Final Report for Deep Learning (2023) Course Project: Data Augmentation with Instant-NeRF

Minxuan Qin [*1] **Haitao Yu** [*2] **Zhiyuan Huang** [*3]

Abstract

Despite classical data augmentation methods like random flip and clip, Neural Radiance Fields (NeRFs) show another possibility to synthesize novel view images to augment datasets for computer vision tasks. However, the cost of training a classical NeRF is time-consuming. In our work, we integrate the state-of-the-art Instant-NGP into the Neural-Sim pipeline, which could be trained only in a few minutes and help to increase the downstream task performance.https://github.com/Instant-Sim/Instant-Sim.

1. Introduction

Neural Radiance Fields (NeRFs), a state-of-the-art implicit volumetric scene representation method, have exhibited enormous potential in view synthesis. Under this general setting, NeRFs show the possibility of being a provider in any multiple-view desired task, for instance, training tasks of some computer vision(CV) models. Particularly, Neural-Sim (Ge et al., 2022) has shown the potential of data augmentation by NeRF in the object detection task. Nevertheless, NeRFs'(Mildenhall et al., 2021)(Yen-Chen, 2020) training and evaluation are slow to be practically integrated into such workflows until Instant-NGP is proposed (Müller et al., 2022) by NVIDIA. In this work, our main contribution is replacing NeRF's usage in Neural-Sim's optimization pipeline with Instant-NeRF (Tang, 2022) and testing its performance on a customized hand gesture blender dataset.

[*]Equal contribution [1]Department of ITET, ETHz, Zurich, Switzerland [2]Department of Mathematics, ETHz, Zurich, Switzerland [3]Department of INFK, ETHz, Zurich, Switzerland. Correspondence to: Minxuan Qin <minqin@student.ethz.ch>, Haitao Yu <haityu@student.ethz.ch>, Zhiyuan Huang <zhiyhuang@student.ethz.ch>.

Deep Learning, ETH Zürich, 2022

2. Models and Methods

2.1. Neural Radiance Field (NeRF) models

2.1.1. First NeRF

A Neural Radiance Field (NeRF) (Mildenhall et al., 2021)(Yen-Chen, 2020) is a fully connected neural network that can generate novel views from 2D images with camera pose and intrinsic information. First, it uses a simple neural network to learn an implicit mapping from viewing direction and space location (5D input) to volume density and color (4D output). Subsequently, it renders 2D images using the same formula of volume rendering in computer graphics. Finally, it computes the rendering loss to train the network. In addition, it encodes positions to capture high-frequency information and hierarchical samples by the rendering process to improve performance.

2.1.2. Instant-NeRF

NeRFs are implicit scene presentations parameterized by fully connected neural networks, whose input encoding step is critical to the network training. Instead of using frequency encoding in the first NeRF, Instant-NGP (Müller et al., 2022) proposed a versatile multiresolution hash-based encoding method, which allows a smaller network in NeRF to parameterize the volumetric information without losing the quality but also accelerates the convergence of NeRF training.

2.2. Detector

Detectron2 (Wu et al., 2019) is Facebook AI Research's next-generation library, which provides state-of-the-art segmentation and detection algorithms. It encapsulates different parts of computer vision tasks into modules, thus easily importable and modulable by researchers. Similar to Neural-Sim, we use it to create datasets, load the RetinaNet and compute the loss and metrics for the downstream object detection task.

2.3. Methodology for Augmentation and Optimization

As shown in Figure 1, the role of our NeRF is a novel view image generator with some prescribed rendering parameters. The synthesized images are subsequently fed to the down-

stream computer vision (CV) task as the training dataset. The loss on the validation dataset is computed as an optimizing criterion and backpropagated through the whole pipeline to obtain $\frac{\partial L_{val}}{\partial \psi}$, followed by an optimizer to optimize the rendering parameters distribution. More efficient training for the downstream CV task under a limited input occasion is expected from this pipeline. The NeRF, as a view generator here, can not only conduct reliable multi-view synthesis for uniform data augmentation but also provide the downstream-preferred dataset.

The main contribution of Neural-Sim (Ge et al., 2022) is making the whole pipeline differentiable by a reparameterization trick to quantify the rendering parameters sampling step in NeRF (e.g., pose, scaling, and illumination) and, thus, making the pipeline optimizable. In reparametrization, it uses Gumble-softmax-based categorical distribution for a differentiable sampling of rendering inputs of NeRF. Additionally, unlike typical REINFORCE-based optimization algorithms that simulate the traditional rendering process as a black box, rendering novel view images from NeRF with given viewing directions is differentiable because of the MLP structure of NeRF and explicit computation of output image with a raymarching-based volumetric rendering algorithm. According to Neural-Sim, we use bi-level optimization to solve the problem:

$$\min_{\psi} \mathcal{L}_{\text{val}}\left(\hat{\theta}(\psi)\right); \quad \text{s.t. } \hat{\theta}(\psi) \in \arg\min_{\theta} \mathcal{L}_{\text{train}}\left(\theta, \psi\right),$$

where θ denotes the parameters of the downstream CV model, ψ denotes the rendering parameters of NeRF (sample pose in this case), L_{train} is the training loss over dataset generated by NeRF, and L_{val} is the validation loss for the specific downstream CV task. For gradient update, we divide the gradient into two terms: ∇_{NeRF} corresponds to the backpropagation through Instant-NeRF, and ∇_{TV} corresponds to the approximate backpropagation of the downstream CV training and validation. Therefore, the objective of the bi-level optimization can be expressed as the following ($\mathcal{H}$ denotes Hessian matrix) :

$$\nabla \left. \frac{\partial \mathcal{L}_{val}(\hat{\theta}(\psi))}{\partial \psi} \right|_{\psi=\psi_t} \approx -\nabla_{NeRF} \nabla_{TV}$$

with

$$\nabla_{NeRF} = \frac{\partial}{\partial \psi} \left[\frac{\partial \mathcal{L}_{train}\left(\hat{\theta}\left(\psi_t\right), \psi\right)}{\partial \theta} \right]^T \Bigg|_{\psi=\psi_t}$$

and

$$\nabla_{TV} = \mathcal{H}\left(\hat{\theta}\left(\psi_t\right), \psi\right)^{-1} \frac{d\mathcal{L}_{val}\left(\hat{\theta}\left(\psi_t\right)\right)}{d\theta}.$$

In this work, the NeRF part from Neural-Sim is detached and replaced by Instant-NeRF based on a faithful PyTorch-version implementation (Tang, 2022). Even though the NeRF training is agnostic from the optimization part, we need to integrate Instant-NeRF in the pipeline for downstream dataset image synthesis and a differentiable inference to guarantee the whole pipeline backpropagation.

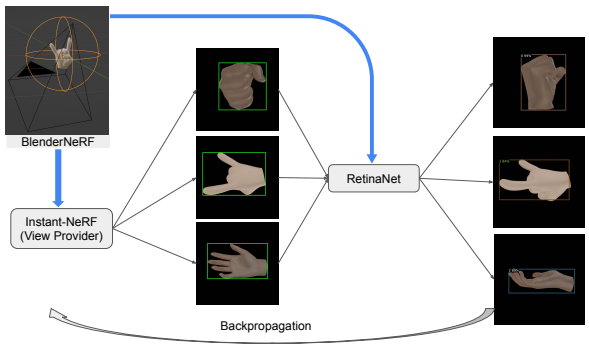

Figure 1. Overview of the whole pipeline. We use BlenderNeRF to generate data for Instant-NeRF and the downstream CV task (here RetinaNet). All images can be automatically labeled by a simple image processing algorithm because of the simple background. Instant-NeRF generates training data for the object to be optimized. Finally, the RetinaNet accomplish the detection task, the loss is then backpropagated to update the sample pose of Instant-NeFR.

3. Results

3.1. Dataset Generation

Our dataset for Instant-NeRF training is generated with BlenderNeRF (Raafat, 2022), an add-on for Blender to support procedural image synthesis from 3D object files. Training data is generated by specifying a camera motion path on an empty sphere mesh and constraining the viewing direction to the object center to achieve a random pose dataset. Similarly, the training and validation dataset for Neural-Sim is created by BlenderNeRF. However, the camera's path is only a circle because the pose varies in one dimension in our setting. The aabb scale, the most important parameter during the Instant-NeRF training, is set to 4. In addition, the background of all images is set to black for training the NeRF model and generating labels of images for the downstream CV task.

3.2. Instant-NeRF Training

We generate 100/100 training/testing data with random poses in BlenderNeRF and train the Instant-NeRF with bound 4.0 and scale 0.8, deploying cuda-ray. As a result, a fast convergence is achieved, as shown in Table 1. In addition, this result is compared with First-NeRF in Table 2,

which uses frequency-encoding to validate the training of Instant-NeRF. We give out the loss, PSNR, and LPIPS of the two NeRF variants at the same epoch and for three different gestures. Instant-NeRF can synthesize relatively higher quality image within only 100 epochs, while frequency-encoding NeRF need further training to export a reliable model for Neural-Sim evaluation.

Table 1. Performance of Instant-NeRF for three gestures

GESTURE	EPOCHS	PSNR	LPIPS
FIST	50	39.61	0.03
	100	42.13	0.02
(TEST)	100	38.81	0.04
YEAH	50	40.56	0.01
	100	42.07	0.01
(TEST)	100	40.18	0.01
CASUAL	50	42.62	0.01
	100	44.78	0.01
(TEST)	100	39.03	0.02

Table 2. Performance of convergence of Instant-NeRF(Multigrid-Hash Encoding) and First-NeRF(Frequency Encoding)

ENCODING	EPOCHS	PSNR	LPIPS
HASH	100	42.07	0.01
	150	42.03	0.01
(TEST)	150	42.13	0.02
FREQUENCY	100	32.80	0.13
	150	33.69	0.12
(TEST)	150	27.56	0.16

3.3. Instant-NeRF Integration

To verify the successful integration of Instant-NeRF in the Neural-Sim pipeline, we first run the bi-level optimization algorithm on YCB box2 (object-id: 2), where Instant-NeRF generates training data for the downstream detection task. Then, the testing dataset is generated with BlenderNeRF with a different seed as ground truth. Finally, both training and testing datasets for the detector are automatically labeled with a bounding box through OpenCV image processing algorithms, modified from the original version in Neural-Sim to handle inputs with slight background noise. Table 3 shows the average precision (AP), AP_{50}, and AP_{75} for all six categories of our work and Neural-Sim. Similar to the original, our Neural-Sim pipeline with Instant-NeRF could not detect one category very well, so the final average precision is slightly lower than the benchmark. Nevertheless, Instant-NeRF can provide same level of reliability as the original NeRF trained in Neural-Sim for downstream tasks.

Table 3. Results of bi-level optimization for YCB dataset. The original Neural-Sim is the benchmark.

METHOD	EPOCHS	AP	AP_{50}	AP_{75}
BENCHMARK	1	28.49	31.82	30.86
OURS	1	38.43	42.63	41.22
BENCHMARK	5	72.70	82.09	79.40
OURS	5	63.01	70.98	67.38
BENCHMARK	20	75.99	87.22	83.38
OURS	20	75.20	82.12	79.47

3.4. Experiment on Hand Gestures

After verifying the successful integration of Instant-NeRF, the bi-level optimization pipeline is conducted on our self-generated hand gesture blender dataset. We substitute the first three objects of the YCB dataset with three hand gestures: fist, yeah, casual, and finally, change the structure of the downstream task model correspondingly. Table 4 shows the AP, AP_{50}, and AP_{75} of our network.

Table 4. Results of bi-level optimization for hand gesture dataset. Lowest AP among all categories highly affects metrics value.

EPOCHS	AP	AP_{50}	AP_{75}	LOWEST AP
1	75.41	76.41	76.41	66.79
2	92.48	93.08	93.08	86.67
5	97.56	98.18	98.18	96.23

4. Discussion

As to the optimization part, the gradient of loss on validation data with respect to pose distribution can be backpropagated, taking advantage of Gumble-softmax categorical distribution and differentiable rendering of NeRF. The gradient backpropagation computation is primarily inherited from Neural-Sim, where we realize the with-gradient inference of Instant-NeRF with autograd function in PyTorch. However, we did not noticeably optimize the poses, which are highly downstream task-dependent. Specifically, we also could not obtain an observable pose optimization when we reproduced the same experiment from the source code of Neural-Sim. The categorical distribution slightly changed during the first 20 epochs on both experiments based on our implementation and Neural-Sim's, even though the autograd computation worked well.

Apart from this, we did not reproduce the optimization on other rendering parameters mentioned in the Neural-Sim article (like scaling and illumination), which, from our perspective, is not optimizable in the current setting. Suppose a similar parameterized sampling method is applied to other rendering parameters. In that case, the gradient of NeRF in

the Neural-Sim article can not be backpropagated with respect to these rendering parameters because they can not be explicitly calculated into NeRF's inputs. Despite this doubt, this pipeline can still be helpful to NeRF's variants, which can take versatile rendering parameters as input, or even some NeRF training hyperparameters with given optimizing criterion.

With this inspirational idea, our future anticipation on this project is searching for other NeRF variants to primarily take advantage of this on-demand optimization concept to increase the efficiency of NeRF training continuously. This is also our first motivation for exploring the combination of Neural-Sim and Instant-NeRF. Instant-NeRF had spared much effort in tuning hash functions and encoding structures. Particularly, this task-dependent optimization can facilitate the tuning of hash function parameters during the training process. Namely, using reparameterization trick on multigrid-hashing parameters sampling and conducting Bayesian optimization to reasonably trade-off parameters searching and NeRF's output quality (like common exploiting-exploration dilemma setting).

5. Summary

This course project aims to obtain theoretical and practical knowledge of NeRFs and their variants, including understanding the rendering pipeline of (Instant-) NeRF, training configurations, dataset generation, and pre-processing. The prime challenges encountered include the Instant-NeRF integration and training configuration adjustment. Remarkably, even though (Instant-) NeRFs have many faithful implementations, they are all highly customized. Thus, instead of a trivial replacement of pertaining functions, a pretty involved modification is anticipated here, which requires a thorough understanding of the NeRF's source code. The implementation detail is elaborated in our project repository. This work integrates Instant-NeRF in the Neural-Sim's optimization pipeline and works well on our self-designed hand gesture dataset. This inspirational result gives the possibility of quick on-demand data augmentation with NeRFs, which also encourages more efficient training of NeRF. We anticipate more NeRF variants integration with this pipeline for versatile applications concerning different downstream tasks.

References

Ge, Y., Behl, H., Xu, J., Gunasekar, S., Joshi, N., Song, Y., Wang, X., Itti, L., and Vineet, V. Neural-sim: Learning to generate training data with nerf, 2022. URL https://arxiv.org/abs/2207.11368.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

Müller, T., Evans, A., Schied, C., and Keller, A. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

Raafat, M. Blendernerf. https://github.com/maximeraafat/BlenderNeRF/, 2022.

Tang, J. Torch-ngp: a pytorch implementation of instant-ngp, 2022. https://github.com/ashawkey/torch-ngp.

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

Yen-Chen, L. Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020.