RBE 549 Project 2
Phase 2 - Neural Radiance Fields

Niranjan Kumar Ilampooranan
MS Robotics Graduate Student
Worcester Polytechnic Institute

Thanikai Adhithiyan Shanmugam
MS Robotics Graduate Student
Worcester Polytechnic Institute

***Abstract*—For this phase of Project 2, the objective is the replication of the original implementation of Neural Radiance Fields. Given multiple images of two objects from different views, the goal is to render a three-dimensional scene representation of the same. The network architecture used, along with the obtained results is discussed in this report.**

Introduction

Neural Radiance Fields by Mildenhall et al. [1] is one such work that took the world by storm. From a set of 2D images, this neural network enables 3D reconstruction of the scene, essentially synthesizing novel and moderately realistic views. Given a sparse set of input images, the goal is to generate volume density and view-dependent emitted radiance by optimizing the underlying continuous volumetric scene function). This is well-captured in the image provided below [1].

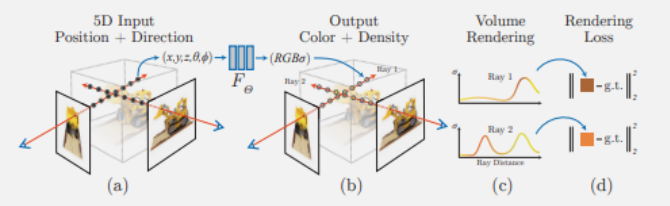

Fig. 1. Overview of NeRF functionality

A multilayer perception is used to approximate this function discussed above, as opposed to using conventional layers. To visualize or render the output Neural radiance fields, the camera rays are passed through the scene to generate 3D points. With these points along with the viewing directions passed as input to the network, the volume density with RGB values is obtained, after which classical volume rendering methods are employed to generate a viewable output.

As part of this project, the lego set and the ship data set used in the original set is to be fed to the network for training and then rendering a complete view of these objects. The crucial points of discussion to be followed in further sections, are given below.

- The employed network architecture used for the implementation
- Results obtained using the dataset from the implemented model along with challenges faced during the process
- Testing the performance of the model on a dataset curated by us

Network Architecture

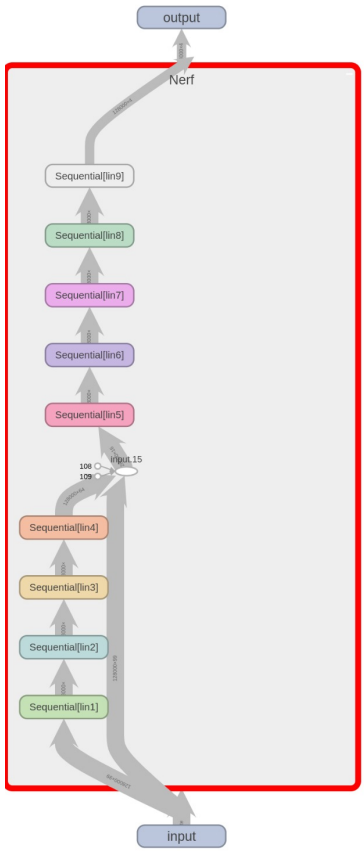

Fig. 2. Network architecture of implemented NeRF

Above is the model architecture implemented for this project, inspired from the Nerf-PyTorch repository. The architecture has 8 layers where the first layer inputs the encoded ray sample of size (3 + 3*2*$n_{encoders}$) and uses a filter of size 256. The next 2 hidden layers are convolved with the filter size 256. In the 4^{th} layer, the input is concatenated with the

output of the 3rd layer. The next 3 layers are convolved with fully connected layer of filter size 256,256,128 respectively. The final layer is a fully connected layer which produces the R,G,B, and α channel values.

Furthermore, a tinyNeRF model architecture has also been implemented which helps to verify the feasibility of the data generation and to ensure faster yet feasible results. The tiny nerf has 3 layers the last being the fully connected layer, which can be seen in the figure below.

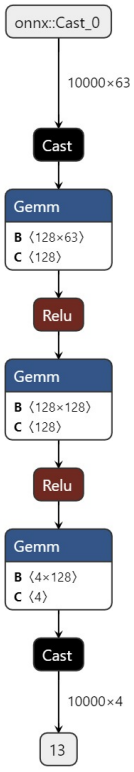

Fig. 3. Network architecure of tinyNeRF

Results and Discussion

The paper exercises NeRF model in pytorch and tries to compute it for different NeRF datasets. There are 4 datasets namely Lego-encoded, Lego-without encoded, ship and a custom data. The training is done for 10,000 epochs with sample ray size of 1024 and batch size of 32. The lego model has been trained with psoitonal encoding and without position encoding. For training, the PSNR loss function for each dataset is plotted.

Coming to the test results, the PSNR and SSIM error have been plotted and the average has been tabulated to know the feasibility of the network. A .gif file has been rendered using the image writer function.

Compared to unencoded, encoding is done in Fourier space. Hence it has better results.

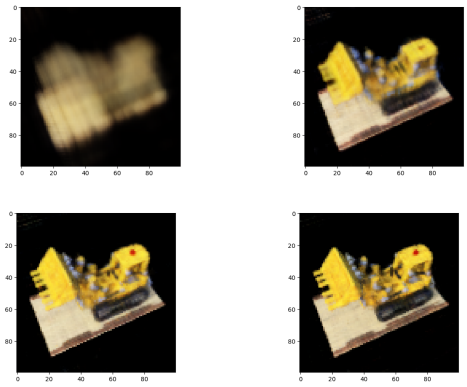

Fig. 4. Training Images- Lego Encoded

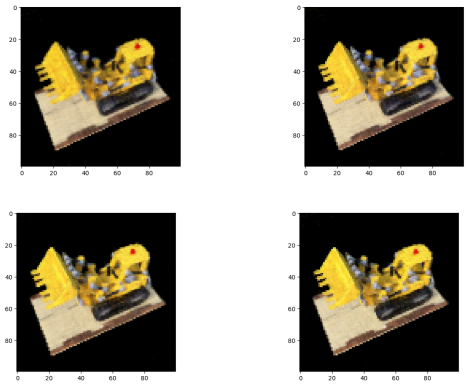

Fig. 5. Training Images- Lego Encoded

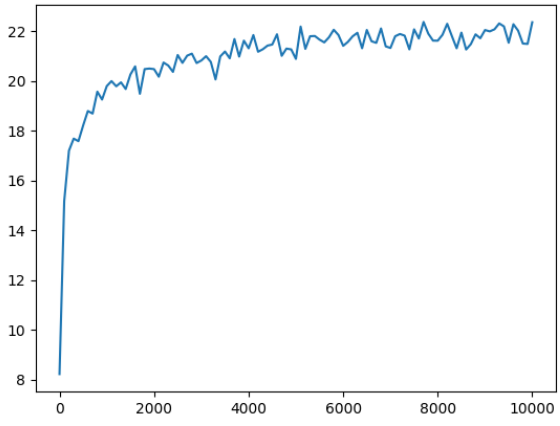

Fig. 6. Training PSNR - Lego Encoded

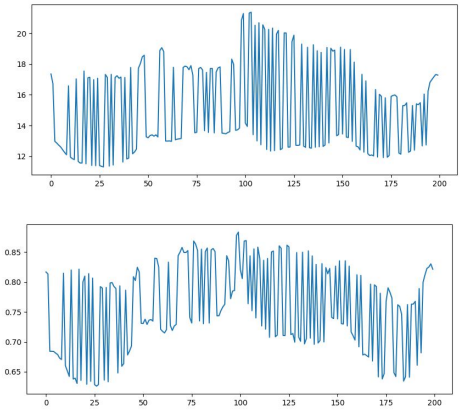

Fig. 7. Training Images- Lego Encoded

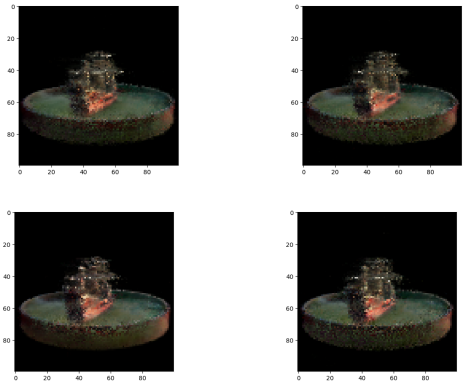

Fig. 10. Training Images- Ship

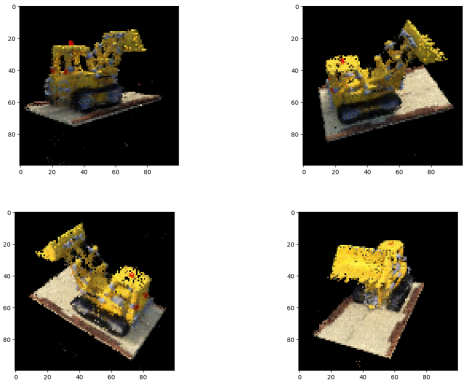

Fig. 8. Testing Output- Lego Encoded

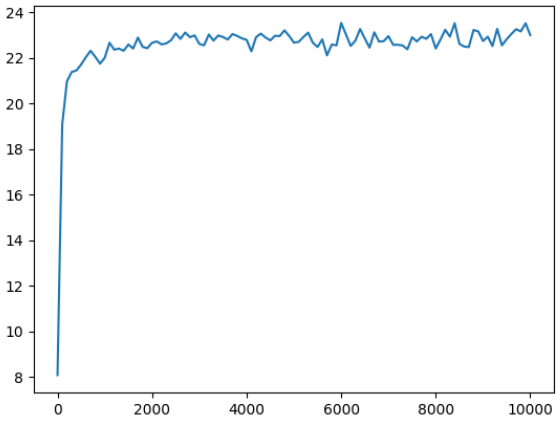

Fig. 11. Training PSNR - Ship

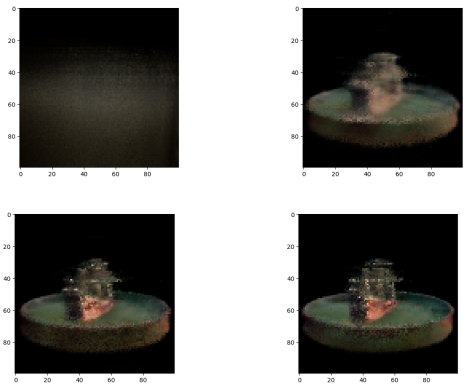

Fig. 9. Training Images- Ship

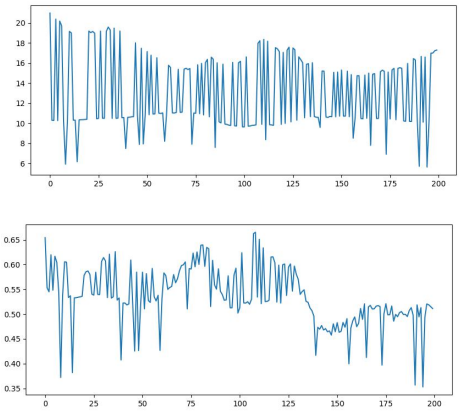

Fig. 12. Training Images- Ship

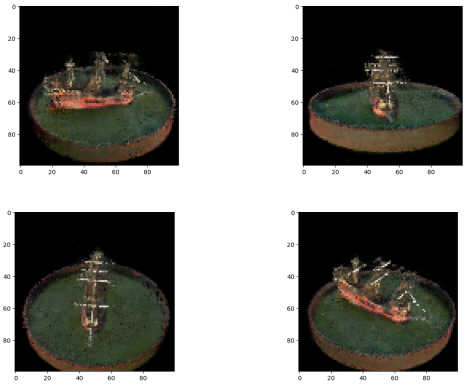

Fig. 13. Testing Output- Ship

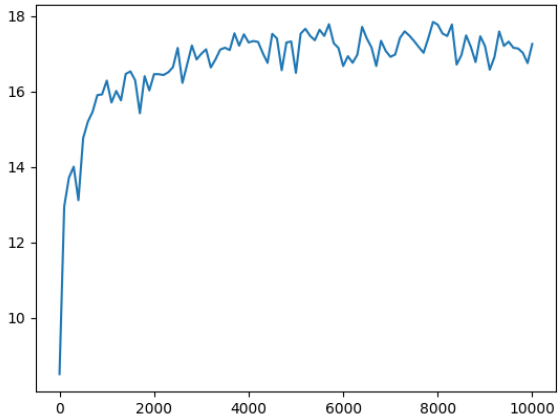

Fig. 16. Training PSNR - Lego Not Encoded

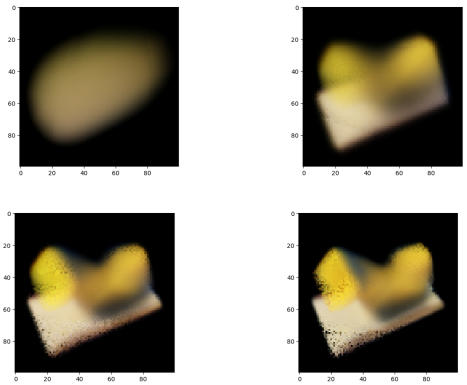

Fig. 14. Training Images- Lego Not Encoded

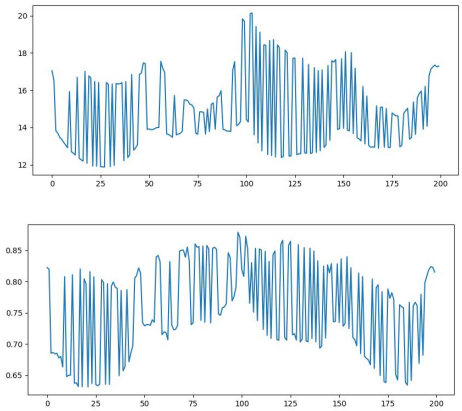

Fig. 17. Training Images- Lego Not Encoded

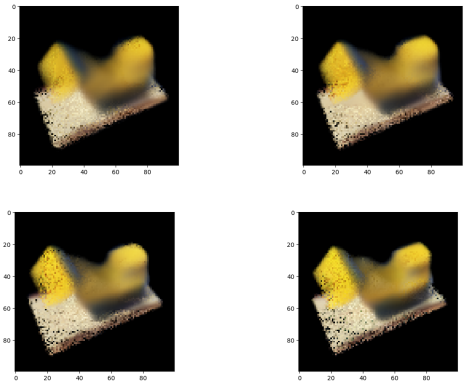

Fig. 15. Training Images- Lego Not Encoded

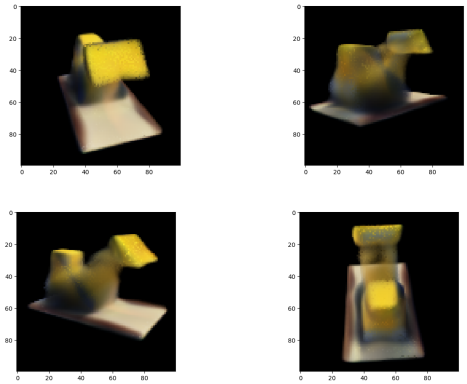

Fig. 18. Testing Output- Lego Not Encoded

	Mean PSNR	Mean SSIM
Lego encoded	15.6	0.69
Lego unencoded	13.2	0.76
Ship encoded	15.2	0.55
Custom	10	0.4

TABLE I

NeRF on Custom Dataset

As part of extra credit, the same procedure used for synthesising novel views around LEGO miniature bulldozer and miniature ship is to be done on a custom dataset. A plushie of Piplup (product of Pokémon) is considered for the same, sample images of which are displayed below. To this extent, a total of 145 images of Piplup is captured to render a complete 360° view of it. This was done on OnePlus 7T camera with the following settings in place (focal length: 4.76mm, Aperture: f/1.6, Exposure time: 1/25, ISO: 100)

Fig. 19. Pictures from Custom Dataset - Piplup (Pokémon)

Given below is the procedure used to obtain the ground truth for training our NeRF model.

- After the photo capture, ensuring the object is in focus and has enough illumination
- Feature extraction and Feature matching with all the images on COLMAP [2]
- Sparse reconstruction and Bundle Adjustment on COLMAP with some tweak in settings to obtain nominal results
- Export the model data (camera poses) to extract the camera intrinsics and extrinsics though colmap2nerf.py from Instant NGP [3] in the form of transforms.json file as in LEGO and Ship dataset

Following this procedure detailed above, helped to obtain the camera poses. This is given in the figure below. During this process, some challenges were faced while performing reconstruction which include - a) imperfect focus that led to issues during reconstruction (only part of the model was reconstructed), b) lighting conditions which led to part of the model missing during background removal, and c) generated quality in the COLMAP setting was reduced from High to Medium along with tweaking of settings such as triangulation.

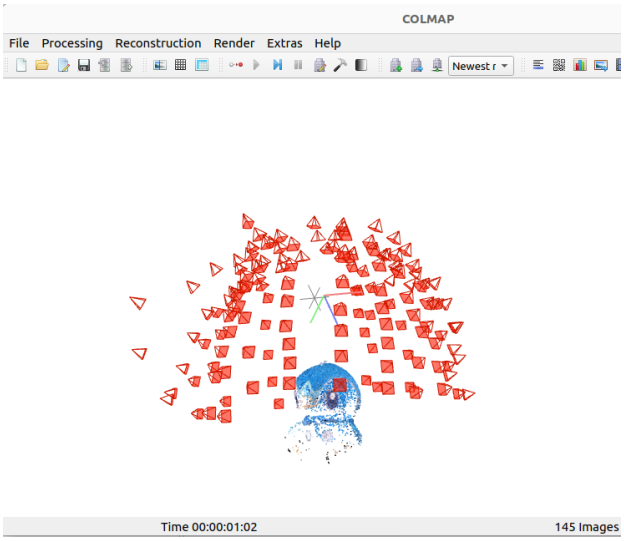

Fig. 20. 3D Reconstruction on COLMAP with camera poses used to take the pictures

The images along with generated transforms.json file is uploaded in this link. The test results for this custom dataset is provided below in the form of generated frames and PSNR values.

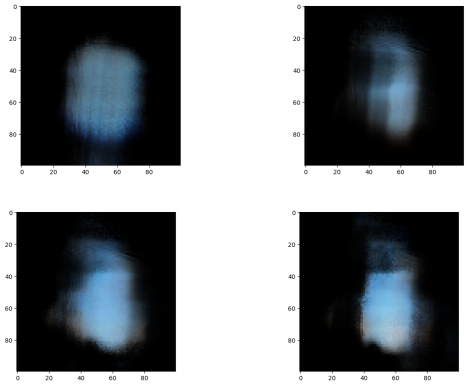

Fig. 21. Training Images- Custom

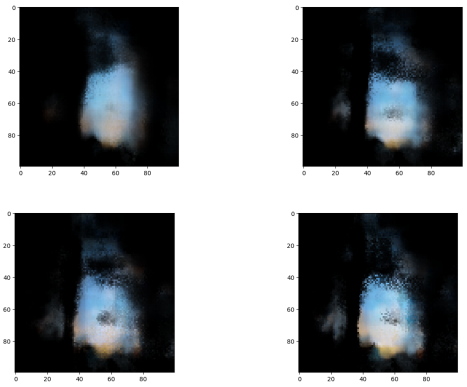

Fig. 22. Training Images- Custom

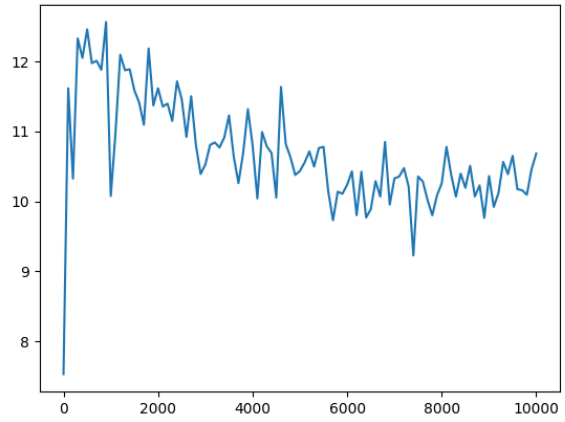

Fig. 23. Training PSNR - Custom

REFERENCES

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," 2020.

[2] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[3] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, pp. 102:1–102:15, July 2022.