RBE 549 Project 2 (Phase 2): NeRF- Neural Radiance Fields for View Synthesis

UdayGirish Maradana
Robotics Engineering (MS)
Worcester Polytechnic Institute
Worcester, MA 01609
Email: umaradana@wpi.edu

Pradnya Sushil Shinde
Robotics Engineering (MS)
Worcester Polytechnic Institute
Worcester, MA 01609
Email: pshinde1@wpi.edu

***Abstract*—The following report consists of a detailed analysis of a deep learning approach to the 3D reconstruction of a scene using a method known as, Neural Radiance Field (NeRF) which represents a scene using a fully connected (non-convolutional) deep network, whose input is a single continuous 5D coordinate (the spatial location (x, y, z) and viewing direction (θ, ϕ)) and whose output is the volume density and view-dependent emitted radiance at that spatial location**

Keywords: 3D Reconstruction, Fully Connected Deep Network, Spatial Location, Radiance, Volume Density

I. Introduction

NeRF presents a novel view synthesis approach that optimizes parameters of a continuous 5D scene representation to minimize the error of rendering a set of captured images. NeRf synthesizes views by querying 5D coordinates along camera rays and uses classic volume rendering techniques to project the output colors and densities into an image. Because volume rendering is naturally differentiable, the only input required to optimize our representation is a set of images with known camera poses. They describe how to effectively optimize neural radiance fields to render photorealistic novel views of scenes with complicated geometry and appearance, and demonstrate results that outperform prior work on neural rendering and view synthesis. The physical interpretation of NeRF is shown in Figure 1.

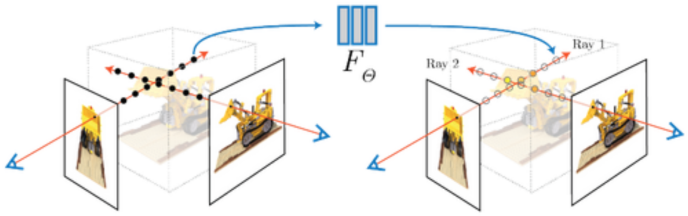

Fig. 1: NeRF

Usual Nerf Pipeline:

II. Methodology

The images are synthesized as described below:

1) Sample the 5D coordinates (location and viewing direction) along camera rays.

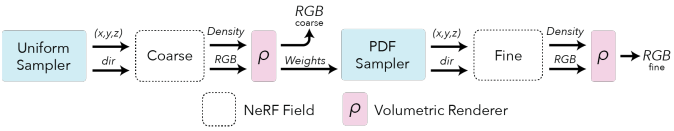

Fig. 2: Nerf Pipeline Explanation - Nerf Studio

2) Feed the locations into a Multilayer Perception (MLP) to produce a color and volume density.
3) Use rendering techniques to composite these values into an image.
4) Since the rendering function is differentiable, the scene representation can be optimized by minimizing the residual between synthesized and ground truth observed images.

We approximate this continuous 5D scene representation with an MLP network $F\theta : (x, d) \longrightarrow (c, \sigma)$, where σ is the volume density and c is the RGB color to be predicted as a function of location (x) and viewing direction(d) and optimize its weights θ to map from each input 5D coordinate to its corresponding volume density and directional emitted color.

A. Volume Rendering with Radiance Fields

The 5D neural radiance field represents a scene as the volume density and directional emitted radiance at any point in space. We render the color of any ray passing through the scene using principles from classical volume rendering. The volume density $\sigma(x)$ can be interpreted as the differential probability of a ray terminating at an infinitesimal particle at location x. The expected color $C(r)$ of camera ray $r(t) = o + td$ with near and far bounds t_n and t_f is:

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t), d)dt,$$

$$\text{where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(r(s))ds\right)$$

The function $T(t)$ denotes the accumulated transmittance along the ray from t_n to t, i.e., the probability that the ray travels from tn to t without hitting any other particle. Rendering a view from our continuous neural radiance field

requires estimating this integral $C(r)$ for a camera ray traced through each pixel of the desired virtual camera. To numerically estimate this continuous integral using quadrature, a stratified sampling approach is followed where we partition $[t_n, t_f]$ into N evenly-spaced bins and then draw one sample uniformly at random from within each bin:

$$t_i \sim U\left(t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)\right).$$

Stratified sampling enables us to represent a continuous scene representation because it results in the MLP being evaluated at continuous positions throughout the optimization.

$$\hat{C}(r) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \delta_i))c_i,$$

where $\Delta i = t_i + 1 - t_i$ is the distance between adjacent samples. This function for calculating $C(r)$ from the set of $(c_i, \sigma i)$ values, is trivially differentiable and reduces to traditional alpha compositing with alpha values $\alpha i = 1 - exp(-\sigma i \Delta i)$.

III. Network Architecture and Implementation

Please refer to the network architecture in Figure 2. The architecture and training is exactly same as the original implementation and loosely based on the official Tensorflow implementation itself. It is a combination of simple Linear Layers with ReLU activation of width 256 everywhere with a skip connection. The network is trained with the same specifications as mentioned in the paper with Adam optimizer with learning rate of 5e-4 and exponential decay. For logging the losses and parameters, we used Weights and Biases as it easier for remote logging and viewing.

IV. Results and Discussion

We have basically trained the network on 2 examples (Lego, Ship) and 1 real world example. The two examples are provided from Nerf synthetic dataset which are from blender sim.

The results of the Lego Model are shown in Fig.9 , Fig.10, Fig.11, Fig.12.

The sample results of the ship model are shown in Fig. 13 (We didn't get time to train without positional encoding for this).

We have also trained a real image by moving around the object (Here a coffee cup) and referred code for Colmap-JSON [5] for converting the video to set of frames and then do SfM and calculate poses. Further we removed backgrounds with a library (Script is present in the code base). After doing this we passed it to training even though the results are not great , almost the shape and color is present and correct. This might be due to the vibrations or movement while we are taking pictures and artifacts from the background removal process. Also the real world experiment novel pose generation is poor. This can be investigated more to get good videos and running feature matching with deep learning features or some other approach. Please check the results in Fig.8,Fig.9.

The real world experiment outputs are shown in Figure 14 and Figure 15. Further the output from NVIDIA instant NGP is shown in Fig.16 Further we calculated the losses such as MSE, SSIM, PSNR and LPIPS. Initially we trained the model with LPIPS backend of Alex because of that we got a bit bad result as it gives best score but it is not same as traditional perceptual similarity. Later, we trained with LPIPS backend as VGG which gave comparable results to paper.

The experiment names (Legend Names in Plots) are as follows:

1) coffe_cup_exp: Experiment with Real world images taken (Coffee Cup)
2) ship_wb_f: Experiment with Ship model with positional encoding
3) lego_wopos: Experiment with Lego model without positional encoding
4) lego_wpos: Experiment with Lego model with positional encoding

Please find some of the plots here (Fig.3,4,5,6,7,8), full level information is present in the table.

TABLE I: Test Metrics (20 Images)

Model/LossName	Lego(WP)	Lego(WOP)	Ship(WP)	Coffee Cup
MSE	0.00075	0.002691	0.001231	0.01853
PSNR	31.267	25.701	29.096	17.321
SSIM	0.9882	0.965	0.9369	0.942
LPIPS	0.000514	0.000076	0.0007206	0.0003348

TABLE II: Train Metrics

Model/LossName	Lego(WP)	Lego(WOP)	Ship(WP)	Coffee Cup
MSE	0.002908	0.0958	0.002672	0.0255
PSNR	33.098	26.014	30.256	26.504

V. Challenges Future Direction

A. Challenges/Learnings

1) Initially we faced a bit confusion regarding tuning the parameters and understand the impact of NDC and also to find the near and far bounds. But we used the values from original implementation later. Figuring out Near and far for custom datasets is still a question for us and the impact on it.
2) The use of positional encoding is clearly showed a positive effect on the overall render.
3) There were issues with taking high memory but that was resolved after following the implementation of nerf pytorch.
4) The training times were too long like 5-6 hours especially for real world we didn't get time to complete a full-fledged training that's why the results are poor.

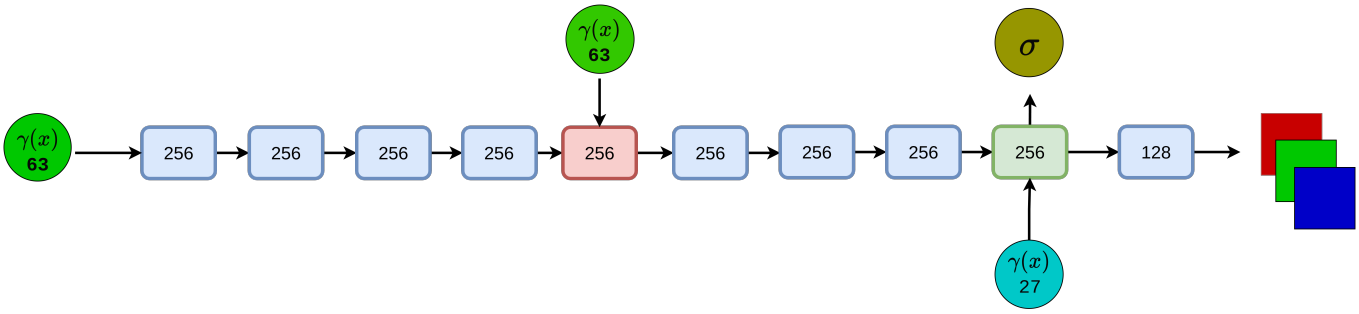

Fig. 3: NeRF Architecture - Similar to Original Paper

5) Even though we tried our best to get the good real world experiment, because of the movement when we are doing and background objects there were issues with both COLMAP not able to finding convergence or taking more than 2 hours for 300 images to find camera poses. Also the output poses were not accurate. Further we have removed the background after extracting the camera poses to avoid learning outside features it improved the performance a bit and the loss was decreasing but still because of the artifacts from background removal it was hard for us to get a proper video output. If done in a lab setting it should be good. We even tried passing the COLMAP transforms to NVIDIA Instant Neural graphics primitives and check the output but that also wasn't that great. It seems the data collection itself should have been done more carefully.

B. *Future Directions*

1) Understand more about the impact of different factors and play with hyperparameters.
2) Try to understand how to capture a good real life experiment and experiment with it.

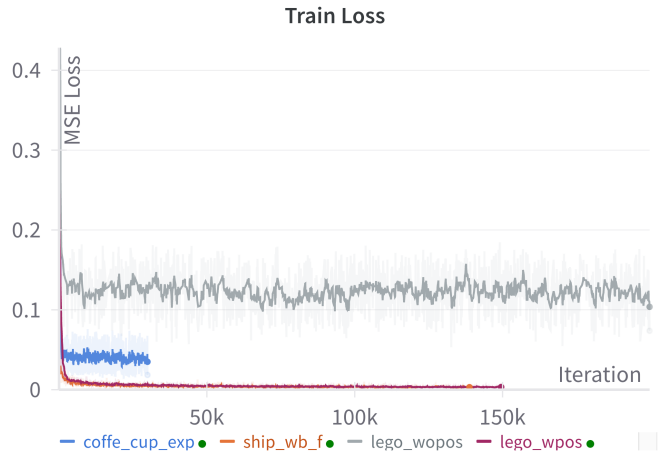

Fig. 4: NeRF - Train MSE Loss

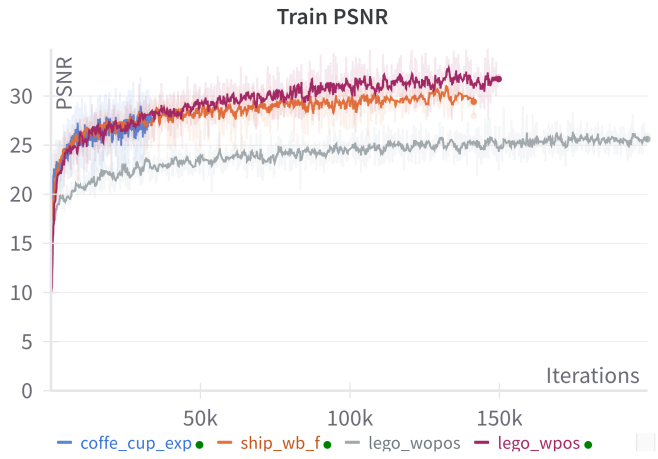

Fig. 5: NeRF - Train PSNR

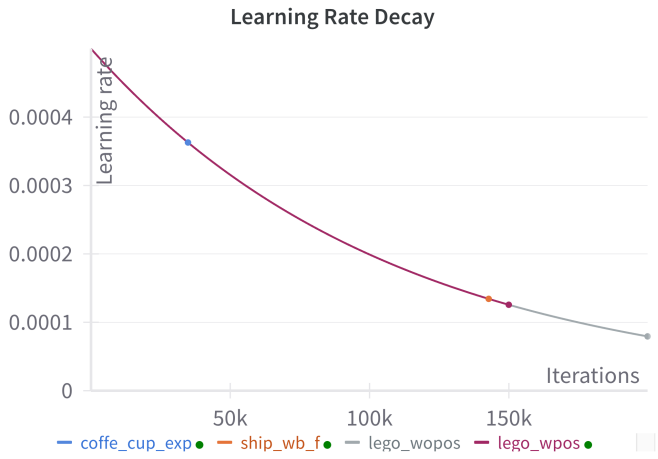

Fig. 6: NeRF - Learning Rate Decay

REFERENCES

[1] Mildenhall, B., Srinivasan, P., Tancik, M., Barron, J., Ramamoorthi, R. & Ng, R. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. (2020)
[2] NERF Original Repo (https://github.com/bmild/nerf) - Written in Tensorflow by original authors

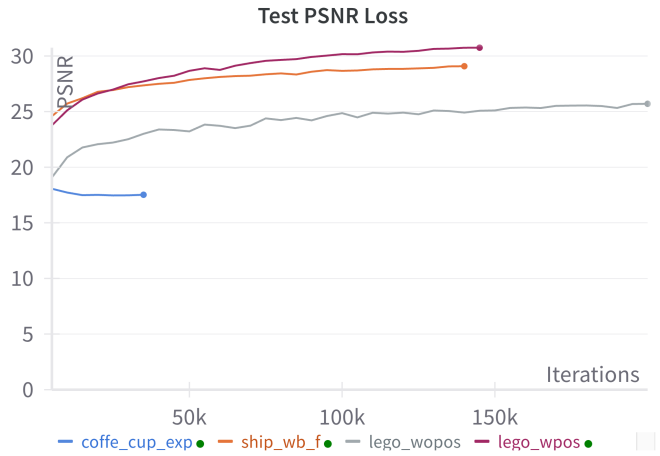

Fig. 7: NeRF - Test PSNR

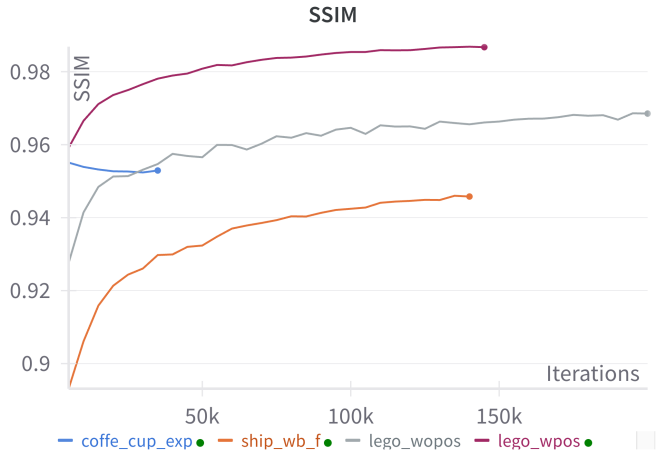

Fig. 8: NeRF - Test SSIM

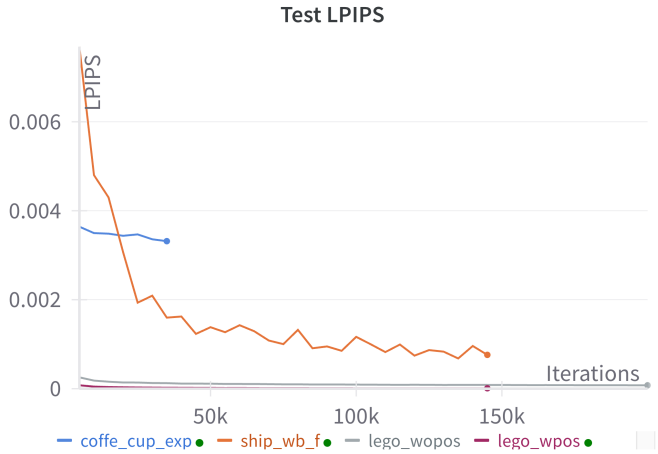

Fig. 9: NeRF - Test LPIPS

[3] NERF With Pytorch (https://github.com/yenchenlin/nerf-pytorch/tree/master) - Pytorch (One of the best cited repos)
[4] NERF Pytorch (Faster Version - https://github.com/krrish94/nerf-pytorch) - Faster implementation based on Repo 2.
[5] NVIDIA Instant NGP (https://github.com/NVlabs/instant-ngp - The Fastest NeRF and extensive library which can render meshes too).
[6] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, O. Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In CVPR, 2018

((a)) GroundTruth

((b)) Output

Fig. 10: NeRF Output for Lego at 10000 Iterations - **Without Positional Encoding**

((a)) GroundTruth

((b)) Output

Fig. 11: NeRF Output for Lego at 10000 Iterations - **With Positional Encoding**

((a)) GroundTruth

((b)) Output

Fig. 12: NeRF Output for Lego at 150000 Iterations - **Without Positional Encoding**

((a)) GroundTruth

((b)) Output

Fig. 13: NeRF Output for Lego at 200000 Iterations - **With Positional Encoding**

((a)) GroundTruth

((b)) Output

Fig. 14: NeRF Output for Ship model at 130000 Iterations - **With Positional Encoding**

((a)) GroundTruth

((b)) Output

Fig. 15: NeRF Output for Real model at 10000 Iterations - **With Positional Encoding**

((a)) GroundTruth

((b)) Output

Fig. 16: NeRF Output for Real model at 25000 Iterations - **With Positional Encoding**

Fig. 17: NVIDIA Instant NGP Output for Real World experiment