Supplementary Material: Novel Object Viewpoint Estimation through Reconstruction Alignment

Mohamed El Banani Jason J. Corso David F. Fouhey
University of Michigan
{mbanani,jjcorso,fouhey}@umich.edu

A. Implementation Details

We implemented our approach in the PyTorch framework [6] and it is made publicly available.[1] Below we present more details on our network architectures and training procedure.

A.1. Network Architecture

We present the architectures of our Shape, 2D Readout, and Discriminator networks. Our Shape Network consists of 3 components: 2D Encoder, 3D UNet, and projection layers. Unless otherwise stated, each convolution and linear layer is followed by batch normalization [3] and a leaky ReLU activation with a negative slope of 0.01.

2D Encoder Our encoder consists of 3 successive `double_conv` modules. Each `double_conv` consists of 2 convolution layers with a kernel size of 3×3 and padding of 1. Each `double_conv` doubles the number of features (with the exception of the first one going from 3 to 8), and reduces the size of the map by a factor of 2 in each spatial dimension using max pooling. The final encoding has a size of $batch \times F \times N \times N$ where $F = N = 32$.

Projection We re-implement the projection layers proposed by Kar *et al*. [4] in PyTorch. We use an orthographic projection to back-project the 2D feature maps to a 3D grid of size $F \times N \times N \times N$, where $N = F = 32$. We use tri-linear sampling to determine the features for each voxel when doing back-projection. When projecting from 3D back to 2D, we sample Z slices along the depth dimension ($Z = 64$) and concatenate all the features along the feature dimension. We use a 1×1 convolution to aggregate the features back from $Z * F$ to F features for every pixel location. After aggregation, our feature map has the same size as the output of the 2D encoder.

[1] https://github.com/mbanani/novelviewpoints

3D UNet We extend the UNet architecture into 3D by extending the kernels along the depth dimension. The 3D UNet architecture is very similar to the 2D Encoder, with the exception of having 3D convolutional kernels and using skip connections between the encoder and the decoder. One difference is that we only use 3 scales for the 3D UNet instead of 4 scales as done by the encoder.

2D Readout The 2D readout network takes as input the projected 2D features and converts them into either a depth map or an object mask. We do this by applying 2 3×3 convolution layers with an $8\times$ bi-linear up-sampling step in between. The 2D readout has an input feature dimension of 32, which gets reduce to 16 after the first layer, and finally to 1 for the output prediction. We do not apply normalization or activation to the final layer.

Discriminator We implement the discriminator as a 3D CNN. Our architecture makes use of the inception module proposed by Szegedy *et al*. [8], adapting it to 3D by expanding the kernels along the depth dimension. Our network architecture consists of two inception modules followed by two fully-connected layers. Each inception module consists of 3 convolution layers and a max-pooling layer that are applied separately to the input. The convolution layers have kernel sizes of 1, 3, and 5. Max pooling has a kernel size of 3. We use zero-padding to ensure that all 4 operations have the same sized output. All the outputs are concatenated, and are followed by a linear layer that aggregate each voxel's features from $4F$ back to F features. The first inception modules maps the features of each voxel from 32 to 16, while the second reduces it to 1.

Our network uses 2 inception modules, with 3D average pooling between them to scale down the voxel grid by a factor of 2. After both inception modules, the voxel grid has size of 16 and feature dimension of 1. We flatten the output and use two linear layers to reduce the number of features to 512, then to 4 (magnitude and Euler angles). We do not apply normalization or activation to the final layer.

A.2. Training Procedure

We initialize all of our networks using Kaiming initialization as proposed in [2]. We use a uniform distribution for linear layers, and a normal distribution for convolutional layers. All the networks are trained using the Rectified Adam optimizer [5] with a learning rate as 10^{-3}, epsilon as 10^{-4}, weight decay as 10^{-6}, and beta values as 0.9 and 0.999. We use Rectified Adam with its default parameters as it has been shown to be robust to choice of learning rate. While we do not observe overfitting to the training set due to the large variance in the dataset as previously observed for rendered datasets [7], we do observe a general trend of diminishing improvements after 20k iterations. As a result, we train for 100k epochs and choose the best performing model on the validation set. While we use a batch size of 16 for our networks due to GPU size considerations, we observe that increasing the batch size for the baselines to 32 improved their performance, so we report those results instead.

B. Rendered Datasets

We use the same rendering pipeline for all of our datasets which is adapted from the Render For CNN pipeline [7] which uses Blender [1]. We uniformly sample azimuth and elevation angles for each model to ensure minimal bias in the dataset. For training and evaluating our models, we randomly pick a subset of all possible pairs or triplets. The distribution of views over azimuth and elevation is presented in Figure 1. We use object materials for rendering when possible, and overlay all images with backgrounds from the SUN dataset [9].

Furthermore, we present visualizations of our instances with the predicted 3D view mask for ShapeNet (Figure 2), Pix3D (Figure 3), and Thingi10K (Figure 4). While the model sees the objects with an overlaid background, we omit the background here for better visibility. Examples with overlaid background can be seen in the paper. The results reveal some failure modes such as missing thin objects (eg, chair legs), failing to register an object part that was occluded in one of the reference views, and filling in small holes in the object.

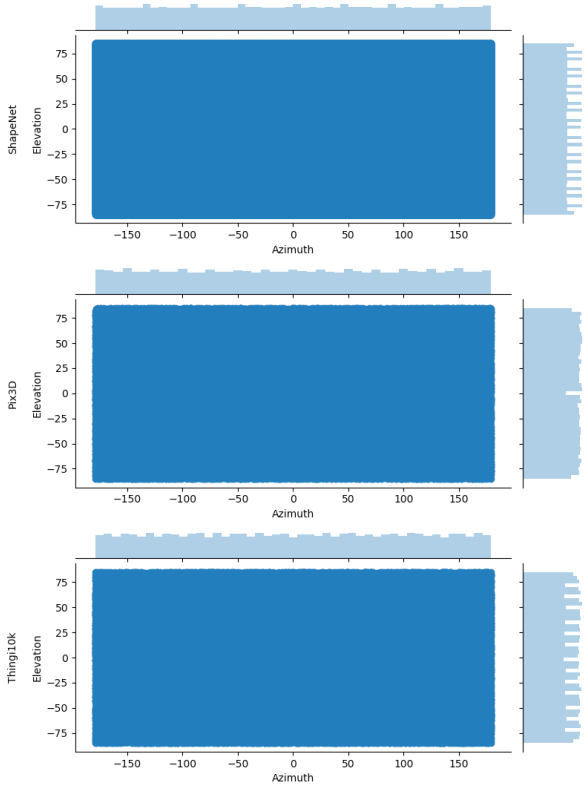

Figure 1. **Viewpoint Distribution for the 3 datasets.** We present the viewpoint distribution for the 3 datasets. As shown, the viewpoints are sampled uniformly, which deviates from other datasets which aim to match the distribution of a target real image datasets to aid generalization.

References

[1] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2018.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[4] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. In *NeurIPS*, 2017.

[5] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint*, 2019.

[6] Adam Paszke, Soumith Chintala, Ronan Collobert, Koray Kavukcuoglu, Clement Farabet, Samy Bengio, Iain Melvin, Jason Weston, and Johnny Mariethoz. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration, 2017.

[7] Hao Su, Charles R. Qi, Yangyan Li, and Leonidas J. Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *ICCV*, 2015.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[9] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.

Figure 2. **Visualization Results for ShapeNet.** We visualize the results of the model trained on ShapeNet for mask prediction on ShapeNet. **Background is removed for better visibility.**

Figure 3. **Visualization Results for Pix3D.** We visualize the results of the model trained on ShapeNet for mask prediction on Pix3D. **Background is removed for better visibility.**

Figure 4. **Visualization Results for Thingi10K.** We visualize the results of the model trained on ShapeNet for mask prediction on Thingi10K. **Background is removed for better visibility.**