelled category specific images to obtain high-fidelity 3D shape models and pose estimates. This is achieved using deep CNNs with a reprojection loss guiding the unsupervised training objective.

1. Introduction

Historically, learning high-fidelity 3D shapes and pose from their projections has been challenging for two main reasons: i) Pose and shape estimation is a chicken and egg problem. ii) Pose estimation is prone to local minima caused by 2D-3D ambiguity. [3] proposes to tackles the first problem by a joint optimization of shape and pose estimation and the second problem by having an diverse ensemble of pose predictors distilled to a signal student model. Shape representation is done using point clouds, as these are *matter-centric* i.e provide finer details. This approach is computationally efficient and flexible when compared to a memory-heavy dense voxel form which also needs a 3D convolutional decoder, or a mesh representation.

In this project we re-implement the paper and corresponding evaluation protocol in PyTorch as detailed in Sec.3 and train our model at 64^3 occupancy grid resolution for $100k$ iterations and find that the results are comparable to paper's results on Shapenet airplanes and chairs dataset as shown in Tab .1 In Sec. 4.1 and Sec.4.2, we carefully design experiments to figure out the type of objects and geometries that the network fails to learn and the ones the network learns well. In Sec. 5.1, we verify whether the paper's results generalize to other Pascal 3D objects that are not shown in the paper. Specifically, we select lamps data. Finally, in Sec.5.2, we demonstrate sample scene editing applications by taking images to point cloud space, editing the scene in point cloud space and projecting back to the image space.

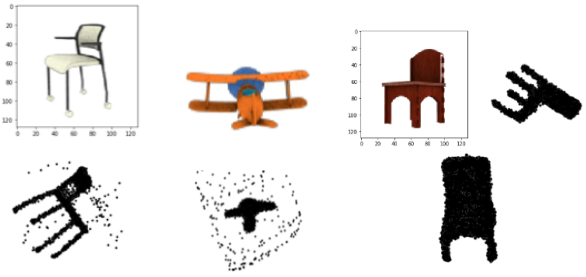

Figure 1. Point cloud renderings for sample images produced using our codebase. The first two images are from our test set, for classes 'chair' and 'airplane' after 100000 epochs and the last image is from our training after 25000 epochs.

2. Model

The task of three-dimensional shape and pose estimation from a single view, can be modeled with or without camera poses for every view. Our dataset D contains views of K distinct objects, with m_i views for the i-th object: D = $\bigcup_{i=i}^{K} \langle x_j^i, p_j^i \rangle_{j=1}^{m}$, where x_j is the color image and p_j is a projection of this image from the same view. Given two images x_1 and x_2 of the same view, we predict 3D shape as $\hat{P}_1 = F_P(x_1, \Theta_P)$ and camera pose as $\hat{c}_2 = F_c(x_2, \Theta_c)$, where F_P and F_c are convolutional neural networks.

3. Implementation details

3.1. Dataset

The original paper[3] uses the ShapeNet dataset with three classes namely chairs, cars and airplanes. We experiment with an additional Pascal3D class, namely lamps, to

study if the results continue to hold.

3.2. Approach

We split the implementation into five steps namely i) Dataloader and batch sampler that samples 4 different views of each object in the mini-batch along with their silhouettes and/or depth maps ii) Prediction module that includes convolutional neural network to predict point cloud, candidate poses, focal length and scale from the image iii) Differentiable rendering module that converts point clouds to voxel grid, predicts ray termination probabilities and projects the object to a view-point as described in the paper [3] iv) Loss functions on depth, silhouette and colored projections v) Training and evaluation. The codebase is made publicly accessible at link.

3.3. Dataloader, Batch Sampler and CNN

For the dataloader and batch sampler, we saved different views of the object, its silhouettes and depth maps in an offline fashion and the dataloader just reads them off the file-system. The main advantage is ease of use and speed while the disadvantage is that it reduces diversity of the data similar to [3]. The images are then passed through an CNN encoder to get a feature vector with 256x4x4 spatial resolution and further through fully connected networks to predict point cloud, poses and scale.

3.4. Rendering and Loss

In the rendering module, we first convert the point cloud to a voxel grid through a trilinear interpolation followed by Gaussian kernel smoothing, described as fast method in [3]. The Gaussian kernel is implemented as a separable kernel to improve the speed of execution while trilinear interpolation is achieved by using repeated *torch._index_put_* operation. This is followed by occlusion reasoning. We use the occupancy of our grid to determine probabilities of occlusion for the points and then project the volume of the plane using these probabilities as described in [3]

$$\mathbf{p}_{k_1,k_2} = \sum_{k_3=1}^{D_3+1} \mathbf{r}_{k_1,k_2,k_3} \mathbf{y}_{k_1,k_2,k_3} \tag{1}$$

Here, r_{k_1,k_2,k_3} are the probabilities and y_{k_1,k_2,k_3} is the signal for the cell at position k_1, k_2, k_3.

Our losses are comprised of a projection loss which computes the standard MSE loss between the rendered image and ground truth silhouette and $L2$ regularization loss with decay of $1e^{-6}$. A snapshot of some of these can be seen in Figure 3. To address the issue of pose ambiguity, we use an ensemble of pose regressors and use the best matching "candidate" which yields a prediction closest to the ground truth for our weight updates.

Figure 2. Ground truth silhouettes and the predicted projections

Model	Paper[3]-128^3 occupancy grid, 16000 points, $600k$ iterations	MVC[3]	Ours - 64^3 occupancy grid, 8000 points, $100k$ iterations
Chairs	4.30	6.51	7.16
Airplanes	3.91	4.43	6.12

Table 1. Chamfer Distance between normalized point clouds with ground truth multiplied by 100. Note: Due to resource constraints, we trained at a lower resolution for a lesser number of iterations using less number of points than the paper.

3.5. Training and Evaluation:

The networks are trained using the Adam optimizer, using 16 samples which include 4 views of 4 objects. The number of points for the predicted point cloud, was based on ground truth resolution and set to 8000 points for 64^2 projections and 128^2 input image size. Dropout is added to ensure an even point distribution on the shape of the 3D objects, and we start from 93% dropout, as shown in Figure 3., reducing it linearly to 0 over training.

For evaluation, we use Chamfer Distance with ground truth between normalized point clouds multiplied by 100 as our metric similar to [3]. Results are presented in Tab.1.

3.6. Implementation Challenges:

In this section, we describe few implementation challenges we faced and our approach to overcome them.

3.6.1 Advanced Indexing - Tensorflow vs Pytorch:

Tensorflow provides advanced indexing for N-dimensional tensors in terms of *scatter_nd* and *gather_nd* functions, which take in input tensors, indexes and updates and applies the updates at the corresponding indexes of the tensor. Although this seems like an in-place operation, complications arise to allow for a differentiable equivalent of this function especially when there are duplicate indices. We want to add

the updates at duplicate indices and this is used in scattering point clouds to voxel grid and achieving trilinear interpolation. The only equivalents we found were *torch._index_put* and *torch._index_put_* operations which allow for accumulation at duplicate indices while allowing differentiability. The former is in-place while latter is out of place. Since the voxel grid size is large, in-place operation is much faster than creating out of place copy. Unfortunately, there is no *torch._index_add_* that allows accumulation at duplicate indices we resorted to the out of place variant.

3.6.2 Debugging and Tensorboard:

Since the codebases are large, we faced a huge debugging challenge as a minor error can put off the entire training process. To handle this, we resorted to unit testing where we verified the behavior of each major function is a) as expected b) tracks tensorflow codebase. We also wrote unit tests for the gradients to ensure accuracy. Since the training is slow and we have a lot of parameters to tune, namely the smoothing kernel sizes, point cloud dropout probabilities, smoothing kernel standard deviations, learning rates, loss weights etc., we used tensorboard extensively. Torch 1.4 provides tensorboard support and the link can be established by using *tb-nightly* module. In Figure 4., we show a screenshot of our tensorboard and training progress.

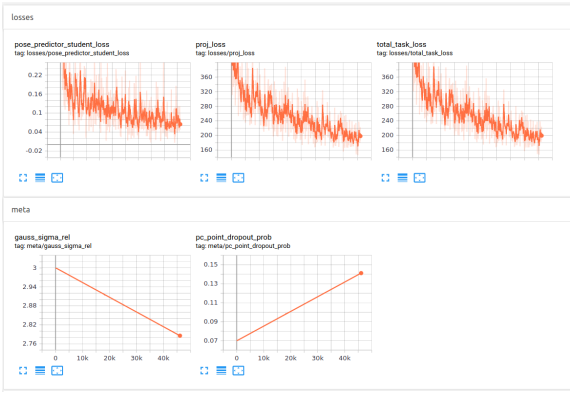

Figure 3. Tensorboard Progress

4. Experimentation

In this section, we analyze how our trained model performs on different categories and poses of input chair images.

4.1. Object Category Splits:

Shapenet chairs dataset is composed of a variety of chairs such as ones with thin legs, bean bags, sofas, long chairs short chairs etc. Since we don't have the category label available, we cluster the input chair images by first taking them to a low-dimensional space using Principal Component Analysis, apply K-Means clustering with $K = 5$ and report the chamfer distance for each group and the perceived visual category of the images. The results are presented in Tab 2

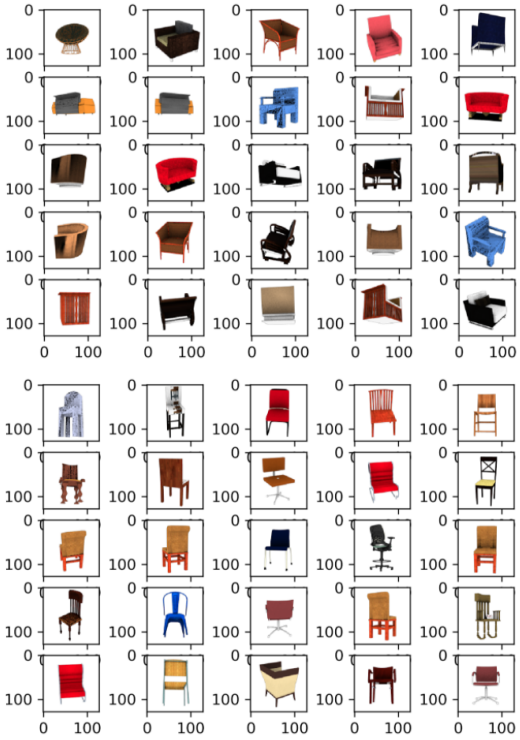

Figure 4. Grouping by Object Category

We observe that in cluster with thicker objects, we have poor mean Chamfer distances since the same number of points in the point cloud are distributed among a bigger volume which increases the distance between their nearest neighbours from the ground truth point cloud.

Group	Visual Category	Chamfer
Group 1	Sofas,high volume chairs	10.01
Group 2	Very Thin legs, grey in color	6.45
Group 3	Thick legs and heavy bottoms, red color	7.06
Group 4	Long Legs and long chair	6.72

Table 2. Images are first clustered into groups using K-Means, and mean chamfer for each group is reported.

4.2. Viewpoint Splits:

The input chair images can be from different view-points and the network may not perform well for all the different

view-points. So, we use the yaw, pitch and roll information to split the input images into groups and report the Chamfer distance for each sub-group. Note that this is implemented as a sub-grouping over the groups from previous experiment so as to understand if different categories of objects work well at different view-points. Sample results are presented in Figure 5 and Tab. 3

Group	Orientation	Chamfer
Group 1	Yaw [1.57, 2.35], Pitch [5.49, 6.28]	7.75
Group 2	Yaw [2.35, 3.14], Pitch [5.49, 6.28]	9.16
Group 3	Yaw [1.57, 2.35], Pitch [0.0, 0.78]	6.13

Table 3. Mean chamfer distances based on Yaw and Pitch based division

We observe that pitch and yaw have a significant effect on the quality of the point cloud learnt and everything is consistent with the quality of the view, as in how clearly all details of the object are visible. For example yaw angles closer to the multiples of π have worse chamfer distances, whereas those having yaw close to $\frac{\pi}{2}$ are better. Similarly, the pitch which is closer to a top viewing angle is better than when we view the object from the bottom.

5. Extension:

5.1. Generalization:

We experiment on Pascal3D lamps data and train our model. The input lamp mask and the projected lamps mask are presented for various types of lamps in Fig.6.

We observe that the conclusions and limitations from previous sections follow through to this new data. Namely, the training generalizes well as we can see from the renderings. Further, when the geometry is complex, the renderings are poor (see top right image in Fig. 6)

5.2. Scene Editing Application:

Scene editing is an important application for various e-commerce and real-estate applications. For example, a furniture shop can show the chairs and sofas placed at different locations and orientations in a virtual scene. Real-time scene editing given only 2D images from unknown view-points is an interesting problem in this line.

In this regard we propose the following mechanism: i) Extract point clouds from input images ii) Place the point-clouds in the normalized space with required orientation, scale and translation iii) Project the point-cloud back to image space. Illustrations are shown in Fig.7 and gif visualizations are provided in the github codebase.

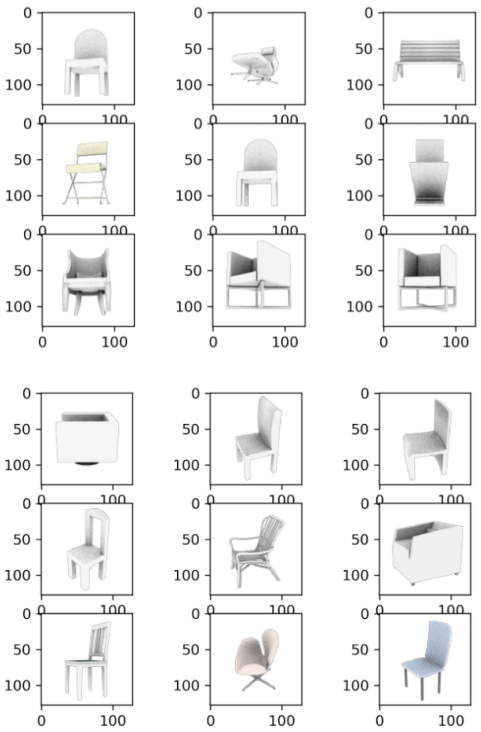

Figure 5. Groups 2(Yaw [2.35, 3.14], Pitch [5.49, 6.28]) and 3(Yaw [1.57, 2.35], Pitch [0.0, 0.78])

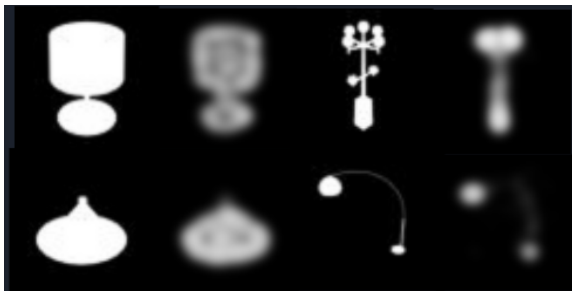

Figure 6. Silhouette images for lamps and corresponding predicted projections

6. Limitations

Our implementation is slightly slower than original tensorflow implementation because of indexing limitations discussed in Section 3.6.1.

7. Conclusion and Future Work

In this report, we have re-implemented the [3] in PyTorch and obtained comparable results to the paper, justifying the correctness. Subsequently, we analyzed the performance of the method on various visual categories and view-points to see where the network is lacking. Further, we extended the implementation to lamps data that is not reported in the paper and finally, we provided a scene-editing

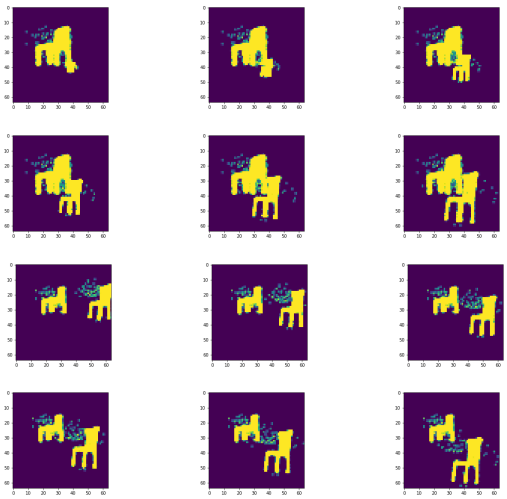

Figure 7. Scene editing by changing scale of the chairs in top 6 images, and by translating chairs in bottom 6 images

application as an immediate extension of the paper.

In future, making the point clouds sparser can be a good exploration since it takes a lot of computation power. In making the point clouds sparse, we can do corner detection on multiple projections from the point cloud and identify critical points and the neighbourhoods around them, which can then be assigned a lower dropout probability. To learn the inverse mapping from sparse point cloud to dense, we propose to use a neural network with an encoder-decoder architecture with the results of forward mapping as ground-truth.

The current training mechanism trains one model per object. A research direction could be to see if a single model can be used train for multiple objects. In this regard, we can replace the FC layers in predicting point clouds with deconvolution architecture to exploit common local structures present across objects.

Another interesting direction is to make the network predict smoothly varying shapes and poses for smoothly varying inputs. In this regard, we can impose smoothness in the loss-function during training time on the point cloud in chamfer sense and on the projections in L2 sense.

Finally, this paper trains only using synthetic data. An interesting direction to explore is to adapt to real datasets. Feature level domain adaptation can be performed in an adversarial way using method such as [1] or a CycleGAN can be used to perform domain mapping at image space[2].

References

[1] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[2] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213*, 2017.

[3] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. *NIPS*, 2018.