Neural-Sim: Learning to Generate Training Data with NeRF (appendix)

Yunhao Ge[1], Harkirat Behl[2*], Jiashu Xu[1*], Suriya Gunasekar[2], Neel Joshi[2], Yale Song[2], Xin Wang[2], Laurent Itti[1], and Vibhav Vineet[2]

[1] University of Southern California
[2] Microsoft Research

1 Implementation Details

1.1 Visualization of reparametrization of pose sampling

We visualize the reparametrization of pose sampling in Fig. 1 as discussed in the Section 3.1 Tool 1 in the main paper.

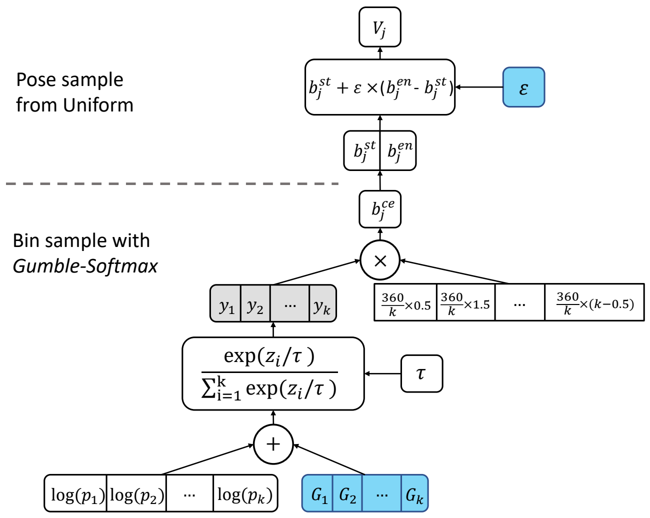

V_j
Pose sample from Uniform
$b_j^{st} + \varepsilon \times (b_j^{en} - b_j^{st})$
ε
b_j^{st} b_j^{en}
b_j^{ce}
Bin sample with *Gumble-Softmax*
$\times$
y_1 y_2 ... y_k
$\frac{360}{k}\times 0.5$ $\frac{360}{k}\times 1.5$ ... $\frac{360}{k}\times(k-0.5)$
$\frac{\exp(z_i/\tau)}{\sum_{i=1}^{k}\exp(z_i/\tau)}$
τ
$+$
$\log(p_1)$ $\log(p_2)$ ... $\log(p_k)$
G_1 G_2 ... G_k

Fig. 1: Reparametrization of pose bin sampling: We first discretize the pose space into a set of k bins, which we will then sample to generate the view parameters for the NeRF. To backpropagate through the sampling process, we approximate the sample from the categorical (i.e. bin) distribution by using a Gumble-softmax "reparameterization trick". Within each bin we sample uniformly.

* Equal contribution as second author

1.2 Memory Efficiency

Tool 2: Twice-forward-once-backward As discussed in the main paper Sec. 3.1 (Tool2), the full gradient update of our bi-level optimization problem involves using the approximation of ∇_{NeRF} in Eq. 5 and back in Eq. 2. There are three terms in this computation with the following dimensions:
(1) $\frac{\partial(\frac{\partial l(x_j,\hat{\theta}(\psi_t))}{\partial\theta})}{\partial x_j} \in \mathbb{R}^{m\times d}$, (2) $\frac{\partial x_j}{\partial\psi} \in \mathbb{R}^{d\times k}$, and (3) $\nabla_{TV} = \mathcal{H}(\hat{\theta}(\psi_t),\psi)^{-1}\frac{d\mathcal{L}_{val}(\hat{\theta}(\psi_t))}{d\theta} \in \mathbb{R}^{m\times 1}$, where $m = |\theta|$ is the # of parameters in object detection model, d is the # of pixels in x, and k is # of pose bins.

Specifically, if we follow the sequence of (3)-(1)-(2), first, the output of (3) is a $1 \times m$ vector and can be used as the weight to compute (1) with Pytorch weighted autograd. In this case, we do not need to explicitly store the huge matrix ($\mathbb{R}^{m\times d}$) of (1) and the corresponding large computation graphs of each element in it. Similarly, the result of previous step ((3)-(1)) is a $1 \times d$ dimension vector and we can use it as the weight to compute (2) with Pytorch weighted autograd. Finally, we obtain the gradient as a $1 \times k$ dimension vector.

Tool 3: Patch-wise gradient computation As discussed in the main paper Sec. 3.1 (Tool3), patch-wise gradient computation helps to save the memory cost of computing (1)-(2) sequence of the gradient. Table. 1 shows the details about the memory cost when we use different patch sizes. Specifically, if we keep NeRF chunk as 512 fix, when we increase the patch size by multiplying 2 each time, the memory cost of gradient computation will also approximately doubled. With Patch-wise gradient computation, image size would not be the bottleneck of gradient computation.

Patch Size	Gradient Computation Memory Cost	Total Memory Cost
512 (32x16)	1910 MB	6450 MB
256 (16x16)	974MB	5514MB
128 (16x8)	464MB	5004MB
64 (8x8)	216MB	4756MB

Table 1: Patch-wise optimization memory cost comparison with different patch-size, for the total memory cost we fix the NeRF chunk as 512.

1.3 Comparison with the graphics pipeline

We provide additional quantitative results to demonstrate that NeRF can replace traditional graphics pipelines like BlenderProc [2] when generating data for downstream computer vision tasks such as object detection. Results are provided in the Fig. 2.

To test this, we consider objects from YCB-video datasets. We render images from NeRF and BlenderProc [2] using the same camera pose and zoom parameters. We use these images to conduct multi-class object detection tasks under the same training and test setting. As shown NeRF generated data can achieve the same accuracy as that of BlenderProc on downstream object detection tasks.

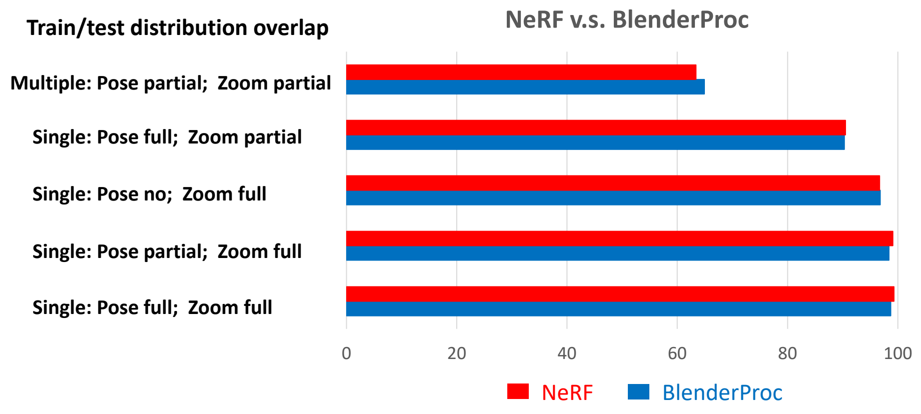


Fig. 2: Object detection performance using NeRF and Blender synthesized images. The X-axis is mAP. Single means each test image contains one object, multiple means each test image contains multiple objects.

1.4 Influence of optimization parameters

We briefly describe the effects of different parameters used in our bilevel optimization updates. In particular, we show the effect of Gubmel softmax temperature parameter used in the main paper. Fig. 3 shows the Gaumble softmax performance under different temperatures. If the temperature parameter is very large, initial categorical distribution takes form of uniform distribution after Gumbel updates and at a lower temperature, the distribution becomes peaky. In our experiments, we have used a parameter value of 0.1.

We used stochastic gradient descent with momentum for ψ parameter updates with learning rates of $1e$-5 and momentum value of 0.9 value. Further, on $YCB-video$ dataset, we use warm start to conduct experiment.

1.5 Optimization Runtime

We now provide running time details of our end-to-end pipeline. Each iteration involves data generation through NeRF, detection model training, backpropagation through detection model including hessian-vector product evaluation, and backpropagation through data generation process. For $YCB-synthetics$ experiments described in Sec.4.2 in the main paper, it takes roughly ten minutes to complete one end-to-end computation. Further, time depends on the image resolution generated by NeRF, and detection model training.

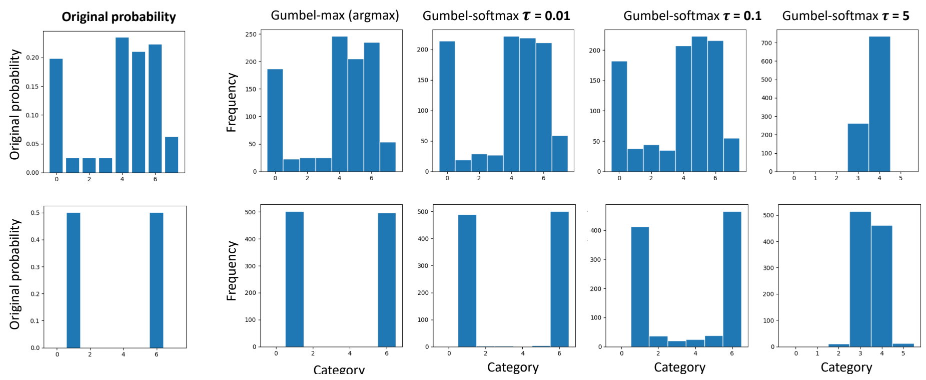


Fig. 3: Effect of Gubmel softmax temperature parameter used to approximate the initial distribution.

Finally, it should be noted that this pipeline does not involve any human effort. In comparison, the traditional graphics pipeline will involve human expert involvement for creating good 3D models of objects.

1.6 Rendering from SFM

In order to generate images from a traditional graphics pipeline, one needs to have accurate 3D object models including accurate geometry, texture, materials of objects. Capturing these accurate properties of the objects requires human expert involvement. However, if we use a standard computer vision pipeline like structure-from-motion [5] pipeline to generate 3D models, the quality of images generated by these models are not as high as that of NeRF. Please refer to the Fig. 4. Thus involving human experts to improve 3D model quality for traditional graphics limits their scalability, and is also expensive. In contrast, NeRF only requires images along with camera pose information, providing benefits over traditional graphics pipelines.

1.7 NeRF-in-the-wild

Fig. 5 shows the results of NeRF-in-the-wild (nerf-w) on controllable illumination change which allows smooth interpolations between color and lighting. The experiments have been conducted on the YCB-objects. For each object, we conduct interpolations between the appearance embedding of two training images (left, right), which results in rendering (middle) where illumination are interpolated but geometry is fixed.

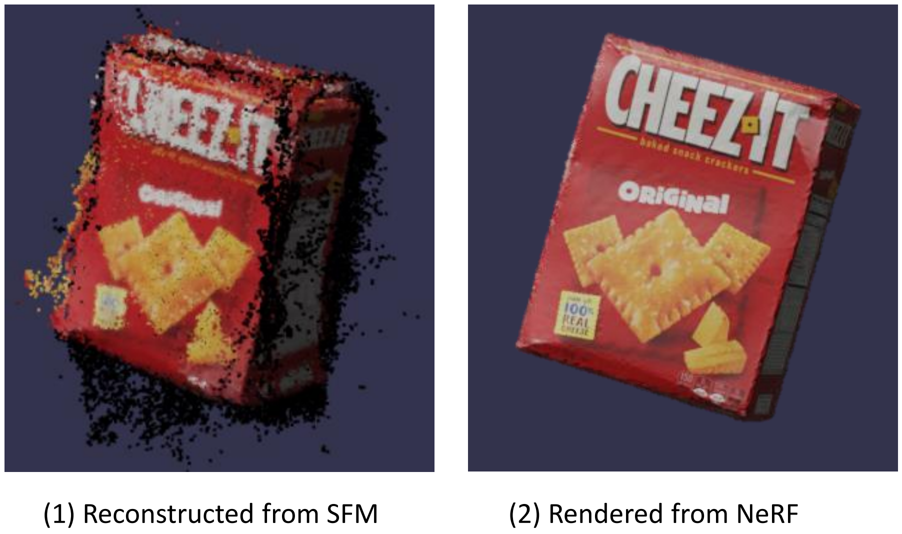


Fig. 4: Quality of images rendered using 3D model generated using a standard structure-from-motion pipeline. In comparison, NeRF can generate high-quality images without needing high-quality 3D models.

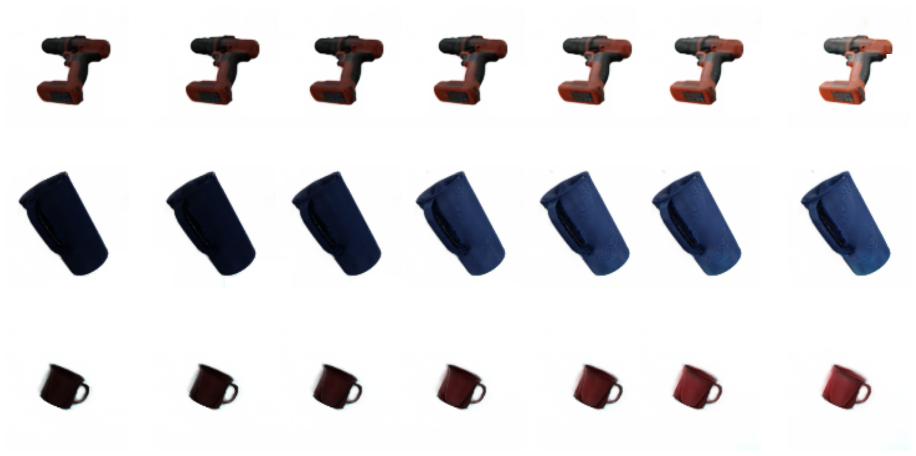


Fig. 5: NeRF-in-the-wild illumination rendering control on YCB objects. Interpolations between the appearance embedding of two training images (left, right), which results in rendering (middle) where illuminations are interpolated but geometry is fixed.

2 Dataset Information

2.1 YCB object visualization

Experiments have been conducted on 21 objects from YCB-video datasets. These objects are: *masterchef can, cracker box, sugar box, tomato soup can, mustard bottle, tuna fish can, pudding box, gelatin box, potted meat can, banana, pitcher base, bleach cleanser, bowl, mug, power drill, wood block, scissor, large marker, large clamp, extra large clamp, foam brick*. These objects are showed in Fig. 6.

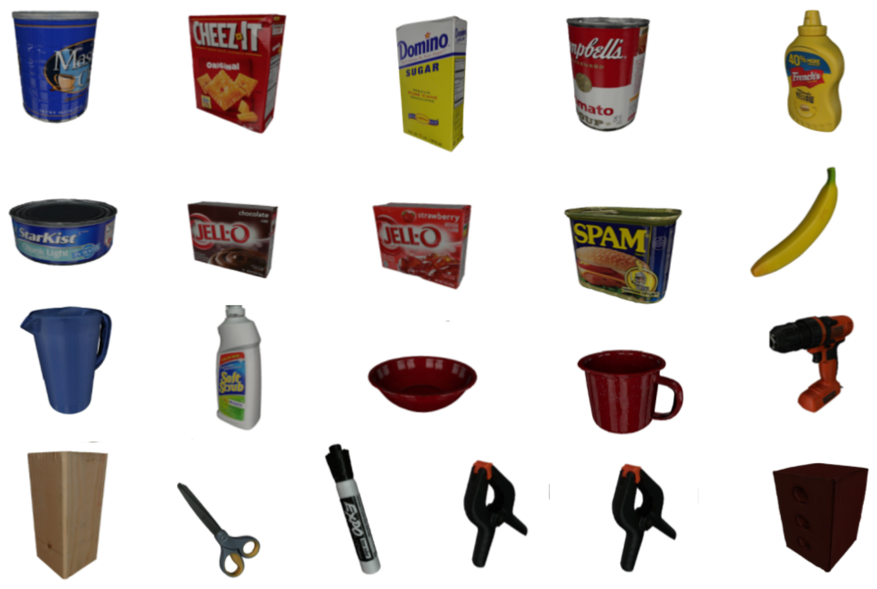


Fig. 6: Object from YCB-video dataset [6]. *master chef can, cracker box, sugar box, tomato soup can, mustard bottle, tuna fish can, pudding box, gelatin box, potted meat can, banana, pitcher base, bleach cleanser, bowl, mug, power drill, wood block, scissor, large marker, large clamp, extra large clamp, foam brick*

2.2 YCB synthetic dataset details

In order to train NeRF, we first use 3D YCB object models from the BOP-benchmark page [3]. We use BlenderProc [2] to generate 100 images per object. These images are captured from poses that are sampled from a uniform distribution. These images along with their corresponding pose values are used to train NeRF. NeRF training takes almost 20 hours for each object.

2.3 YCB-in-the-wild dataset

As described in the main paper, in order to evaluate the performance of the proposed NS and NSO approaches on a real-world dataset, we have created a real-world YCB-in-the-wild dataset. The dataset has 6 YCB objects in it, which are the same as in the YCB-synthetic dataset: *masterchef, cheezit, gelatin, pitcher, mug, driller*. All images are captured using a smartphone camera in common indoor environments: living room, kitchen, bedroom and bathroom, under natural pose and illumination. We manually labelled each image with object bounding boxes. Further, to explore the effect of distribution shifts on the object detection task, we manually labelled the object pose in each image using the the same eight bins discussed earlier. The dataset consists of total around 1300 test images with each object having over 200 images. Images from the dataset capturing different environment properties are shown in Fig. 7.

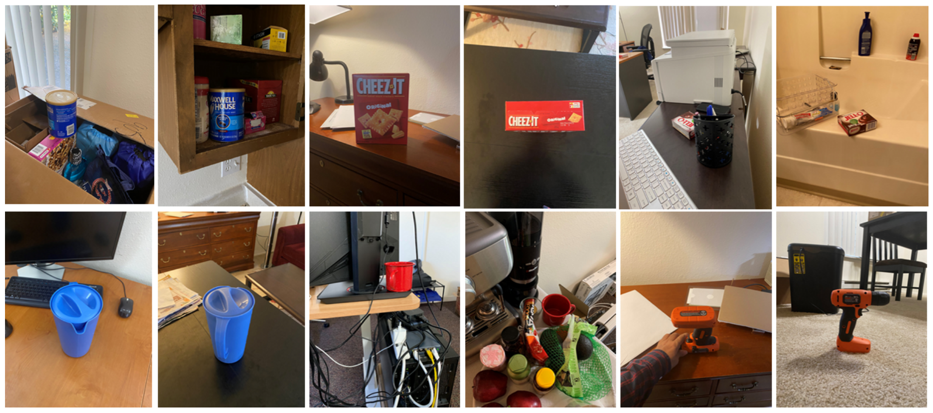


Fig. 7: Images from our YCB-in-the-wild dataset. It consists of six YCB objects *masterchef, cheezeit, gelatin, pitcher, mug, driller* captured in common indoor environments: living room, kitchen, bedroom, bathroom.

2.4 YCB-video dataset

Images from the YCB-video dataset captured in different scenes are shown in Fig. 8.

3 Additional Experiments

We provide additional experiments below.

Fig. 8: Example images from the YCB-Video test set.

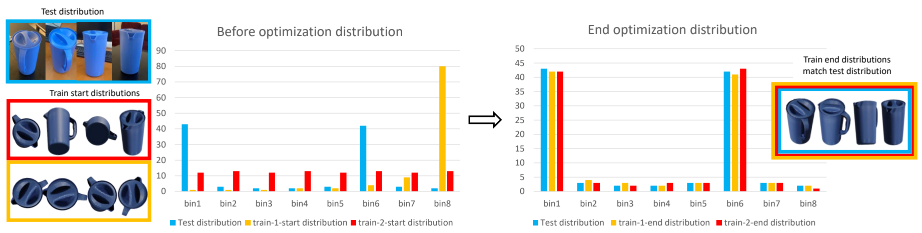


Fig. 9: Interpretability results on multi-modal test distributions. Visualization provides evidence that the proposed Neural-Sim (NSO) approach generates interpretable outputs. In the shown example, test images are sampled from multi-modal distribution bin 1 and bin 6 as dominant bins. For Neural-Sim optimization (NSO), initial training pose distributions are uniform and bin 8 as dominant bin. Observe the bin distribution at the optimization - the final bin distribution at the end of Neural-Sim (NSO) training matches with the test bin distribution.

3.1 Interpretable experiments visualization

In order to support the claim that the proposed Neural-Sim Optimization (NSO) approach can learn interpretable results, we provide additional visualization on the YCB-in-the-wild dataset illustrated in Fig. 9, Fig. 10 and Fig. 11. In particular, we demonstrate interpretability of our method on two scenarios. First, we conduct experiments where test images are generated from multi-modal distributions - two modal and three modal distributions. Second, we also show results on zoom experiments on cheeze box and driller objects. In both these experiment setup, we consider two starting bin distributions for training: a uniform distribution and a randomly selected bin as a dominant bin. As shown in the figure, we observe that no matter what the starting training distributions our NSO approach use, the final learnt distributions match with the test distributions.

Finally, we also provide qualitative comparison between images generated from the learned distributions using the proposed NSO approach and the baseline Auto-Sim approach. Fig. 12 provides visualization for *Cheeze box* and Fig. 13 for *pitcher* object. We have visualized eighteen images sampled from the learned distributions from the NSO and Auto-Sim approaches. We observe that the proposed NSO approach can generate images that resemble the test images in both these objects. However, Auto-Sim generates images where objects are not always aligned with test images. These visualizations provide ample evidence

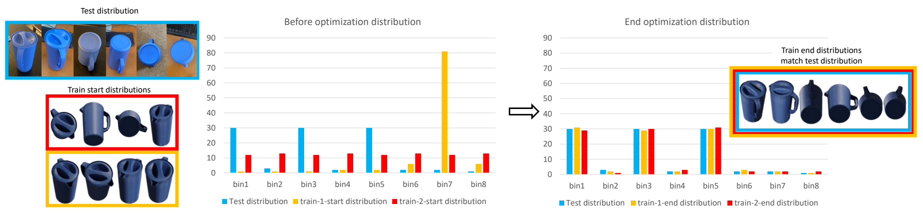


Fig. 10: Interpretability results on multi-modal test distributions. Visualization provides evidence that the proposed Neural-Sim (NSO) approach generates interpretable outputs. In the shown example, test images are sampled from multi-modal distribution bin 1, bin 3 and bin 5 as dominant bins. For Neural-Sim optimization (NSO), initial training pose distributions are uniform and bin 7 as dominant bin. Observe the bin distribution at the optimization - the final bin distribution at the end of Neural-Sim (NSO) training matches with the test bin distribution.

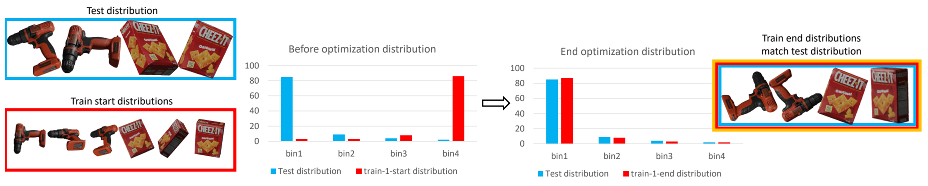


Fig. 11: Interpretability results on zoom test distributions. Visualization provides evidence that the proposed Neural-Sim (NSO) approach generates interpretable outputs on driller and cheeze box. Observe the bin distribution at the end of the optimization - the final bin distribution at the end of Neural-Sim (NSO) training matches with the test bin distribution.

to support the significant improvement in performance achieved by our NSO approach over the baseline.

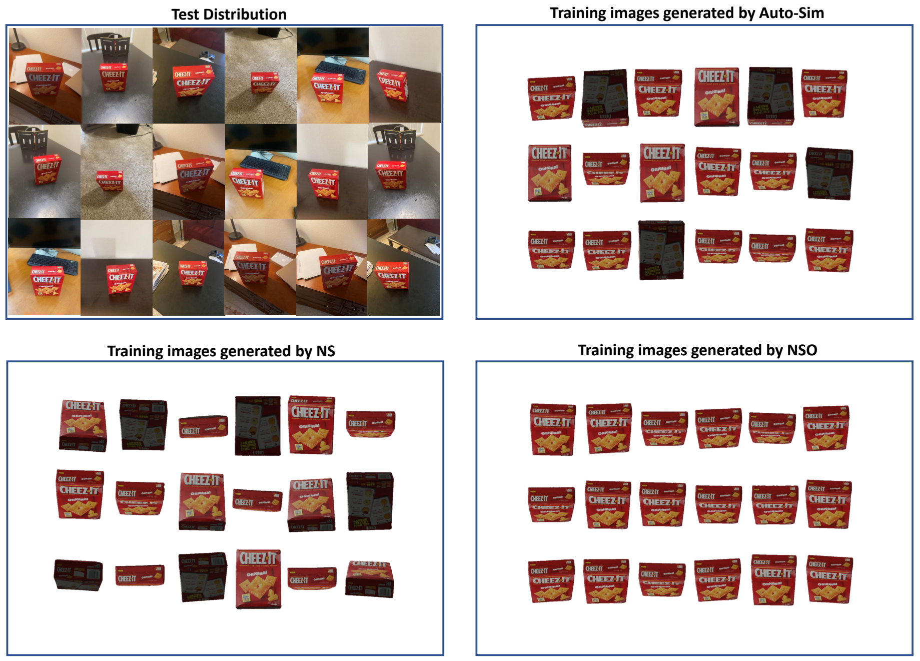

Test Distribution
Training images generated by Auto-Sim
Training images generated by NS
Training images generated by NSO

Fig. 12: Visualization of cheeze box images generated from the learned distributions using the proposed NSO approach and the baseline Auto-Sim approach. Observe how images generated from our approach align with the test distribution. In comparison, there are many noisy samples in the Auto-Sim approach.

3.2 Detection visualization

Object detection results from our pipeline on YCB-in-the-wild and YCB-video datasets are shown in Fig. 14.

3.3 Full YCB-Video dataset results

If we use 100% full YCB-Video training images to train RetinaNet, the mean Average Precision (mAP) reaches 58.5% on all 21 classes. After we use NSO with combine optimization which combines both real-world training images and NeRF synthesized images into training and optimization, the accuracy can improve from 58.3% to 58.8%

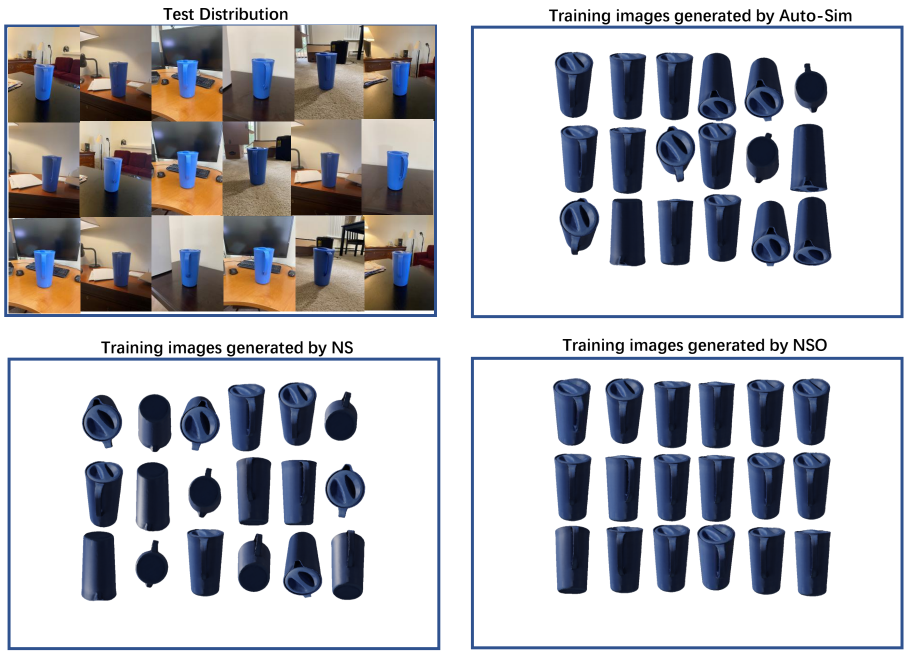
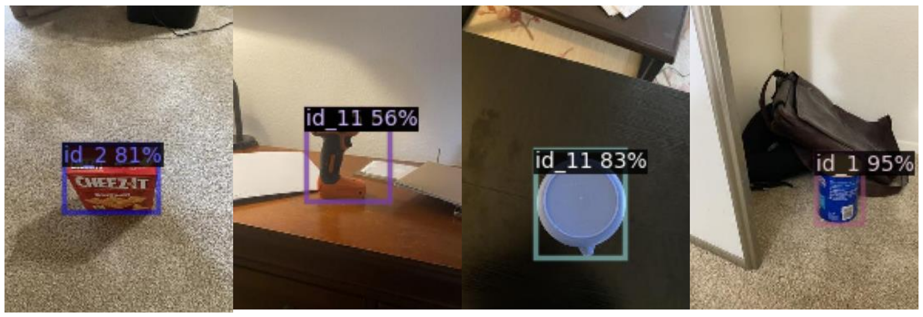
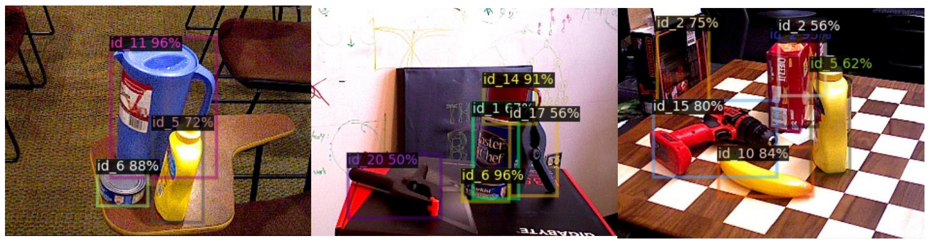

Test Distribution
Training images generated by Auto-Sim
Training images generated by NS
Training images generated by NSO

Fig. 13: Visualization of pitcher images generated from the learned distributions using the proposed NSO approach and the baseline Auto-Sim approach. Observe how images generated from our approach align with the test distribution. In comparison, there are many noisy samples in the Auto-Sim approach.



(a) YCB-in-the-wild dataset



(b) YCB Video dataset

Fig. 14: Visualization for detection results on YCB-in-the-wild and YCB-Video datasets.

3.4 Extension to ObjectNet dataset

We conduct experiments on ObjectNet [1] dataset, which is a large real-world dataset for object recognition. The dataset consists of 313 object classes with 113 overlapping ImageNet classes. In order to synthesize training images, we use CO3D [4] dataset that provides data to train NeRF. There are 17 classes that overlap with ImageNet and ObjectNet classes. After we trained NeRFs for these classes, we find by using NeRF synthesized data to finetune an ImageNet pretrained model provides a 4% improvement on the 17 ObjectNet classes.

4 Limitations and Dataset copyright

In this work, we have focused on optimizing camera pose, zoom factor, and illumination parameters. In the real world, there are other scene parameters that affect accuracy, like materials, etc. However, our approach can be extended to include other parameters by incorporating new advances in neural rendering.

Dataset copyright. We used publically available data. The YCB-Video dataset is released under the MIT License. Further, we will release our YCB-in-the-wild dataset under the creative commons license.

Societal impact Our work focuses on using neural rendering for generating images for solving downstream computer vision tasks. This provides an opportunity to reduce reliance on human or web-captured training data that has potential privacy issues.

References

1. Barbu, A., Mayo, D., Alverio, J., Luo, W., Wang, C., Gutfreund, D., Tenenbaum, J., Katz, B.: Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. Advances in neural information processing systems **32** (2019)
2. Denninger, M., Sundermeyer, M., Winkelbauer, D., Zidan, Y., Olefir, D., Elbadrawy, M., Lodhi, A., Katam, H.: Blenderproc. arXiv preprint arXiv:1911.01911 (2019)
3. Hodan, T., Michel, F., Brachmann, E., Kehl, W., GlentBuch, A., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., et al.: Bop: Benchmark for 6d object pose estimation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018)
4. Reizenstein, J., Shapovalov, R., Henzler, P., Sbordone, L., Labatut, P., Novotny, D.: Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 10901–10911 (2021)
5. Schonberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2016)
6. Xiang, Y., Schmidt, T., Narayanan, V., Fox, D.: Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. arXiv preprint arXiv:1711.00199 (2017)